Cinder Documentation

Release 25.1.0.dev115

Cinder Contributors

Mar 06, 2025

CONTENTS

CHAPTER

ONE

WHAT IS CINDER?

Cinder is the OpenStack Block Storage service for providing volumes to Nova virtual machines, Ironic bare metal hosts, containers and more. Some of the goals of Cinder are to be/have:

- **Component based architecture**: Quickly add new behaviors
- **Highly available**: Scale to very serious workloads
- **Fault-Tolerant**: Isolated processes avoid cascading failures
- **Recoverable**: Failures should be easy to diagnose, debug, and rectify
- **Open Standards**: Be a reference implementation for a community-driven api

CHAPTER

TWO

FOR END USERS

As an end user of Cinder, youll use Cinder to create and manage volumes using the Horizon user interface, command line tools such as the python-cinderclient, or by directly using the REST API.

2.1 Tools for using Cinder

- Horizon: The official web UI for the OpenStack Project.
- OpenStack Client: The official CLI for OpenStack Projects. You should use this as your CLI for most things, it includes not just nova commands but also commands for most of the projects in OpenStack.
- Cinder Client: The **openstack** CLI is recommended, but there are some advanced features and administrative commands that are not yet available there. For CLI access to these commands, the **cinder** CLI can be used instead.

2.2 Using the Cinder API

All features of Cinder are exposed via a REST API that can be used to build more complicated logic or automation with Cinder. This can be consumed directly or via various SDKs. The following resources can help you get started consuming the API directly.

- Cinder API
- *Cinder microversion history*

CHAPTER

THREE

FOR OPERATORS

This section has details for deploying and maintaining Cinder services.

3.1 Installing Cinder

Cinder can be configured standalone using the configuration setting `auth_strategy = noauth`, but in most cases you will want to at least have the Keystone Identity service and other OpenStack services installed.

3.1.1 Cinder Installation Guide

The Block Storage service (cinder) provides block storage devices to guest instances. The method in which the storage is provisioned and consumed is determined by the Block Storage driver, or drivers in the case of a multi-backend configuration. There are a variety of drivers that are available: NAS/SAN, NFS, iSCSI, Ceph, and more.

The Block Storage API and scheduler services typically run on the controller nodes. Depending upon the drivers used, the volume service can run on controller nodes, compute nodes, or standalone storage nodes.

For more information, see the *Configuration Reference*.

Prerequisites

This documentation specifically covers the installation of the Cinder Block Storage service. Before following this guide you will need to prepare your OpenStack environment using the instructions in the OpenStack Installation Guide.

Once able to Launch an instance in your OpenStack environment follow the instructions below to add Cinder to the base environment.

Adding Cinder to your OpenStack Environment

The following links describe how to install the Cinder Block Storage Service:

Warning

For security reasons **Service Tokens must be configured** in OpenStack for Cinder to operate securely. Pay close attention to the *specific section describing it:*. See https://bugs.launchpad.net/nova/+bug/2004555 for details.

Cinder Block Storage service overview

The OpenStack Block Storage service (Cinder) adds persistent storage to a virtual machine. Block Storage provides an infrastructure for managing volumes, and interacts with OpenStack Compute to provide volumes for instances. The service also enables management of volume snapshots, and volume types.

The Block Storage service consists of the following components:

cinder-api
: Accepts API requests, and routes them to the `cinder-volume` for action.

cinder-volume
: Interacts directly with the Block Storage service, and processes such as the `cinder-scheduler`. It also interacts with these processes through a message queue. The `cinder-volume` service responds to read and write requests sent to the Block Storage service to maintain state. It can interact with a variety of storage providers through a driver architecture.

cinder-scheduler daemon
: Selects the optimal storage provider node on which to create the volume. A similar component to the `nova-scheduler`.

cinder-backup daemon
: The `cinder-backup` service provides backing up volumes of any type to a backup storage provider. Like the `cinder-volume` service, it can interact with a variety of storage providers through a driver architecture.

Messaging queue
: Routes information between the Block Storage processes.

The default volume type

Since the Train release, it is required that each volume must have a *volume type*, and thus the required configuration option `default_volume_type` must have a value. A system-defined volume type named `__DEFAULT__` is created in the database during installation and is the default value of the `default_volume_type` configuration option.

You (or your deployment tool) may wish to have a different volume type that is more suitable for your particular installation as the default type. This can be accomplished by creating the volume type you want using the Block Storage API, and then setting that volume type as the value for the configuration option. (The latter operation, of course, cannot be done via the Block Storage API.)

The system defined `__DEFAULT__` volume type is a regular volume type that may be updated or deleted. There is nothing special about it. It only exists because there must always be at least one volume type in a cinder deployment, and before the Block Storage API comes up, there is no way for there to be a volume type unless the system creates it.

Given that since the Victoria release it is possible to set a default volume type for any project, having a volume type named `__DEFAULT__` in your deployment may be confusing to your users, leading them to think this is the type that will be assigned while creating volumes (if the user doesnt specify one) or them specifically requesting `__DEFAULT__` when creating a volume instead of the actual configured default type for the system or their project.

If you dont wish to use the `__DEFAULT__` type, you may delete it. The Block Storage API will prevent deletion under these circumstances:

- If `__DEFAULT__` is the value of the `default_volume_type` configuration option then it cannot be deleted. The solution is to make a different volume type the value of that configuration option.

- If there are volumes in the deployment of the `__DEFAULT__` type, then it cannot be deleted. The solution is to retype those volumes to some other appropriate volume type.

Cinder Installation Guide for openSUSE and SUSE Linux Enterprise

This section describes how to install and configure storage nodes for the Block Storage service. For simplicity, this configuration references one storage node with an empty local block storage device. The instructions use `/dev/sdb`, but you can substitute a different value for your particular node.

The service provisions logical volumes on this device using the *LVM* driver and provides them to instances via *iSCSI* transport. You can follow these instructions with minor modifications to horizontally scale your environment with additional storage nodes.

Install and configure controller node

This section describes how to install and configure the Block Storage service, code-named cinder, on the controller node. This service requires at least one additional storage node that provides volumes to instances.

Prerequisites

Before you install and configure the Block Storage service, you must create a database, service credentials, and API endpoints.

1. To create the database, complete these steps:

   1. Use the database access client to connect to the database server as the `root` user:

      ```
      $ mysql -u root -p
      ```

   2. Create the `cinder` database:

      ```
      MariaDB [(none)]> CREATE DATABASE cinder;
      ```

   3. Grant proper access to the `cinder` database:

      ```
      MariaDB [(none)]> GRANT ALL PRIVILEGES ON cinder.* TO 'cinder'@
      ↪'localhost' \
        IDENTIFIED BY 'CINDER_DBPASS';
      MariaDB [(none)]> GRANT ALL PRIVILEGES ON cinder.* TO 'cinder'@'%' \
        IDENTIFIED BY 'CINDER_DBPASS';
      ```

      Replace `CINDER_DBPASS` with a suitable password.

   4. Exit the database access client.

2. Source the `admin` credentials to gain access to admin-only CLI commands:

   ```
   $ . admin-openrc
   ```

3. To create the service credentials, complete these steps:

   1. Create a `cinder` user:

```
$ openstack user create --domain default --password-prompt cinder

User Password:
Repeat User Password:
+---------------------+----------------------------------+
| Field               | Value                            |
+---------------------+----------------------------------+
| domain_id           | default                          |
| enabled             | True                             |
| id                  | 9d7e33de3e1a498390353819bc7d245d |
| name                | cinder                           |
| options             | {}                               |
| password_expires_at | None                             |
+---------------------+----------------------------------+
```

2. Add the `admin` role to the `cinder` user:

```
$ openstack role add --project service --user cinder admin
```

Note

This command provides no output.

3. Create the `cinderv3` service entity:

```
$ openstack service create --name cinderv3 \
  --description "OpenStack Block Storage" volumev3

+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | OpenStack Block Storage          |
| enabled     | True                             |
| id          | ab3bbbef780845a1a283490d281e7fda |
| name        | cinderv3                         |
| type        | volumev3                         |
+-------------+----------------------------------+
```

Note

Beginning with the Xena release, the Block Storage services require only one service entity. For prior releases, please consult the documentation for that specific release.

4. Create the Block Storage service API endpoints:

```
$ openstack endpoint create --region RegionOne \
  volumev3 public http://controller:8776/v3/%\(project_id\)s
```

(continues on next page)

(continued from previous page)

```
+--------------+------------------------------------------+
| Field        | Value                                    |
+--------------+------------------------------------------+
| enabled      | True                                     |
| id           | 03fa2c90153546c295bf30ca86b1344b         |
| interface    | public                                   |
| region       | RegionOne                                |
| region_id    | RegionOne                                |
| service_id   | ab3bbbef780845a1a283490d281e7fda         |
| service_name | cinderv3                                 |
| service_type | volumev3                                 |
| url          | http://controller:8776/v3/%(project_id)s |
+--------------+------------------------------------------+

$ openstack endpoint create --region RegionOne \
  volumev3 internal http://controller:8776/v3/%\(project_id\)s

+--------------+------------------------------------------+
| Field        | Value                                    |
+--------------+------------------------------------------+
| enabled      | True                                     |
| id           | 94f684395d1b41068c70e4ecb11364b2         |
| interface    | internal                                 |
| region       | RegionOne                                |
| region_id    | RegionOne                                |
| service_id   | ab3bbbef780845a1a283490d281e7fda         |
| service_name | cinderv3                                 |
| service_type | volumev3                                 |
| url          | http://controller:8776/v3/%(project_id)s |
+--------------+------------------------------------------+

$ openstack endpoint create --region RegionOne \
  volumev3 admin http://controller:8776/v3/%\(project_id\)s

+--------------+------------------------------------------+
| Field        | Value                                    |
+--------------+------------------------------------------+
| enabled      | True                                     |
| id           | 4511c28a0f9840c78bacb25f10f62c98         |
| interface    | admin                                    |
| region       | RegionOne                                |
| region_id    | RegionOne                                |
| service_id   | ab3bbbef780845a1a283490d281e7fda         |
| service_name | cinderv3                                 |
| service_type | volumev3                                 |
| url          | http://controller:8776/v3/%(project_id)s |
+--------------+------------------------------------------+
```

Install and configure components

1. Install the packages:

```
# zypper install openstack-cinder-api openstack-cinder-scheduler
```

2. Edit the /etc/cinder/cinder.conf file and complete the following actions:

   1. In the [database] section, configure database access:

      ```
      [database]
      # ...
      connection = mysql+pymysql://cinder:CINDER_DBPASS@controller/cinder
      ```

      Replace CINDER_DBPASS with the password you chose for the Block Storage database.

   2. In the [DEFAULT] section, configure RabbitMQ message queue access:

      ```
      [DEFAULT]
      # ...
      transport_url = rabbit://openstack:RABBIT_PASS@controller
      ```

      Replace RABBIT_PASS with the password you chose for the openstack account in RabbitMQ.

   3. In the [DEFAULT] and [keystone_authtoken] sections, configure Identity service access:

      ```
      [DEFAULT]
      # ...
      auth_strategy = keystone

      [keystone_authtoken]
      # ...
      www_authenticate_uri = http://controller:5000
      auth_url = http://controller:5000
      memcached_servers = controller:11211
      auth_type = password
      project_domain_name = default
      user_domain_name = default
      project_name = service
      username = cinder
      password = CINDER_PASS
      ```

      Replace CINDER_PASS with the password you chose for the cinder user in the Identity service.

      Note

      Comment out or remove any other options in the [keystone_authtoken] section.

   4. In the [DEFAULT] section, configure the my_ip option to use the management interface IP address of the controller node:

```
[DEFAULT]
# ...
my_ip = 10.0.0.11
```

3. In the [oslo_concurrency] section, configure the lock path:

```
[oslo_concurrency]
# ...
lock_path = /var/lib/cinder/tmp
```

Configure Compute to use Block Storage

1. Edit the /etc/nova/nova.conf file and add the following to it:

```
[cinder]
os_region_name = RegionOne
```

Finalize installation

1. Restart the Compute API service:

```
# systemctl restart openstack-nova-api.service
```

2. Start the Block Storage services and configure them to start when the system boots:

```
# systemctl enable openstack-cinder-api.service openstack-cinder-
↪scheduler.service
# systemctl start openstack-cinder-api.service openstack-cinder-scheduler.
↪service
```

Install and configure a storage node

Prerequisites

Before you install and configure the Block Storage service on the storage node, you must prepare the storage device.

Note

Perform these steps on the storage node.

1. Install the supporting utility packages.
2. Install the LVM packages:

```
# zypper install lvm2
```

3. (Optional) If you intend to use non-raw image types such as QCOW2 and VMDK, install the QEMU package:

```
# zypper install qemu
```

Note

Some distributions include LVM by default.

4. Create the LVM physical volume `/dev/sdb`:

```
# pvcreate /dev/sdb

Physical volume "/dev/sdb" successfully created
```

5. Create the LVM volume group `cinder-volumes`:

```
# vgcreate cinder-volumes /dev/sdb

Volume group "cinder-volumes" successfully created
```

The Block Storage service creates logical volumes in this volume group.

6. Only instances can access Block Storage volumes. However, the underlying operating system manages the devices associated with the volumes. By default, the LVM volume scanning tool scans the `/dev` directory for block storage devices that contain volumes. If projects use LVM on their volumes, the scanning tool detects these volumes and attempts to cache them which can cause a variety of problems with both the underlying operating system and project volumes. You must reconfigure LVM to scan only the devices that contain the `cinder-volumes` volume group. Edit the `/etc/lvm/lvm.conf` file and complete the following actions:

   - In the `devices` section, add a filter that accepts the `/dev/sdb` device and rejects all other devices:

   ```
   devices {
   ...
   filter = [ "a/sdb/", "r/.*/"]
   ```

   Each item in the filter array begins with `a` for **accept** or `r` for **reject** and includes a regular expression for the device name. The array must end with `r/.*/` to reject any remaining devices. You can use the **vgs -vvvv** command to test filters.

Warning

If your storage nodes use LVM on the operating system disk, you must also add the associated device to the filter. For example, if the /dev/sda device contains the operating system:

```
filter = [ "a/sda/", "a/sdb/", "r/.*/"]
```

Similarly, if your compute nodes use LVM on the operating system disk, you must also modify the filter in the /etc/lvm/lvm.conf file on those nodes to include only the operating system disk. For example, if the /dev/sda device contains the operating system:

```
filter = [ "a/sda/", "r/.*/"]
```

Install and configure components

1. Install the packages:

```
# zypper install openstack-cinder-volume tgt
```

2. Edit the /etc/cinder/cinder.conf file and complete the following actions:

   - In the [database] section, configure database access:

```
[database]
# ...
connection = mysql+pymysql://cinder:CINDER_DBPASS@controller/cinder
```

   Replace CINDER_DBPASS with the password you chose for the Block Storage database.

   - In the [DEFAULT] section, configure RabbitMQ message queue access:

```
[DEFAULT]
# ...
transport_url = rabbit://openstack:RABBIT_PASS@controller
```

   Replace RABBIT_PASS with the password you chose for the openstack account in RabbitMQ.

   - In the [DEFAULT] and [keystone_authtoken] sections, configure Identity service access:

```
[DEFAULT]
# ...
auth_strategy = keystone

[keystone_authtoken]
# ...
www_authenticate_uri = http://controller:5000
auth_url = http://controller:5000
memcached_servers = controller:11211
auth_type = password
project_domain_name = default
user_domain_name = default
```

(continues on next page)

(continued from previous page)

```
project_name = service
username = cinder
password = CINDER_PASS
```

Replace `CINDER_PASS` with the password you chose for the `cinder` user in the Identity service.

Note

Comment out or remove any other options in the `[keystone_authtoken]` section.

- In the `[DEFAULT]` section, configure the `my_ip` option:

```
[DEFAULT]
# ...
my_ip = MANAGEMENT_INTERFACE_IP_ADDRESS
```

Replace `MANAGEMENT_INTERFACE_IP_ADDRESS` with the IP address of the management network interface on your storage node, typically 10.0.0.41 for the first node in the example architecture.

- In the `[lvm]` section, configure the LVM back end with the LVM driver, `cinder-volumes` volume group, iSCSI protocol, and appropriate iSCSI service:

```
[lvm]
# ...
volume_driver = cinder.volume.drivers.lvm.LVMVolumeDriver
volume_group = cinder-volumes
target_protocol = iscsi
target_helper = tgtadm
```

- In the `[DEFAULT]` section, enable the LVM back end:

```
[DEFAULT]
# ...
enabled_backends = lvm
```

Note

Back-end names are arbitrary. As an example, this guide uses the name of the driver as the name of the back end.

- In the `[DEFAULT]` section, configure the location of the Image service API:

```
[DEFAULT]
# ...
glance_api_servers = http://controller:9292
```

- In the `[oslo_concurrency]` section, configure the lock path:

```
[oslo_concurrency]
# ...
lock_path = /var/lib/cinder/tmp
```

3. Create the `/etc/tgt/conf.d/cinder.conf` file with the following data:

```
include /var/lib/cinder/volumes/*
```

Finalize installation

1. Start the Block Storage volume service including its dependencies and configure them to start when the system boots:

```
# systemctl enable openstack-cinder-volume.service tgtd.service
# systemctl start openstack-cinder-volume.service tgtd.service
```

Install and configure the backup service

Optionally, install and configure the backup service. For simplicity, this configuration uses the Block Storage node and the Object Storage (swift) driver, thus depending on the Object Storage service.

Note

You must *install and configure a storage node* prior to installing and configuring the backup service.

Install and configure components

Note

Perform these steps on the Block Storage node.

1. Install the packages:

```
# zypper install openstack-cinder-backup
```

2. Edit the `/etc/cinder/cinder.conf` file and complete the following actions:

   1. In the `[DEFAULT]` section, configure backup options:

   ```
   [DEFAULT]
   # ...
   backup_driver = cinder.backup.drivers.swift.SwiftBackupDriver
   backup_swift_url = SWIFT_URL
   ```

   Replace `SWIFT_URL` with the URL of the Object Storage service. The URL can be found by showing the object-store API endpoints:

   ```
   $ openstack catalog show object-store
   ```

Finalize installation

Start the Block Storage backup service and configure it to start when the system boots:

```
# systemctl enable openstack-cinder-backup.service
# systemctl start openstack-cinder-backup.service
```

Verify Cinder operation

Verify operation of the Block Storage service.

Note

Perform these commands on the controller node.

1. Source the `admin` credentials to gain access to admin-only CLI commands:

```
$ . admin-openrc
```

2. List service components to verify successful launch of each process:

```
$ openstack volume service list

+------------------+------------+------+---------+-------+---------------
↪------------+
| Binary           | Host       | Zone | Status  | State | Updated_at     ␣
↪           |
+------------------+------------+------+---------+-------+---------------
↪------------+
| cinder-scheduler | controller | nova | enabled | up    | 2016-09-
↪30T02:27:41.000000 |
| cinder-volume    | block@lvm  | nova | enabled | up    | 2016-09-
↪30T02:27:46.000000 |
| cinder-backup    | controller | nova | enabled | up    | 2016-09-
↪30T02:27:41.000000 |
+------------------+------------+------+---------+-------+---------------
↪------------+
```

Cinder Installation Guide for Red Hat Enterprise Linux and CentOS

This section describes how to install and configure storage nodes for the Block Storage service. For simplicity, this configuration references one storage node with an empty local block storage device. The instructions use `/dev/sdb`, but you can substitute a different value for your particular node.

The service provisions logical volumes on this device using the *LVM* driver and provides them to instances via *iSCSI* transport. You can follow these instructions with minor modifications to horizontally scale your environment with additional storage nodes.

Install and configure controller node

This section describes how to install and configure the Block Storage service, code-named cinder, on the controller node. This service requires at least one additional storage node that provides volumes to instances.

Prerequisites

Before you install and configure the Block Storage service, you must create a database, service credentials, and API endpoints.

1. To create the database, complete these steps:

   1. Use the database access client to connect to the database server as the `root` user:

      ```
      $ mysql -u root -p
      ```

   2. Create the `cinder` database:

      ```
      MariaDB [(none)]> CREATE DATABASE cinder;
      ```

   3. Grant proper access to the `cinder` database:

      ```
      MariaDB [(none)]> GRANT ALL PRIVILEGES ON cinder.* TO 'cinder'@
      ↪'localhost' \
        IDENTIFIED BY 'CINDER_DBPASS';
      MariaDB [(none)]> GRANT ALL PRIVILEGES ON cinder.* TO 'cinder'@'%' \
        IDENTIFIED BY 'CINDER_DBPASS';
      ```

      Replace `CINDER_DBPASS` with a suitable password.

   4. Exit the database access client.

2. Source the `admin` credentials to gain access to admin-only CLI commands:

   ```
   $ . admin-openrc
   ```

3. To create the service credentials, complete these steps:

   1. Create a `cinder` user:

      ```
       $ openstack user create --domain default --password-prompt cinder

      User Password:
      Repeat User Password:
      +---------------------+----------------------------------+
      | Field               | Value                            |
      +---------------------+----------------------------------+
      | domain_id           | default                          |
      | enabled             | True                             |
      | id                  | 9d7e33de3e1a498390353819bc7d245d |
      | name                | cinder                           |
      | options             | {}                               |
      | password_expires_at | None                             |
      +---------------------+----------------------------------+
      ```

2. Add the admin role to the cinder user:

```
$ openstack role add --project service --user cinder admin
```

Note

This command provides no output.

3. Create the cinderv3 service entity:

```
$ openstack service create --name cinderv3 \
  --description "OpenStack Block Storage" volumev3

+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | OpenStack Block Storage          |
| enabled     | True                             |
| id          | ab3bbbef780845a1a283490d281e7fda |
| name        | cinderv3                         |
| type        | volumev3                         |
+-------------+----------------------------------+
```

Note

Beginning with the Xena release, the Block Storage services require only one service entity. For prior releases, please consult the documentation for that specific release.

4. Create the Block Storage service API endpoints:

```
$ openstack endpoint create --region RegionOne \
  volumev3 public http://controller:8776/v3/%\(project_id\)s

+--------------+------------------------------------------+
| Field        | Value                                    |
+--------------+------------------------------------------+
| enabled      | True                                     |
| id           | 03fa2c90153546c295bf30ca86b1344b         |
| interface    | public                                   |
| region       | RegionOne                                |
| region_id    | RegionOne                                |
| service_id   | ab3bbbef780845a1a283490d281e7fda         |
| service_name | cinderv3                                 |
| service_type | volumev3                                 |
| url          | http://controller:8776/v3/%(project_id)s |
+--------------+------------------------------------------+

$ openstack endpoint create --region RegionOne \
  volumev3 internal http://controller:8776/v3/%\(project_id\)s
```

(continues on next page)

(continued from previous page)

```
+--------------+-------------------------------------------+
| Field        | Value                                     |
+--------------+-------------------------------------------+
| enabled      | True                                      |
| id           | 94f684395d1b41068c70e4ecb11364b2          |
| interface    | internal                                  |
| region       | RegionOne                                 |
| region_id    | RegionOne                                 |
| service_id   | ab3bbbef780845a1a283490d281e7fda          |
| service_name | cinderv3                                  |
| service_type | volumev3                                  |
| url          | http://controller:8776/v3/%(project_id)s |
+--------------+-------------------------------------------+

$ openstack endpoint create --region RegionOne \
  volumev3 admin http://controller:8776/v3/%\(project_id\)s

+--------------+-------------------------------------------+
| Field        | Value                                     |
+--------------+-------------------------------------------+
| enabled      | True                                      |
| id           | 4511c28a0f9840c78bacb25f10f62c98          |
| interface    | admin                                     |
| region       | RegionOne                                 |
| region_id    | RegionOne                                 |
| service_id   | ab3bbbef780845a1a283490d281e7fda          |
| service_name | cinderv3                                  |
| service_type | volumev3                                  |
| url          | http://controller:8776/v3/%(project_id)s |
+--------------+-------------------------------------------+
```

Install and configure components

1. Install the packages:

```
# yum install openstack-cinder
```

2. Edit the `/etc/cinder/cinder.conf` file and complete the following actions:

   1. In the `[database]` section, configure database access:

   ```
   [database]
   # ...
   connection = mysql+pymysql://cinder:CINDER_DBPASS@controller/cinder
   ```

   Replace `CINDER_DBPASS` with the password you chose for the Block Storage database.

   2. In the `[DEFAULT]` section, configure `RabbitMQ` message queue access:

```
[DEFAULT]
# ...
transport_url = rabbit://openstack:RABBIT_PASS@controller
```

Replace `RABBIT_PASS` with the password you chose for the `openstack` account in `RabbitMQ`.

3. In the `[DEFAULT]` and `[keystone_authtoken]` sections, configure Identity service access:

```
[DEFAULT]
# ...
auth_strategy = keystone

[keystone_authtoken]
# ...
www_authenticate_uri = http://controller:5000
auth_url = http://controller:5000
memcached_servers = controller:11211
auth_type = password
project_domain_name = default
user_domain_name = default
project_name = service
username = cinder
password = CINDER_PASS
```

Replace `CINDER_PASS` with the password you chose for the `cinder` user in the Identity service.

Note

Comment out or remove any other options in the `[keystone_authtoken]` section.

4. In the `[DEFAULT]` section, configure the `my_ip` option to use the management interface IP address of the controller node:

```
[DEFAULT]
# ...
my_ip = 10.0.0.11
```

3. In the `[oslo_concurrency]` section, configure the lock path:

```
[oslo_concurrency]
# ...
lock_path = /var/lib/cinder/tmp
```

4. Populate the Block Storage database:

```
# su -s /bin/sh -c "cinder-manage db sync" cinder
```

Note

Ignore any deprecation messages in this output.

Configure Compute to use Block Storage

1. Edit the `/etc/nova/nova.conf` file and add the following to it:

```
[cinder]
os_region_name = RegionOne
```

Finalize installation

1. Restart the Compute API service:

```
# systemctl restart openstack-nova-api.service
```

2. Start the Block Storage services and configure them to start when the system boots:

```
# systemctl enable openstack-cinder-api.service openstack-cinder-
↪scheduler.service
# systemctl start openstack-cinder-api.service openstack-cinder-scheduler.
↪service
```

Install and configure a storage node

Prerequisites

Before you install and configure the Block Storage service on the storage node, you must prepare the storage device.

Note

Perform these steps on the storage node.

1. Install the supporting utility packages:

   - Install the LVM packages:

   ```
   # yum install lvm2 device-mapper-persistent-data
   ```

   - This is not required for CentOS 8 or later, which comes with a version of LVM that does not use the lvmetad service:

   ```
   # systemctl enable lvm2-lvmetad.service
   # systemctl start lvm2-lvmetad.service
   ```

   Note

   Some distributions include LVM by default.

2. Create the LVM physical volume `/dev/sdb`:

```
# pvcreate /dev/sdb

Physical volume "/dev/sdb" successfully created
```

3. Create the LVM volume group `cinder-volumes`:

```
# vgcreate cinder-volumes /dev/sdb

Volume group "cinder-volumes" successfully created
```

The Block Storage service creates logical volumes in this volume group.

4. Only instances can access Block Storage volumes. However, the underlying operating system manages the devices associated with the volumes. By default, the LVM volume scanning tool scans the `/dev` directory for block storage devices that contain volumes. If projects use LVM on their volumes, the scanning tool detects these volumes and attempts to cache them which can cause a variety of problems with both the underlying operating system and project volumes. You must reconfigure LVM to scan only the devices that contain the `cinder-volumes` volume group. Edit the `/etc/lvm/lvm.conf` file and complete the following actions:

   - In the `devices` section, add a filter that accepts the `/dev/sdb` device and rejects all other devices:

     ```
     devices {
     ...
     filter = [ "a/sdb/", "r/.*/"]
     ```

     Each item in the filter array begins with `a` for **accept** or `r` for **reject** and includes a regular expression for the device name. The array must end with `r/.*/` to reject any remaining devices. You can use the **vgs -vvvv** command to test filters.

     Warning

     If your storage nodes use LVM on the operating system disk, you must also add the associated device to the filter. For example, if the `/dev/sda` device contains the operating system:

     ```
     filter = [ "a/sda/", "a/sdb/", "r/.*/"]
     ```

     Similarly, if your compute nodes use LVM on the operating system disk, you must also modify the filter in the `/etc/lvm/lvm.conf` file on those nodes to include only the operating system disk. For example, if the `/dev/sda` device contains the operating system:

     ```
     filter = [ "a/sda/", "r/.*/"]
     ```

Install and configure components

1. Install the packages:

```
# yum install openstack-cinder targetcli
```

2. Edit the `/etc/cinder/cinder.conf` file and complete the following actions:

- In the [database] section, configure database access:

```
[database]
# ...
connection = mysql+pymysql://cinder:CINDER_DBPASS@controller/cinder
```

Replace CINDER_DBPASS with the password you chose for the Block Storage database.

- In the [DEFAULT] section, configure RabbitMQ message queue access:

```
[DEFAULT]
# ...
transport_url = rabbit://openstack:RABBIT_PASS@controller
```

Replace RABBIT_PASS with the password you chose for the openstack account in RabbitMQ.

- In the [DEFAULT] and [keystone_authtoken] sections, configure Identity service access:

```
[DEFAULT]
# ...
auth_strategy = keystone

[keystone_authtoken]
# ...
www_authenticate_uri = http://controller:5000
auth_url = http://controller:5000
memcached_servers = controller:11211
auth_type = password
project_domain_name = default
user_domain_name = default
project_name = service
username = cinder
password = CINDER_PASS
```

Replace CINDER_PASS with the password you chose for the cinder user in the Identity service.

Note

Comment out or remove any other options in the [keystone_authtoken] section.

- In the [DEFAULT] section, configure the my_ip option:

```
[DEFAULT]
# ...
my_ip = MANAGEMENT_INTERFACE_IP_ADDRESS
```

Replace MANAGEMENT_INTERFACE_IP_ADDRESS with the IP address of the management network interface on your storage node, typically 10.0.0.41 for the first node in the example architecture.

- In the [lvm] section, configure the LVM back end with the LVM driver, cinder-volumes volume group, iSCSI protocol, and appropriate iSCSI service. If the [lvm] section does not exist, create it:

```
[lvm]
volume_driver = cinder.volume.drivers.lvm.LVMVolumeDriver
volume_group = cinder-volumes
target_protocol = iscsi
target_helper = lioadm
```

- In the [DEFAULT] section, enable the LVM back end:

```
[DEFAULT]
# ...
enabled_backends = lvm
```

> **Note**
>
> Back-end names are arbitrary. As an example, this guide uses the name of the driver as the name of the back end.

- In the [DEFAULT] section, configure the location of the Image service API:

```
[DEFAULT]
# ...
glance_api_servers = http://controller:9292
```

- In the [oslo_concurrency] section, configure the lock path:

```
[oslo_concurrency]
# ...
lock_path = /var/lib/cinder/tmp
```

Finalize installation

- Start the Block Storage volume service including its dependencies and configure them to start when the system boots:

```
# systemctl enable openstack-cinder-volume.service target.service
# systemctl start openstack-cinder-volume.service target.service
```

Install and configure the backup service

Optionally, install and configure the backup service. For simplicity, this configuration uses the Block Storage node and the Object Storage (swift) driver, thus depending on the Object Storage service.

> **Note**
>
> You must *install and configure a storage node* prior to installing and configuring the backup service.

Install and configure components

Note

Perform these steps on the Block Storage node.

1. Install the packages:

```
# yum install openstack-cinder
```

2. Edit the /etc/cinder/cinder.conf file and complete the following actions:

   1. In the [DEFAULT] section, configure backup options:

```
[DEFAULT]
# ...
backup_driver = cinder.backup.drivers.swift.SwiftBackupDriver
backup_swift_url = SWIFT_URL
```

   Replace SWIFT_URL with the URL of the Object Storage service. The URL can be found by showing the object-store API endpoints:

```
$ openstack catalog show object-store
```

Finalize installation

Start the Block Storage backup service and configure it to start when the system boots:

```
# systemctl enable openstack-cinder-backup.service
# systemctl start openstack-cinder-backup.service
```

Cinder Installation Guide for Ubuntu

This section describes how to install and configure storage nodes for the Block Storage service. For simplicity, this configuration references one storage node with an empty local block storage device. The instructions use /dev/sdb, but you can substitute a different value for your particular node.

The service provisions logical volumes on this device using the *LVM* driver and provides them to instances via *iSCSI* transport. You can follow these instructions with minor modifications to horizontally scale your environment with additional storage nodes.

Install and configure controller node

This section describes how to install and configure the Block Storage service, code-named cinder, on the controller node. This service requires at least one additional storage node that provides volumes to instances.

Prerequisites

Before you install and configure the Block Storage service, you must create a database, service credentials, and API endpoints.

1. To create the database, complete these steps:

   1. Use the database access client to connect to the database server as the `root` user:

      ```
      # mysql
      ```

   2. Create the `cinder` database:

      ```
      MariaDB [(none)]> CREATE DATABASE cinder;
      ```

   3. Grant proper access to the `cinder` database:

      ```
      MariaDB [(none)]> GRANT ALL PRIVILEGES ON cinder.* TO 'cinder'@
      ↪'localhost' \
        IDENTIFIED BY 'CINDER_DBPASS';
      MariaDB [(none)]> GRANT ALL PRIVILEGES ON cinder.* TO 'cinder'@'%' \
        IDENTIFIED BY 'CINDER_DBPASS';
      ```

      Replace `CINDER_DBPASS` with a suitable password.

   4. Exit the database access client.

2. Source the `admin` credentials to gain access to admin-only CLI commands:

   ```
   $ . admin-openrc
   ```

3. To create the service credentials, complete these steps:

   1. Create a `cinder` user:

      ```
      $ openstack user create --domain default --password-prompt cinder

      User Password:
      Repeat User Password:
      +---------------------+----------------------------------+
      | Field               | Value                            |
      +---------------------+----------------------------------+
      | domain_id           | default                          |
      | enabled             | True                             |
      | id                  | 9d7e33de3e1a498390353819bc7d245d |
      | name                | cinder                           |
      | options             | {}                               |
      | password_expires_at | None                             |
      +---------------------+----------------------------------+
      ```

   2. Add the `admin` role to the `cinder` user:

      ```
      $ openstack role add --project service --user cinder admin
      ```

Note

This command provides no output.

3. Create the `cinderv3` service entity:

```
$ openstack service create --name cinderv3 \
  --description "OpenStack Block Storage" volumev3

+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | OpenStack Block Storage          |
| enabled     | True                             |
| id          | ab3bbbef780845a1a283490d281e7fda |
| name        | cinderv3                         |
| type        | volumev3                         |
+-------------+----------------------------------+
```

Note

Beginning with the Xena release, the Block Storage services require only one service entity. For prior releases, please consult the documentation for that specific release.

4. Create the Block Storage service API endpoints:

```
$ openstack endpoint create --region RegionOne \
  volumev3 public http://controller:8776/v3/%\(project_id\)s

+--------------+------------------------------------------+
| Field        | Value                                    |
+--------------+------------------------------------------+
| enabled      | True                                     |
| id           | 03fa2c90153546c295bf30ca86b1344b         |
| interface    | public                                   |
| region       | RegionOne                                |
| region_id    | RegionOne                                |
| service_id   | ab3bbbef780845a1a283490d281e7fda         |
| service_name | cinderv3                                 |
| service_type | volumev3                                 |
| url          | http://controller:8776/v3/%(project_id)s |
+--------------+------------------------------------------+

$ openstack endpoint create --region RegionOne \
  volumev3 internal http://controller:8776/v3/%\(project_id\)s

+--------------+------------------------------------------+
| Field        | Value                                    |
+--------------+------------------------------------------+
```

(continues on next page)

(continued from previous page)

```
| enabled      | True                                      |
| id           | 94f684395d1b41068c70e4ecb11364b2          |
| interface    | internal                                  |
| region       | RegionOne                                 |
| region_id    | RegionOne                                 |
| service_id   | ab3bbbef780845a1a283490d281e7fda          |
| service_name | cinderv3                                  |
| service_type | volumev3                                  |
| url          | http://controller:8776/v3/%(project_id)s |
+--------------+------------------------------------------+

$ openstack endpoint create --region RegionOne \
  volumev3 admin http://controller:8776/v3/%\(project_id\)s

+--------------+------------------------------------------+
| Field        | Value                                     |
+--------------+------------------------------------------+
| enabled      | True                                      |
| id           | 4511c28a0f9840c78bacb25f10f62c98          |
| interface    | admin                                     |
| region       | RegionOne                                 |
| region_id    | RegionOne                                 |
| service_id   | ab3bbbef780845a1a283490d281e7fda          |
| service_name | cinderv3                                  |
| service_type | volumev3                                  |
| url          | http://controller:8776/v3/%(project_id)s |
+--------------+------------------------------------------+
```

Install and configure components

1. Install the packages:

```
# apt install cinder-api cinder-scheduler
```

2. Edit the `/etc/cinder/cinder.conf` file and complete the following actions:

   1. In the `[database]` section, configure database access:

   ```
   [database]
   # ...
   connection = mysql+pymysql://cinder:CINDER_DBPASS@controller/cinder
   ```

   Replace `CINDER_DBPASS` with the password you chose for the Block Storage database.

   2. In the `[DEFAULT]` section, configure `RabbitMQ` message queue access:

   ```
   [DEFAULT]
   # ...
   transport_url = rabbit://openstack:RABBIT_PASS@controller
   ```

   Replace `RABBIT_PASS` with the password you chose for the `openstack` account in `RabbitMQ`.

3. In the `[DEFAULT]` and `[keystone_authtoken]` sections, configure Identity service access:

```
[DEFAULT]
# ...
auth_strategy = keystone

[keystone_authtoken]
# ...
www_authenticate_uri = http://controller:5000
auth_url = http://controller:5000
memcached_servers = controller:11211
auth_type = password
project_domain_name = default
user_domain_name = default
project_name = service
username = cinder
password = CINDER_PASS
```

   Replace `CINDER_PASS` with the password you chose for the `cinder` user in the Identity service.

   Note

   Comment out or remove any other options in the `[keystone_authtoken]` section.

4. In the `[DEFAULT]` section, configure the `my_ip` option to use the management interface IP address of the controller node:

```
[DEFAULT]
# ...
my_ip = 10.0.0.11
```

3. In the `[oslo_concurrency]` section, configure the lock path:

```
[oslo_concurrency]
# ...
lock_path = /var/lib/cinder/tmp
```

4. Populate the Block Storage database:

```
# su -s /bin/sh -c "cinder-manage db sync" cinder
```

Note

Ignore any deprecation messages in this output.

Configure Compute to use Block Storage

1. Edit the /etc/nova/nova.conf file and add the following to it:

```
[cinder]
os_region_name = RegionOne
```

Finalize installation

1. Restart the Compute API service:

```
# service nova-api restart
```

2. Restart the Block Storage services:

```
# service cinder-scheduler restart
# service apache2 restart
```

Install and configure a storage node

Prerequisites

Before you install and configure the Block Storage service on the storage node, you must prepare the storage device.

Note

Perform these steps on the storage node.

1. Install the supporting utility packages:

```
# apt install lvm2 thin-provisioning-tools
```

Note

Some distributions include LVM by default.

2. Create the LVM physical volume /dev/sdb:

```
# pvcreate /dev/sdb

Physical volume "/dev/sdb" successfully created
```

3. Create the LVM volume group cinder-volumes:

```
# vgcreate cinder-volumes /dev/sdb

Volume group "cinder-volumes" successfully created
```

The Block Storage service creates logical volumes in this volume group.

4. Only instances can access Block Storage volumes. However, the underlying operating system manages the devices associated with the volumes. By default, the LVM volume scanning tool scans the `/dev` directory for block storage devices that contain volumes. If projects use LVM on their volumes, the scanning tool detects these volumes and attempts to cache them which can cause a variety of problems with both the underlying operating system and project volumes. You must reconfigure LVM to scan only the devices that contain the `cinder-volumes` volume group. Edit the `/etc/lvm/lvm.conf` file and complete the following actions:

   - In the `devices` section, add a filter that accepts the `/dev/sdb` device and rejects all other devices:

     ```
     devices {
     ...
     filter = [ "a/sdb/", "r/.*/"]
     ```

     Each item in the filter array begins with `a` for **accept** or `r` for **reject** and includes a regular expression for the device name. The array must end with `r/.*/` to reject any remaining devices. You can use the **vgs -vvvv** command to test filters.

     Warning

     If your storage nodes use LVM on the operating system disk, you must also add the associated device to the filter. For example, if the `/dev/sda` device contains the operating system:

     ```
     filter = [ "a/sda/", "a/sdb/", "r/.*/"]
     ```

     Similarly, if your compute nodes use LVM on the operating system disk, you must also modify the filter in the `/etc/lvm/lvm.conf` file on those nodes to include only the operating system disk. For example, if the `/dev/sda` device contains the operating system:

     ```
     filter = [ "a/sda/", "r/.*/"]
     ```

Install and configure components

1. Install the packages:

   ```
   # apt install cinder-volume tgt
   ```

2. Edit the `/etc/cinder/cinder.conf` file and complete the following actions:

   - In the `[database]` section, configure database access:

     ```
     [database]
     # ...
     connection = mysql+pymysql://cinder:CINDER_DBPASS@controller/cinder
     ```

     Replace `CINDER_DBPASS` with the password you chose for the Block Storage database.

   - In the `[DEFAULT]` section, configure `RabbitMQ` message queue access:

     ```
     [DEFAULT]
     # ...
     transport_url = rabbit://openstack:RABBIT_PASS@controller
     ```

Replace RABBIT_PASS with the password you chose for the openstack account in RabbitMQ.

- In the [DEFAULT] and [keystone_authtoken] sections, configure Identity service access:

```
[DEFAULT]
# ...
auth_strategy = keystone

[keystone_authtoken]
# ...
www_authenticate_uri = http://controller:5000
auth_url = http://controller:5000
memcached_servers = controller:11211
auth_type = password
project_domain_name = default
user_domain_name = default
project_name = service
username = cinder
password = CINDER_PASS
```

Replace CINDER_PASS with the password you chose for the cinder user in the Identity service.

Note

Comment out or remove any other options in the [keystone_authtoken] section.

- In the [DEFAULT] section, configure the my_ip option:

```
[DEFAULT]
# ...
my_ip = MANAGEMENT_INTERFACE_IP_ADDRESS
```

Replace MANAGEMENT_INTERFACE_IP_ADDRESS with the IP address of the management network interface on your storage node, typically 10.0.0.41 for the first node in the example architecture.

- In the [lvm] section, configure the LVM back end with the LVM driver, cinder-volumes volume group, iSCSI protocol, and appropriate iSCSI service:

```
[lvm]
# ...
volume_driver = cinder.volume.drivers.lvm.LVMVolumeDriver
volume_group = cinder-volumes
target_protocol = iscsi
target_helper = tgtadm
```

- In the [DEFAULT] section, enable the LVM back end:

```
[DEFAULT]
# ...
```

(continues on next page)

(continued from previous page)

```
enabled_backends = lvm
```

Note

Back-end names are arbitrary. As an example, this guide uses the name of the driver as the name of the back end.

- In the `[DEFAULT]` section, configure the location of the Image service API:

```
[DEFAULT]
# ...
glance_api_servers = http://controller:9292
```

- In the `[oslo_concurrency]` section, configure the lock path:

```
[oslo_concurrency]
# ...
lock_path = /var/lib/cinder/tmp
```

3. Create the `/etc/tgt/conf.d/cinder.conf` file with the following data:

Note

Perform this step only when using tgt target.

```
include /var/lib/cinder/volumes/*
```

Finalize installation

1. Restart the Block Storage volume service including its dependencies:

```
# service tgt restart
# service cinder-volume restart
```

Install and configure the backup service

Optionally, install and configure the backup service. For simplicity, this configuration uses the Block Storage node and the Object Storage (swift) driver, thus depending on the Object Storage service.

Note

You must *install and configure a storage node* prior to installing and configuring the backup service.

Install and configure components

> **Note**
>
> Perform these steps on the Block Storage node.

1. Install the packages:

```
# apt install cinder-backup
```

2. Edit the `/etc/cinder/cinder.conf` file and complete the following actions:

   - In the `[DEFAULT]` section, configure backup options:

```
[DEFAULT]
# ...
backup_driver = cinder.backup.drivers.swift.SwiftBackupDriver
backup_swift_url = SWIFT_URL
```

   Replace `SWIFT_URL` with the URL of the Object Storage service. The URL can be found by showing the object-store API endpoints:

```
$ openstack catalog show object-store
```

Finalize installation

Restart the Block Storage backup service:

```
# service cinder-backup restart
```

Cinder Installation Guide for Windows

This section describes how to install and configure storage nodes for the Block Storage service.

For the moment, Cinder Volume is the only Cinder service supported on Windows.

Install and configure a storage node

Prerequisites

The following Windows versions are officially supported by Cinder:

- `Windows Server 2012`
- `Windows Server 2012 R2`
- `Windows Server 2016`

The OpenStack Cinder Volume MSI installer is the recommended deployment tool for Cinder on Windows. You can find it at https://cloudbase.it/openstack-windows-storage/#download.

It installs an independent Python environment, in order to avoid conflicts with existing applications. It can dynamically generate a `cinder.conf` file based on the parameters you provide.

The OpenStack Cinder Volume MSI installer can be deployed in a fully automated way using Puppet, Chef, SaltStack, Ansible, Juju, DSC, Windows Group Policies or any other automated configuration framework.

Configure NTP

Network time services must be configured to ensure proper operation of the OpenStack nodes. To set network time on your Windows host you must run the following commands:

```
net stop w32time
w32tm /config /manualpeerlist:pool.ntp.org,0x8 /syncfromflags:MANUAL
net start w32time
```

Keep in mind that the node will have to be time synchronized with the other nodes of your OpenStack environment, so it is important to use the same NTP server.

Note

In case of an Active Directory environment, you may do this only for the AD Domain Controller.

Install and configure components

The MSI may be run in the following modes:

Graphical mode

The installer will walk you through the commonly used cinder options, automatically generating a config file based on your input.

You may run the following in order to run the installer in graphical mode, also specifying a log file. Please use the installer full path.

```
msiexec /i CinderVolumeSetup.msi /l*v msi_log.txt
```

Unattended mode

The installer will deploy Cinder, taking care of required Windows services and features. A minimal sample config file will be generated and need to be updated accordingly.

Run the following in order to install Cinder in unattended mode, enabling the iSCSI and SMB volume drivers.

```
msiexec /i CinderVolumeSetup.msi /qn /l*v msi_log.txt `
           ADDLOCAL="iscsiDriver,smbDriver"
```

By default, Cinder will be installed at `%ProgramFiles%\Cloudbase Solutions\OpenStack`. You may choose a different install directory by using the `INSTALLDIR` argument, as following:

```
msiexec /i CinderVolumeSetup.msi /qn /l*v msi_log.txt `
           ADDLOCAL="iscsiDriver,smbDriver" `
           INSTALLDIR="C:\cinder"
```

The installer will generate a Windows service, called `cinder-volume`.

Note

Previous MSI releases may use a separate service per volume backend (e.g. cinder-volume-smb). You may double check the cinder services along with their executable paths by running the following:

```
get-service cinder-volume*
sc.exe qc cinder-volume-smb
```

Note that `sc` is also an alias for `Set-Content`. To use the service control utility, you have to explicitly call `sc.exe`.

Configuring Cinder

If youve run the installer in graphical mode, you may skip this part as the MSI already took care of generating the configuration files.

The Cinder Volume Windows service configured by the MSI expects the cinder config file to reside at:

```
%INSTALLDIR%\etc\cinder.conf
```

You may use the following config sample, updating fields appropriately.

```
[DEFAULT]
my_ip = MANAGEMENT_INTERFACE_IP_ADDRESS
auth_strategy = keystone
transport_url = rabbit://RABBIT_USER:RABBIT_PASS@controller:5672
glance_api_servers = http://controller/image
sql_connection = mysql+pymysql://cinder:CINDER_DBPASS@controller/cinder
image_conversion_dir = C:\OpenStack\ImageConversionDir\
lock_path = C:\OpenStack\Lock\
log_dir = C:\OpenStack\Log\
log_file = cinder-volume.log

[coordination]
backend_url = file:///C:/OpenStack/Lock/

[key_manager]
api_class = cinder.keymgr.conf_key_mgr.ConfKeyManager
```

Note

The above sample doesnt configure any Cinder Volume driver. To do so, follow the configuration guide for the driver of choice, appending driver specific config options.

Currently supported drivers on Windows:

- *Windows SMB volume driver*
- *Windows iSCSI volume driver*

Finalize installation

1. Restart the Cinder Volume service:

```
Restart-Service cinder-volume
```

2. Ensure that the Cinder Volume service is running:

```
Get-Service cinder-volume
```

3.2 Administrating Cinder

Contents:

3.2.1 Cinder Administration

The OpenStack Block Storage service works through the interaction of a series of daemon processes named `cinder-*` that reside persistently on the host machine or machines. You can run all the binaries from a single node, or spread across multiple nodes. You can also run them on the same node as other OpenStack services.

To administer the OpenStack Block Storage service, it is helpful to understand a number of concepts. You must make certain choices when you configure the Block Storage service in OpenStack. The bulk of the options come down to two choices - single node or multi-node install. You can read a longer discussion about Storage Decisions in the OpenStack Operations Guide.

OpenStack Block Storage enables you to add extra block-level storage to your OpenStack Compute instances. This service is similar to the Amazon EC2 Elastic Block Storage (EBS) offering.

Security

Network traffic

Depending on your deployments security requirements, you might be required to encrypt network traffic. This can be accomplished with TLS.

There are multiple deployment options, with the most common and recommended ones being:

- Only encrypt traffic between clients and public endpoints. This approach results in fewer certificates to manage, and we refer to it as public TLS. Public endpoints, in this sense, are endpoints only exposed to end-users. Traffic between internal endpoints is not encrypted.
- Leverages TLS for all endpoints in the entire deployment, including internal endpoints of the OpenStack services and with auxiliary services such as the database and the message broker.

You can look at TripleOs documentation on TLS for examples on how to do this.

Cinder drivers should support secure TLS/SSL communication between the cinder volume service and the backend, as configured by the `driver_ssl_cert_verify` and `driver_ssl_cert_path` options in `cinder.conf`.

If unsure whether your driver supports TLS/SSL, please check the drivers specific page in the *Volume drivers* page or contact the vendor.

Data at rest

Volumes data can be secured at rest using Cinders volume encryption feature.

For encryption keys Cinder uses a Key management service, with Barbican being the recommended service.

More information on encryption can be found on the *Volume encryption supported by the key manager* section.

Data leakage

Some users and admins worry about data leakage between OpenStack projects or users caused by a new volume containing partial or full data from a previously deleted volume.

These concerns are sometimes instigated by the `volume_clear` and `volume_clear_size` configuration options, but these options are only relevant to the LVM driver, and only when using thick volumes (which are not the default, thin volumes are).

Writing data on a Cinder volume as a generic mechanism to prevent data leakage is not implemented for other drivers because it does not ensure that the data will be actually erased on the physical disks, as the storage solution could be doing copy-on-write or other optimizations.

Thin provisioned volumes return zeros for unallocated blocks, so we dont have to worry about data leakage. As for thick volumes, each of the individual Cinder drivers must ensure that data from a deleted volume can never leak to a newly created volume.

This prevents other OpenStack projects and users from being able to get data from deleted volumes, but since the data may still be present on the physical disks, somebody with physical access to the disks may still be able to retrieve that data.

For those concerned with this, we recommend using encrypted volumes or read your storage solutions documentation or contact your vendor to see if they have some kind of clear policy option available on their storage solution.

Accelerate image compression

A general framework to accommodate hardware compression accelerators for compression of volumes uploaded to the Image service (Glance) as images and decompression of compressed images used to create volumes is introduced in Train release.

The only accelerator supported in this release is Intel QuickAssist Technology (QAT), which produces a compressed file in gzip format. Additionally, the framework provides software-based compression using GUNzip tool if a suitable hardware accelerator is not available. Because this software fallback could cause performance problems if the Cinder services are not deployed on sufficiently powerful nodes, the default setting is *not* to enable compression on image upload or download.

The compressed image of a volume will be stored in the Image service (Glance) with the `container_format` image property of `compressed`. See the Image service documentation for more information about this image container format.

Configure image compression

To enable the image compression feature, set the following configuration option in the `cinder.conf` file:

```
allow_compression_on_image_upload = True
```

By default it will be set to False, which means image compression is disabled.

```
compression_format = gzip
```

This is to specify image compression format. The only supported format is `gzip` in Train release.

System requirement

In order to use this feature, there should be a hardware accelerator existing in system, otherwise no benefit will get from this feature. Regarding the two accelerators that supported, system should be configured as below:

- `Intel QuickAssist Technology (QAT)` - This is the hardware accelerator from Intel. The driver of QAT should be installed, refer to https://01.org/intel-quickassist-technology. Also the compression library QATzip should be installed, refer to https://github.com/intel/QATzip.
- `GUNzip` - The related package of `GUNzip` should be installed and the command `gzip` should be available. This is used as fallback when hardware accelerator is not available.

Increase Block Storage API service throughput

By default, the Block Storage API service runs in one process. This limits the number of API requests that the Block Storage service can process at any given time. In a production environment, you should increase the Block Storage API throughput by allowing the Block Storage API service to run in as many processes as the machine capacity allows.

> **Note**
>
> The Block Storage API service is named `openstack-cinder-api` on the following distributions: CentOS, Fedora, openSUSE, Red Hat Enterprise Linux, and SUSE Linux Enterprise. In Ubuntu and Debian distributions, the Block Storage API service is named `cinder-api`.

To do so, use the Block Storage API service option `osapi_volume_workers`. This option allows you to specify the number of API service workers (or OS processes) to launch for the Block Storage API service.

To configure this option, open the `/etc/cinder/cinder.conf` configuration file and set the `osapi_volume_workers` configuration key to the number of CPU cores/threads on a machine.

On distributions that include `openstack-config`, you can configure this by running the following command instead:

```
# openstack-config --set /etc/cinder/cinder.conf \
  DEFAULT osapi_volume_workers CORES
```

Replace `CORES` with the number of CPU cores/threads on a machine.

Manage volumes

The default OpenStack Block Storage service implementation is an iSCSI solution that uses *Logical Volume Manager (LVM)* for Linux.

> **Note**
>
> The OpenStack Block Storage service also provides drivers that enable you to use several vendors back-end storage devices in addition to the base LVM implementation. These storage devices can also be used instead of the base LVM installation.

This high-level procedure shows you how to create and attach a volume to a server instance.

To create and attach a volume to an instance

1. Configure the OpenStack Compute and the OpenStack Block Storage services through the `/etc/cinder/cinder.conf` file.

2. Use the **`openstack volume create`** command to create a volume. This command creates an LV into the volume group (VG) `cinder-volumes`.

3. Use the **`openstack server add volume`** command to attach the volume to an instance. This command creates a unique *IQN* that is exposed to the compute node.

   - The compute node, which runs the instance, now has an active iSCSI session and new local storage (usually a `/dev/sdX` disk).

   - Libvirt uses that local storage as storage for the instance. The instance gets a new disk (usually a `/dev/vdX` disk).

For this particular walkthrough, one cloud controller runs `nova-api`, `nova-scheduler`, `nova-conductor` and `cinder-*` services. Two additional compute nodes run `nova-compute`. The walkthrough uses a custom partitioning scheme that carves out 60 GB of space and labels it as LVM. The network uses the `FlatManager` and `NetworkManager` settings for OpenStack Compute.

The network mode does not interfere with OpenStack Block Storage operations, but you must set up networking for Block Storage to work. For details, see networking.

To set up Compute to use volumes, ensure that Block Storage is installed along with `lvm2`. This guide describes how to troubleshoot your installation and back up your Compute volumes.

Boot from volume

In some cases, you can store and run instances from inside volumes. For information, see Launch an instance from a volume.

Configure an NFS storage back end

This section explains how to configure OpenStack Block Storage to use NFS storage. You must be able to access the NFS shares from the server that hosts the `cinder` volume service.

> **Note**
>
> The `cinder` volume service is named `openstack-cinder-volume` on the following distributions:
>
> - CentOS

- Fedora
- openSUSE
- Red Hat Enterprise Linux
- SUSE Linux Enterprise

In Ubuntu and Debian distributions, the `cinder` volume service is named `cinder-volume`.

Configure Block Storage to use an NFS storage back end

1. Log in as `root` to the system hosting the `cinder` volume service.
2. Create a text file named `nfs_shares` in the `/etc/cinder/` directory.
3. Add an entry to `/etc/cinder/nfs_shares` for each NFS share that the `cinder` volume service should use for back end storage. Each entry should be a separate line, and should use the following format:

   ```
   HOST:SHARE
   ```

   Where:

   - HOST is the IP address or host name of the NFS server.
   - SHARE is the absolute path to an existing and accessible NFS share.

4. Set `/etc/cinder/nfs_shares` to be owned by the `root` user and the `cinder` group:

   ```
   # chown root:cinder /etc/cinder/nfs_shares
   ```

5. Set `/etc/cinder/nfs_shares` to be readable by members of the cinder group:

   ```
   # chmod 0640 /etc/cinder/nfs_shares
   ```

6. Configure the `cinder` volume service to use the `/etc/cinder/nfs_shares` file created earlier. To do so, open the `/etc/cinder/cinder.conf` configuration file and set the `nfs_shares_config` configuration key to `/etc/cinder/nfs_shares`.

   On distributions that include `openstack-config`, you can configure this by running the following command instead:

   ```
   # openstack-config --set /etc/cinder/cinder.conf \
     DEFAULT nfs_shares_config /etc/cinder/nfs_shares
   ```

   The following distributions include openstack-config:

   - CentOS
   - Fedora
   - openSUSE

- Red Hat Enterprise Linux
- SUSE Linux Enterprise

7. Optionally, provide any additional NFS mount options required in your environment in the `nfs_mount_options` configuration key of `/etc/cinder/cinder.conf`. If your NFS shares do not require any additional mount options (or if you are unsure), skip this step.

   On distributions that include `openstack-config`, you can configure this by running the following command instead:

   ```
   # openstack-config --set /etc/cinder/cinder.conf \
     DEFAULT nfs_mount_options OPTIONS
   ```

   Replace OPTIONS with the mount options to be used when accessing NFS shares. See the manual page for NFS for more information on available mount options (**`man nfs`**).

8. Configure the `cinder` volume service to use the correct volume driver, namely `cinder.volume.drivers.nfs.NfsDriver`. To do so, open the `/etc/cinder/cinder.conf` configuration file and set the volume_driver configuration key to `cinder.volume.drivers.nfs.NfsDriver`.

   On distributions that include `openstack-config`, you can configure this by running the following command instead:

   ```
   # openstack-config --set /etc/cinder/cinder.conf \
     DEFAULT volume_driver cinder.volume.drivers.nfs.NfsDriver
   ```

9. You can now restart the service to apply the configuration.

Note

The `nfs_sparsed_volumes` configuration key determines whether volumes are created as sparse files and grown as needed or fully allocated up front. The default and recommended value is `true`, which ensures volumes are initially created as sparse files.

Setting `nfs_sparsed_volumes` to `false` will result in volumes being fully allocated at the time of creation. This leads to increased delays in volume creation.

However, should you choose to set `nfs_sparsed_volumes` to `false`, you can do so directly in `/etc/cinder/cinder.conf`.

On distributions that include `openstack-config`, you can configure this by running the following command instead:

```
# openstack-config --set /etc/cinder/cinder.conf \
  DEFAULT nfs_sparsed_volumes false
```

Warning

If a client host has SELinux enabled, the `virt_use_nfs` boolean should also be enabled if the host requires access to NFS volumes on an instance. To enable this boolean, run the following command as the `root` user:

```
# setsebool -P virt_use_nfs on
```

This command also makes the boolean persistent across reboots. Run this command on all client hosts that require access to NFS volumes on an instance. This includes all compute nodes.

Configure multiple-storage back ends

When you configure multiple-storage back ends, you can create several back-end storage solutions that serve the same OpenStack Compute configuration and one `cinder-volume` is launched for each back-end storage or back-end storage pool.

In a multiple-storage back-end configuration, each back end has a name (`volume_backend_name`). Several back ends can have the same name. In that case, the scheduler properly decides which back end the volume has to be created in.

The name of the back end is declared as an extra-specification of a volume type (such as, `volume_backend_name=LVM`). When a volume is created, the scheduler chooses an appropriate back end to handle the request, according to the volume type specified by the user.

Enable multiple-storage back ends

To enable a multiple-storage back ends, you must set the *enabled_backends* flag in the `cinder.conf` file. This flag defines the names (separated by a comma) of the configuration groups for the different back ends: one name is associated to one configuration group for a back end (such as, `[lvmdriver-1]`).

Note

The configuration group name is not related to the `volume_backend_name`.

Note

After setting the `enabled_backends` flag on an existing cinder service, and restarting the Block Storage services, the original `host` service is replaced with a new host service. The new service appears with a name like `host@backend`. Use:

```
$ cinder-manage volume update_host --currenthost CURRENTHOST --newhost␣
↪CURRENTHOST@BACKEND
```

to convert current block devices to the new host name.

The options for a configuration group must be defined in the group (or default options are used). All the standard Block Storage configuration options (`volume_group`, `volume_driver`, and so on) might be used in a configuration group. Configuration values in the `[DEFAULT]` configuration group are not used.

These examples show three back ends:

```
enabled_backends=lvmdriver-1,lvmdriver-2,lvmdriver-3
[lvmdriver-1]
volume_group=cinder-volumes-1
volume_driver=cinder.volume.drivers.lvm.LVMVolumeDriver
volume_backend_name=LVM
```

(continues on next page)

(continued from previous page)

```
[lvmdriver-2]
volume_group=cinder-volumes-2
volume_driver=cinder.volume.drivers.lvm.LVMVolumeDriver
volume_backend_name=LVM
[lvmdriver-3]
volume_group=cinder-volumes-3
volume_driver=cinder.volume.drivers.lvm.LVMVolumeDriver
volume_backend_name=LVM_b
```

In this configuration, `lvmdriver-1` and `lvmdriver-2` have the same `volume_backend_name`. If a volume creation requests the `LVM` back end name, the scheduler uses the capacity filter scheduler to choose the most suitable driver, which is either `lvmdriver-1` or `lvmdriver-2`. The capacity filter scheduler is enabled by default. The next section provides more information. In addition, this example presents a `lvmdriver-3` back end.

Note

For Fiber Channel drivers that support multipath, the configuration group requires the `use_multipath_for_image_xfer=true` option. In the example below, you can see details for HPE 3PAR and EMC Fiber Channel drivers.

```
[3par]
use_multipath_for_image_xfer = true
volume_driver = cinder.volume.drivers.hpe.hpe_3par_fc.HPE3PARFCDriver
volume_backend_name = 3parfc

[emc]
use_multipath_for_image_xfer = true
volume_driver = cinder.volume.drivers.emc.emc_smis_fc.EMCSMISFCDriver
volume_backend_name = emcfc
```

Configure shared volume driver backends

When configuring multiple volume backends, common configuration parameters can be shared using the *[backend_defaults]* section. As an example:

```
[DEFAULT]
enabled_backends=backend1,backend2,backend3

[backend_defaults]
image_volume_cache_enabled = True
volume_clear = none
target_helper = tgtadm
volume_driver = cinder.volume.drivers.lvm.LVMVolumeDriver

[backend1]
volume_group = cinder-volume-1
image_volume_cache_enabled = False
```

(continues on next page)

(continued from previous page)

```
[backend2]
volume_group = cinder-volume-2

[backend3]
volume_group = cinder-volume-3
```

In this configuration, `backend2` and `backend3` have the same `image_volume_cache_enabled` as it is defined in the `backend_defaults` section. In other words, `backend2` and `backend3` have enabled the image cache features. `image_volume_cache_enabled` in `backend1` is False, that means any overwritten configuration in a volume backend will ignore the original value in `backend_defaults`.

Note

The `backend_defaults` section should be configured according to your cloud environment or your backend driver information.

Configure Block Storage scheduler multi back end

You must enable the *filter_scheduler* option to use multiple-storage back ends. The filter scheduler:

1. Filters the available back ends. By default, `AvailabilityZoneFilter`, `CapacityFilter` and `CapabilitiesFilter` are enabled.
2. Weights the previously filtered back ends. By default, the *CapacityWeigher* option is enabled. When this option is enabled, the filter scheduler assigns the highest weight to back ends with the most available capacity.

The scheduler uses filters and weights to pick the best back end to handle the request. The scheduler uses volume types to explicitly create volumes on specific back ends. For more information about filter and weighing, see *Configure and use driver filter and weighing for scheduler*.

Volume type

Before using it, a volume type has to be declared to Block Storage. This can be done by the following command:

```
$ openstack --os-username admin --os-tenant-name admin volume type create lvm
```

Then, an extra-specification has to be created to link the volume type to a back end name. Run this command:

```
$ openstack --os-username admin --os-tenant-name admin volume type set lvm \
  --property volume_backend_name=LVM_iSCSI
```

This example creates a `lvm` volume type with `volume_backend_name=LVM_iSCSI` as extra-specifications.

Create another volume type:

```
$ openstack --os-username admin --os-tenant-name admin volume type create lvm_
↪gold
```

(continues on next page)

(continued from previous page)

```
$ openstack --os-username admin --os-tenant-name admin volume type set lvm_
↪gold \
  --property volume_backend_name=LVM_iSCSI_b
```

This second volume type is named `lvm_gold` and has `LVM_iSCSI_b` as back end name.

Note

To list the extra-specifications, use this command:

```
$ openstack --os-username admin --os-tenant-name admin volume type list --
↪long
```

Note

If a volume type points to a `volume_backend_name` that does not exist in the Block Storage configuration, the `filter_scheduler` returns an error that it cannot find a valid host with the suitable back end.

Usage

When you create a volume, you must specify the volume type. The extra-specifications of the volume type are used to determine which back end has to be used.

```
$ openstack volume create --size 1 --type lvm test_multi_backend
```

Considering the `cinder.conf` described previously, the scheduler creates this volume on `lvmdriver-1` or `lvmdriver-2`.

```
$ openstack volume create --size 1 --type lvm_gold test_multi_backend
```

This second volume is created on `lvmdriver-3`.

Back up Block Storage service disks

While you can use the LVM snapshot to create snapshots, you can also use it to back up your volumes. By using LVM snapshot, you reduce the size of the backup; only existing data is backed up instead of the entire volume.

To back up a volume, you must create a snapshot of it. An LVM snapshot is the exact copy of a logical volume, which contains data in a frozen state. This prevents data corruption because data cannot be manipulated during the volume creation process. Remember that the volumes created through an **`openstack volume create`** command exist in an LVM logical volume.

You must also make sure that the operating system is not using the volume and that all data has been flushed on the guest file systems. This usually means that those file systems have to be unmounted during the snapshot creation. They can be mounted again as soon as the logical volume snapshot has been created.

Before you create the snapshot you must have enough space to save it. As a precaution, you should have at least twice as much space as the potential snapshot size. If insufficient space is available, the snapshot might become corrupted.

For this example assume that a 100 GB volume named `volume-00000001` was created for an instance while only 4 GB are used. This example uses these commands to back up only those 4 GB:

- **lvm2** command. Directly manipulates the volumes.
- **kpartx** command. Discovers the partition table created inside the instance.
- **tar** command. Creates a minimum-sized backup.
- **sha1sum** command. Calculates the backup checksum to check its consistency.

You can apply this process to volumes of any size.

To back up Block Storage service disks

1. Create a snapshot of a used volume

   - Use this command to list all volumes

   ```
   # lvdisplay
   ```

   - Create the snapshot; you can do this while the volume is attached to an instance:

   ```
   # lvcreate --size 10G --snapshot --name volume-00000001-snapshot \
     /dev/cinder-volumes/volume-00000001
   ```

   Use the `--snapshot` configuration option to tell LVM that you want a snapshot of an already existing volume. The command includes the size of the space reserved for the snapshot volume, the name of the snapshot, and the path of an already existing volume. Generally, this path is `/dev/cinder-volumes/VOLUME_NAME`.

   The size does not have to be the same as the volume of the snapshot. The `--size` parameter defines the space that LVM reserves for the snapshot volume. As a precaution, the size should be the same as that of the original volume, even if the whole space is not currently used by the snapshot.

   - Run the **lvdisplay** command again to verify the snapshot:

   ```
   --- Logical volume ---
   LV Name                /dev/cinder-volumes/volume-00000001
   VG Name                cinder-volumes
   LV UUID                gI8hta-p21U-IW2q-hRN1-nTzN-UC2G-dKbdKr
   LV Write Access        read/write
   LV snapshot status     source of
                          /dev/cinder-volumes/volume-00000026-snap␣
   ↪[active]
   LV Status              available
   # open                 1
   LV Size                15,00 GiB
   Current LE             3840
   Segments               1
   Allocation             inherit
   Read ahead sectors     auto
   ```

(continues on next page)

(continued from previous page)

```
- currently set to     256
Block device           251:13

--- Logical volume ---
LV Name                /dev/cinder-volumes/volume-00000001-snap
VG Name                cinder-volumes
LV UUID                HlW3Ep-g5I8-KGQb-IRvi-IRYU-lIKe-wE9zYr
LV Write Access        read/write
LV snapshot status     active destination for /dev/cinder-volumes/
↪volume-00000026
LV Status              available
# open                 0
LV Size                15,00 GiB
Current LE             3840
COW-table size         10,00 GiB
COW-table LE           2560
Allocated to snapshot  0,00%
Snapshot chunk size    4,00 KiB
Segments               1
Allocation             inherit
Read ahead sectors     auto
- currently set to     256
Block device           251:14
```

2. Partition table discovery

   - To exploit the snapshot with the **tar** command, mount your partition on the Block Storage service server.

     The **kpartx** utility discovers and maps table partitions. You can use it to view partitions that are created inside the instance. Without using the partitions created inside instances, you cannot see its content and create efficient backups.

     ```
     # kpartx -av /dev/cinder-volumes/volume-00000001-snapshot
     ```

     Note

     On a Debian-based distribution, you can use the **apt-get install kpartx** command to install **kpartx**.

     If the tools successfully find and map the partition table, no errors are returned.

   - To check the partition table map, run this command:

     ```
     $ ls /dev/mapper/nova*
     ```

     You can see the `cinder--volumes-volume--00000001--snapshot1` partition.

     If you created more than one partition on that volume, you see several partitions; for example: `cinder--volumes-volume--00000001--snapshot2`, `cinder--volumes-volume--00000001--snapshot3`, and so on.

- Mount your partition

```
# mount /dev/mapper/cinder--volumes-volume--volume--00000001--
↪snapshot1 /mnt
```

If the partition mounts successfully, no errors are returned.

You can directly access the data inside the instance. If a message prompts you for a partition or you cannot mount it, determine whether enough space was allocated for the snapshot or the **kpartx** command failed to discover the partition table.

Allocate more space to the snapshot and try the process again.

3. Use the **tar** command to create archives

Create a backup of the volume:

```
$ tar --exclude="lost+found" --exclude="some/data/to/exclude" -czf \
volume-00000001.tar.gz -C /mnt/ /backup/destination
```

This command creates a `tar.gz` file that contains the data, *and data only*. This ensures that you do not waste space by backing up empty sectors.

4. Checksum calculation

You should always have the checksum for your backup files. When you transfer the same file over the network, you can run a checksum calculation to ensure that your file was not corrupted during its transfer. The checksum is a unique ID for a file. If the checksums are different, the file is corrupted.

Run this command to run a checksum for your file and save the result to a file:

```
$ sha1sum volume-00000001.tar.gz > volume-00000001.checksum
```

> **Note**
>
> Use the **sha1sum** command carefully because the time it takes to complete the calculation is directly proportional to the size of the file.
>
> Depending on your CPU, the process might take a long time for files larger than around 4 to 6 GB.

5. After work cleaning

Now that you have an efficient and consistent backup, use this command to clean up the file system:

- Unmount the volume.

```
$ umount /mnt
```

- Delete the partition table.

```
$ kpartx -dv /dev/cinder-volumes/volume-00000001-snapshot
```

- Remove the snapshot.

```
$ lvremove -f /dev/cinder-volumes/volume-00000001-snapshot
```

Repeat these steps for all your volumes.

6. Automate your backups

   Because more and more volumes might be allocated to your Block Storage service, you might want to automate your backups. The SCR_5005_V01_NUAC-OPENSTACK-EBS-volumes-backup.sh script assists you with this task. The script performs the operations from the previous example, but also provides a mail report and runs the backup based on the `backups_retention_days` setting.

   Launch this script from the server that runs the Block Storage service.

   This example shows a mail report:

```
Backup Start Time - 07/10 at 01:00:01
Current retention - 7 days

The backup volume is mounted. Proceed...
Removing old backups...  : /BACKUPS/EBS-VOL/volume-00000019/volume-
↪00000019_28_09_2011.tar.gz
     /BACKUPS/EBS-VOL/volume-00000019 - 0 h 1 m and 21 seconds. Size - 3,
↪5G

The backup volume is mounted. Proceed...
Removing old backups...  : /BACKUPS/EBS-VOL/volume-0000001a/volume-
↪0000001a_28_09_2011.tar.gz
     /BACKUPS/EBS-VOL/volume-0000001a - 0 h 4 m and 15 seconds. Size - 6,
↪9G
---------------------------------------
Total backups size - 267G - Used space : 35%
Total execution time - 1 h 75 m and 35 seconds
```

   The script also enables you to SSH to your instances and run a **mysqldump** command into them. To make this work, enable the connection to the Compute project keys. If you do not want to run the **mysqldump** command, you can add `enable_mysql_dump=0` to the script to turn off this functionality.

Migrate volumes

OpenStack has the ability to migrate volumes between back ends which support its volume-type. Migrating a volume transparently moves its data from the current back end for the volume to a new one. This is an administrator function, and can be used for functions including storage evacuation (for maintenance or decommissioning), or manual optimizations (for example, performance, reliability, or cost).

These workflows are possible for a migration:

1. If the storage can migrate the volume on its own, it is given the opportunity to do so. This allows the Block Storage driver to enable optimizations that the storage might be able to perform. If the back end is not able to perform the migration, the Block Storage uses one of two generic flows, as follows.

2. If the volume is not attached, the Block Storage service creates a volume and copies the data from the original to the new volume.

Note

While most back ends support this function, not all do. See the *driver documentation* for more details.

3. If the volume is attached to a VM instance, the Block Storage creates a volume, and calls Compute to copy the data from the original to the new volume. Currently this is supported only by the Compute libvirt driver.

As an example, this scenario shows two LVM back ends and migrates an attached volume from one to the other. This scenario uses the third migration flow.

First, list the available back ends:

```
# cinder get-pools
+----------+----------------------------------------------------+
| Property |                       Value                        |
+----------+----------------------------------------------------+
|   name   |           server1@lvmstorage-1#lvmstorage-1        |
+----------+----------------------------------------------------+
+----------+----------------------------------------------------+
| Property |                      Value                         |
+----------+----------------------------------------------------+
|   name   |           server2@lvmstorage-2#lvmstorage-2        |
+----------+----------------------------------------------------+
```

Note

Block Storage API supports **cinder get-pools** since V2 version.

You can also get available back ends like following:

```
# cinder-manage host list
server1@lvmstorage-1    zone1
server2@lvmstorage-2    zone1
```

But it needs to add pool name in the end. For example, `server1@lvmstorage-1#zone1`.

Next, as the admin user, you can see the current status of the volume (replace the example ID with your own):

```
$ openstack volume show 6088f80a-f116-4331-ad48-9afb0dfb196c

+--------------------------------+--------------------------------------+
| Field                          | Value                                |
+--------------------------------+--------------------------------------+
| attachments                    | []                                   |
| availability_zone              | zone1                                |
| bootable                       | false                                |
| consistencygroup_id            | None                                 |
| created_at                     | 2013-09-01T14:53:22.000000           |
```

(continues on next page)

(continued from previous page)

```
| description                    | test                                 |
| encrypted                      | False                                |
| id                             | 6088f80a-f116-4331-ad48-9afb0dfb196c |
| migration_status               | None                                 |
| multiattach                    | False                                |
| name                           | test                                 |
| os-vol-host-attr:host          | server1@lvmstorage-1#lvmstorage-1    |
| os-vol-mig-status-attr:migstat | None                                 |
| os-vol-mig-status-attr:name_id | None                                 |
| os-vol-tenant-attr:tenant_id   | d88310717a8e4ebcae84ed075f82c51e     |
| properties                     | readonly='False'                     |
| replication_status             | disabled                             |
| size                           | 1                                    |
| snapshot_id                    | None                                 |
| source_volid                   | None                                 |
| status                         | in-use                               |
| type                           | None                                 |
| updated_at                     | 2016-07-31T07:22:19.000000           |
| user_id                        | d8e5e5727f3a4ce1886ac8ecec058e83     |
+--------------------------------+--------------------------------------+
```

Note these attributes:

- `os-vol-host-attr:host` - the volumes current back end.
- `os-vol-mig-status-attr:migstat` - the status of this volumes migration (None means that a migration is not currently in progress).
- `os-vol-mig-status-attr:name_id` - the volume ID that this volumes name on the back end is based on. Before a volume is ever migrated, its name on the back end storage may be based on the volumes ID (see the `volume_name_template` configuration parameter). For example, if `volume_name_template` is kept as the default value (`volume-%s`), your first LVM back end has a logical volume named `volume-6088f80a-f116-4331-ad48-9afb0dfb196c`. During the course of a migration, if you create a volume and copy over the data, the volume get the new name but keeps its original ID. This is exposed by the `name_id` attribute.

Note

If you plan to decommission a block storage node, you must stop the `cinder` volume service on the node after performing the migration.

On nodes that run CentOS, Fedora, openSUSE, Red Hat Enterprise Linux, or SUSE Linux Enterprise, run:

```
# service openstack-cinder-volume stop
# chkconfig openstack-cinder-volume off
```

On nodes that run Ubuntu or Debian, run:

```
# service cinder-volume stop
# chkconfig cinder-volume off
```

Stopping the cinder volume service will prevent volumes from being allocated to the node.

Migrate this volume to the second LVM back end:

```
$ openstack volume migrate 6088f80a-f116-4331-ad48-9afb0dfb196c \
  --host server2@lvmstorage-2#lvmstorage-2
```

You can use the **openstack volume show** command to see the status of the migration. While migrating, the `migstat` attribute shows states such as `migrating` or `completing`. On error, `migstat` is set to None and the host attribute shows the original `host`. On success, in this example, the output looks like:

```
$ openstack volume show 6088f80a-f116-4331-ad48-9afb0dfb196c

+--------------------------------+--------------------------------------+
| Field                          | Value                                |
+--------------------------------+--------------------------------------+
| attachments                    | []                                   |
| availability_zone              | zone1                                |
| bootable                       | false                                |
| consistencygroup_id            | None                                 |
| created_at                     | 2013-09-01T14:53:22.000000           |
| description                    | test                                 |
| encrypted                      | False                                |
| id                             | 6088f80a-f116-4331-ad48-9afb0dfb196c |
| migration_status               | None                                 |
| multiattach                    | False                                |
| name                           | test                                 |
| os-vol-host-attr:host          | server2@lvmstorage-2#lvmstorage-2    |
| os-vol-mig-status-attr:migstat | completing                           |
| os-vol-mig-status-attr:name_id | None                                 |
| os-vol-tenant-attr:tenant_id   | d88310717a8e4ebcae84ed075f82c51e     |
| properties                     | readonly='False'                     |
| replication_status             | disabled                             |
| size                           | 1                                    |
| snapshot_id                    | None                                 |
| source_volid                   | None                                 |
| status                         | in-use                               |
| type                           | None                                 |
| updated_at                     | 2017-02-22T02:35:03.000000           |
| user_id                        | d8e5e5727f3a4ce1886ac8ecec058e83     |
+--------------------------------+--------------------------------------+
```

Note that `migstat` is None, host is the new host, and `name_id` holds the ID of the volume created by the migration. If you look at the second LVM back end, you find the logical volume `volume-133d1f56-9ffc-4f57-8798-d5217d851862`.

Note

The migration is not visible to non-admin users (for example, through the volume `status`). However, some operations are not allowed while a migration is taking place, such as attaching/detaching a volume and deleting a volume. If a user performs such an action during a migration, an error is returned.

Note

Migrating volumes that have snapshots are currently not allowed.

Back up and restore volumes and snapshots

The openstack command-line interface provides the tools for creating a volume backup. You can restore a volume from a backup as long as the backups associated database information (or backup metadata) is intact in the Block Storage database.

Run this command to create a backup of a volume:

```
$ openstack volume backup create [--incremental] [--force] VOLUME
```

Where VOLUME is the name or ID of the volume, incremental is a flag that indicates whether an incremental backup should be performed, and force is a flag that allows or disallows backup of a volume when the volume is attached to an instance.

Without the incremental flag, a full backup is created by default. With the incremental flag, an incremental backup is created.

Without the force flag, the volume will be backed up only if its status is available. With the force flag, the volume will be backed up whether its status is available or in-use. A volume is in-use when it is attached to an instance. The backup of an in-use volume means your data is crash consistent. The force flag is False by default.

Note

The force flag is new in OpenStack Liberty.

The incremental backup is based on a parent backup which is an existing backup with the latest timestamp. The parent backup can be a full backup or an incremental backup depending on the timestamp.

Note

The first backup of a volume has to be a full backup. Attempting to do an incremental backup without any existing backups will fail. There is an is_incremental flag that indicates whether a backup is incremental when showing details on the backup. Another flag, has_dependent_backups, returned when showing backup details, will indicate whether the backup has dependent backups. If it is true, attempting to delete this backup will fail.

A new configure option backup_swift_block_size is introduced into cinder.conf for the default Swift backup driver. This is the size in bytes that changes are tracked for incremental backups. The existing backup_swift_object_size option, the size in bytes of Swift backup objects, has to be a multiple of backup_swift_block_size. The default is 32768 for backup_swift_block_size, and the default is 52428800 for backup_swift_object_size.

The configuration option backup_swift_enable_progress_timer in cinder.conf is used when backing up the volume to Object Storage back end. This option enables or disables the timer. It is enabled by default to send the periodic progress notifications to the Telemetry service.

This command also returns a backup ID. Use this backup ID when restoring the volume:

```
$ openstack volume backup restore BACKUP_ID VOLUME_ID
```

When restoring from a full backup, it is a full restore.

When restoring from an incremental backup, a list of backups is built based on the IDs of the parent backups. A full restore is performed based on the full backup first, then restore is done based on the incremental backup, laying on top of it in order.

You can view a backup list with the **openstack volume backup list** command. Optional arguments to clarify the status of your backups include: running `--name`, `--status`, and `--volume` to filter through backups by the specified name, status, or volume-id. Search with `--all-projects` for details of the projects associated with the listed backups.

Because volume backups are dependent on the Block Storage database, you must also back up your Block Storage database regularly to ensure data recovery.

Note

Alternatively, you can export and save the metadata of selected volume backups. Doing so precludes the need to back up the entire Block Storage database. This is useful if you need only a small subset of volumes to survive a catastrophic database failure.

If you specify a UUID encryption key when setting up the volume specifications, the backup metadata ensures that the key will remain valid when you back up and restore the volume.

For more information about how to export and import volume backup metadata, see the section called *Export and import backup metadata*.

By default, the swift object store is used for the backup repository.

If instead you want to use an NFS export as the backup repository, add the following configuration options to the `[DEFAULT]` section of the `cinder.conf` file and restart the Block Storage services:

```
backup_driver = cinder.backup.drivers.nfs
backup_share = HOST:EXPORT_PATH
```

For the `backup_share` option, replace `HOST` with the DNS resolvable host name or the IP address of the storage server for the NFS share, and `EXPORT_PATH` with the path to that share. If your environment requires that non-default mount options be specified for the share, set these as follows:

```
backup_mount_options = MOUNT_OPTIONS
```

`MOUNT_OPTIONS` is a comma-separated string of NFS mount options as detailed in the NFS man page.

There are several other options whose default values may be overridden as appropriate for your environment:

```
backup_compression_algorithm = zlib
backup_sha_block_size_bytes = 32768
backup_file_size = 1999994880
```

The option `backup_compression_algorithm` can be set to `zlib`, `bz2`, `zstd` or `none`. The value `none` can be a useful setting when the server providing the share for the backup repository itself performs deduplication or compression on the backup data.

The option backup_file_size must be a multiple of backup_sha_block_size_bytes. It is effectively the maximum file size to be used, given your environment, to hold backup data. Volumes larger than this will be stored in multiple files in the backup repository. backup_file_size also determines the buffer size used to produce backup files; on smaller hosts it may need to be scaled down to avoid OOM issues. The backup_sha_block_size_bytes option determines the size of blocks from the cinder volume being backed up on which digital signatures are calculated in order to enable incremental backup capability.

You also have the option of resetting the state of a backup. When creating or restoring a backup, sometimes it may get stuck in the creating or restoring states due to problems like the database or rabbitmq being down. In situations like these resetting the state of the backup can restore it to a functional status.

Run this command to restore the state of a backup:

```
$ openstack volume backup set --state <state> BACKUP
```

Run this command to create a backup of a snapshot:

```
$ openstack volume backup create [--incremental] [--force] \
  [--snapshot SNAPSHOT_ID] VOLUME
```

Where VOLUME is the name or ID of the volume, SNAPSHOT_ID is the ID of the volumes snapshot.

Cancelling

Since Liberty it is possible to cancel an ongoing backup operation on any of the Chunked Backup type of drivers such as Swift, NFS, Google, GlusterFS, and Posix.

To issue a backup cancellation on a backup we must request a force delete on the backup.

```
$ openstack volume backup delete --force BACKUP_ID
```

Note

The policy on force delete defaults to admin only.

Even if the backup is immediately deleted, and therefore no longer appears in the listings, the cancellation may take a little bit longer, so please check the status of the source resource to see when it stops being backing-up.

Note

Before Pike the backing-up status would always be stored in the volume, even when backing up a snapshot, so when backing up a snapshot any delete operation on the snapshot that followed a cancellation could result in an error if the snapshot was still mapped. Polling on the volume to stop being backing-up prior to the deletion is required to ensure success.

Since Rocky it is also possible to cancel an ongoing restoring operation on any of the Chunked Backup type of drivers.

To issue a backup restoration cancellation we need to alter its status to anything other than *restoring*. We strongly recommend using the error state to avoid any confusion on whether the restore was successful or not.

```
$ openstack volume backup set --state error BACKUP_ID
```

Warning

After a restore operation has started, if it is then cancelled, the destination volume is useless, as there is no way of knowing how much data, or if any, was actually restored, hence our recommendation of using the error state.

backup_max_operations

With this configuration option will let us select the maximum number of operations, backup and restore, that can be performed concurrently.

This option has a default value of 15, which means that we can have 15 concurrent backups, or 15 concurrent restores, or any combination of backups and restores as long as the sum of the 2 operations dont exceed 15.

The concurrency limitation of this configuration option is also enforced when we run multiple processes for the same backup service using the `backup_workers` configuration option. It is not a per process restriction, but global to the service, so we wont be able to run `backup_max_operations` on each one of the processes, but on all the running processes from the same backup service.

Backups and restore operations are both CPU and memory intensive, but thanks to this option we can limit the concurrency and prevent DoS attacks or just service disruptions caused by many concurrent requests that lead to Out of Memory (OOM) kills.

The amount of memory (RAM) used during the operation depends on the configured chunk size as well as the compression ratio achieved on the data during the operation.

Example:

> Lets have a look at how much memory would be needed if we use the default backup chunk size (~1.86 GB) while doing a restore to an RBD volume from a non Ceph backend (Swift, NFS etc).
>
> In a restore operation the worst case scenario, from the memory point of view, is when the compression ratio is close to 0% (the compressed data chunk is almost the same size as the uncompressed data).
>
> In this case the memory usage would be ~5.58 GB of data for each chunk: ~5.58 GB = read buffer + decompressed buffer + write buffer used by the librbd library = ~1.86 GB + 1.86 GB + 1.86 GB
>
> For 15 concurrent restore operations, the cinder-backup service will require ~83.7 GB of memory.

Similar calculations can be done for environment specific scenarios and this config option can be set accordingly.

Export and import backup metadata

A volume backup can only be restored on the same Block Storage service. This is because restoring a volume from a backup requires metadata available on the database used by the Block Storage service.

> **Note**
>
> For information about how to back up and restore a volume, see the section called *Back up and restore volumes and snapshots*.

You can, however, export the metadata of a volume backup. To do so, run this command as an OpenStack `admin` user (presumably, after creating a volume backup):

```
$ cinder backup-export BACKUP_ID
```

Where `BACKUP_ID` is the volume backups ID. This command should return the backups corresponding database information as encoded string metadata.

Exporting and storing this encoded string metadata allows you to completely restore the backup, even in the event of a catastrophic database failure. This will preclude the need to back up the entire Block Storage database, particularly if you only need to keep complete backups of a small subset of volumes.

If you have placed encryption on your volumes, the encryption will still be in place when you restore the volume if a UUID encryption key is specified when creating volumes. Using backup metadata support, UUID keys set up for a volume (or volumes) will remain valid when you restore a backed-up volume. The restored volume will remain encrypted, and will be accessible with your credentials.

In addition, having a volume backup and its backup metadata also provides volume portability. Specifically, backing up a volume and exporting its metadata will allow you to restore the volume on a completely different Block Storage database, or even on a different cloud service. To do so, first import the backup metadata to the Block Storage database and then restore the backup.

To import backup metadata, run the following command as an OpenStack `admin`:

```
$ cinder backup-import METADATA
```

Where `METADATA` is the backup metadata exported earlier.

Once you have imported the backup metadata into a Block Storage database, restore the volume (see the section called *Back up and restore volumes and snapshots*).

Use LIO iSCSI support

The default mode for the `target_helper` tool is `tgtadm`. To use LIO iSCSI, install the `python-rtslib` package, and set `target_helper=lioadm` in the `cinder.conf` file.

Once configured, you can use the **`cinder-rtstool`** command to manage the volumes. This command enables you to create, delete, and verify volumes and determine targets and add iSCSI initiators to the system.

Configure and use volume number weigher

OpenStack Block Storage enables you to choose a volume back end according to `free_capacity` and `allocated_capacity`. The volume number weigher feature lets the scheduler choose a volume back end based on its volume number in the volume back end. This can provide another means to improve the volume back ends I/O balance and the volumes I/O performance.

Enable volume number weigher

To enable a volume number weigher, set the `scheduler_default_weighers` to `VolumeNumberWeigher` flag in the `cinder.conf` file to define `VolumeNumberWeigher` as the selected weigher.

Configure multiple-storage back ends

To configure `VolumeNumberWeigher`, use `LVMVolumeDriver` as the volume driver.

This configuration defines two LVM volume groups: `stack-volumes` with 10 GB capacity and `stack-volumes-1` with 60 GB capacity. This example configuration defines two back ends:

```
scheduler_default_weighers=VolumeNumberWeigher
enabled_backends=lvmdriver-1,lvmdriver-2
[lvmdriver-1]
volume_group=stack-volumes
volume_driver=cinder.volume.drivers.lvm.LVMVolumeDriver
volume_backend_name=LVM

[lvmdriver-2]
volume_group=stack-volumes-1
volume_driver=cinder.volume.drivers.lvm.LVMVolumeDriver
volume_backend_name=LVM
```

Volume type

Define a volume type in Block Storage:

```
$ openstack volume type create lvm
```

Create an extra specification that links the volume type to a back-end name:

```
$ openstack volume type set lvm --property volume_backend_name=LVM
```

This example creates a lvm volume type with `volume_backend_name=LVM` as extra specifications.

Usage

To create six 1-GB volumes, run the **`openstack volume create --size 1 --type lvm volume1`** command six times:

```
$ openstack volume create --size 1 --type lvm volume1
```

This command creates three volumes in `stack-volumes` and three volumes in `stack-volumes-1`.

List the available volumes:

```
# lvs
LV                                         VG              Attr      LSize ␣
↪Pool Origin Data%  Move Log Copy%  Convert
volume-3814f055-5294-4796-b5e6-1b7816806e5d stack-volumes   -wi-a----  1.00g
volume-72cf5e79-99d2-4d23-b84e-1c35d3a293be stack-volumes   -wi-a----  1.00g
volume-96832554-0273-4e9d-902b-ad421dfb39d1 stack-volumes   -wi-a----  1.00g
volume-169386ef-3d3e-4a90-8439-58ceb46889d9 stack-volumes-1 -wi-a----  1.00g
volume-460b0bbb-d8a0-4bc3-9882-a129a5fe8652 stack-volumes-1 -wi-a----  1.00g
volume-9a08413b-0dbc-47c9-afb8-41032ab05a41 stack-volumes-1 -wi-a----  1.00g
```

Capacity based quality of service

In many environments, the performance of the storage system which Cinder manages scales with the storage space in the cluster. For example, a Ceph RBD cluster could have a capacity of 10,000 IOPs and 1000 GB storage. However, as the RBD cluster scales to 2000 GB, the IOPs scale to 20,000 IOPs.

Basic QoS allows you to define hard limits for volumes, however, if you have a limit of 1000 IOPs for a volume and you have a user which creates 10x 1GB volumes with 1000 IOPs (in a cluster with 1000GB storage and 10,000 IOPs), youre not able to guarantee the quality of service without having to add extra capacity (which will go un-used). The inverse can be problematic, if a user creates a 1000GB volume with 1000 IOPs, leaving 9000 un-used IOPs.

Capacity based quality of service allows you to multiply the quality of service values by the size of the volume, which will allow you to efficiently use the storage managed by Cinder. In some cases, it will force the user to provision a larger volume than they need to get the IOPs they need, but that extra space would have gone un-used if they didnt use it in order to deliver the quality of service.

There are currently 6 options to control capacity based quality of service which values should be fairly self explanatory:

For dynamic IOPS per volume.

- `read_iops_sec_per_gb`
- `write_iops_sec_per_gb`
- `total_iops_sec_per_gb`

For dynamic bandwidth per volume.

- `read_bytes_sec_per_gb`
- `write_bytes_sec_per_gb`
- `total_bytes_sec_per_gb`

In addition, there are 6 more options which allow you to control the minimum possible value. This can be useful in cases where a user creates a volume that is very small and ends up with an unusable volume because of performance.

For minimum IOPS per volume.

- `read_iops_sec_per_gb_min`
- `write_iops_sec_per_gb_min`
- `total_iops_sec_per_gb_min`

For minimum bandwidth per volume.

- `read_bytes_sec_per_gb_min`
- `write_bytes_sec_per_gb_min`
- `total_bytes_sec_per_gb_min`

Capacity based options might be used in conjunction with basic options, like `*_sec_max`, in order to set upper limits for volumes. This may be useful for large volumes, which may consume all storage performance.

For example, in order to create a QoS with 30 IOPs total writes per GB and a throughput of 1MB per GB, you might use the Cinder client in the following way:

```
$ cinder qos-create high-iops consumer="front-end" \
  total_iops_sec_per_gb=30 total_bytes_sec_per_gb=1048576
+-----------+-------------------------------------+
| Property  | Value                               |
+-----------+-------------------------------------+
| consumer  | front-end                           |
| id        | f448f61c-4238-4eef-a93a-2024253b8f75 |
| name      | high-iops                           |
| specs     | total_iops_sec_per_gb : 30          |
|           | total_bytes_sec_per_gb : 1048576    |
+-----------+-------------------------------------+
```

Once this is done, you can associate this QoS with a volume type by using the `qos-associate` Cinder client command.

```
$ cinder qos-associate <qos-id> <volume-type-id>
```

You can now create a new volume and attempt to attach it to a consumer such as Nova. If you login to a Nova compute host, youll be able to see the new calculated limits when checking the XML definition of the virtual machine with `virsh dumpxml`.

Consistency groups

Consistency group support is available in OpenStack Block Storage. The support is added for creating snapshots of consistency groups. This feature leverages the storage level consistency technology. It allows snapshots of multiple volumes in the same consistency group to be taken at the same point-in-time to ensure data consistency. The consistency group operations can be performed using the Block Storage command line.

> **Note**
>
> The Consistency Group APIs have been deprecated since the Queens release. Use the Generic Volume Group APIs instead.
>
> The Consistency Group APIs are governed by the same policies as the Generic Volume Group APIs. For information about configuring cinder policies, see *Policy configuration*.

Before using consistency groups, make sure the Block Storage driver that you are running has consistency group support by reading the Block Storage manual or consulting the driver maintainer. There are a small number of drivers that have implemented this feature. The default LVM driver does not support consistency groups yet because the consistency technology is not available at the storage level.

The following consistency group operations are supported:

- Create a consistency group, given volume types.

> **Note**
>
> A consistency group can support more than one volume type. The scheduler is responsible for finding a back end that can support all given volume types.
>
> A consistency group can only contain volumes hosted by the same back end.
>
> A consistency group is empty upon its creation. Volumes need to be created and added to it later.

- Show a consistency group.
- List consistency groups.
- Create a volume and add it to a consistency group, given volume type and consistency group id.
- Create a snapshot for a consistency group.
- Show a snapshot of a consistency group.
- List consistency group snapshots.
- Delete a snapshot of a consistency group.
- Delete a consistency group.
- Modify a consistency group.
- Create a consistency group from the snapshot of another consistency group.
- Create a consistency group from a source consistency group.

The following operations are not allowed if a volume is in a consistency group:

- Volume migration.
- Volume retype.
- Volume deletion.

> **Note**
>
> A consistency group has to be deleted as a whole with all the volumes.

The following operations are not allowed if a volume snapshot is in a consistency group snapshot:

- Volume snapshot deletion.

> **Note**
>
> A consistency group snapshot has to be deleted as a whole with all the volume snapshots.

The details of consistency group operations are shown in the following.

Note

Currently, no OpenStack client command is available to run in place of the cinder consistency group creation commands. Use the cinder commands detailed in the following examples.

Create a consistency group:

```
cinder consisgroup-create
[--name name]
[--description description]
[--availability-zone availability-zone]
volume-types
```

Note

The parameter `volume-types` is required. It can be a list of names or UUIDs of volume types separated by commas without spaces in between. For example, `volumetype1,volumetype2,volumetype3.`.

```
$ cinder consisgroup-create --name bronzeCG2 volume_type_1

+-------------------+--------------------------------------+
|      Property     |                Value                 |
+-------------------+--------------------------------------+
| availability_zone |                 nova                 |
|     created_at    |      2014-12-29T12:59:08.000000      |
|    description    |                 None                 |
|         id        | 1de80c27-3b2f-47a6-91a7-e867cbe36462 |
|        name       |               bronzeCG2              |
|       status      |                creating              |
+-------------------+--------------------------------------+
```

Show a consistency group:

```
$ cinder consisgroup-show 1de80c27-3b2f-47a6-91a7-e867cbe36462

+-------------------+--------------------------------------+
|      Property     |                Value                 |
+-------------------+--------------------------------------+
| availability_zone |                 nova                 |
|     created_at    |      2014-12-29T12:59:08.000000      |
|    description    |                 None                 |
|         id        | 2a6b2bda-1f43-42ce-9de8-249fa5cbae9a |
|        name       |               bronzeCG2              |
|       status      |               available              |
|    volume_types   |             volume_type_1            |
+-------------------+--------------------------------------+
```

List consistency groups:

```
$ cinder consisgroup-list

+--------------------------------------+-----------+-----------+
|                  ID                  |   Status  |    Name   |
+--------------------------------------+-----------+-----------+
| 1de80c27-3b2f-47a6-91a7-e867cbe36462 | available | bronzeCG2 |
| 3a2b3c42-b612-479a-91eb-1ed45b7f2ad5 |   error   |  bronzeCG |
+--------------------------------------+-----------+-----------+
```

Create a volume and add it to a consistency group:

Note

When creating a volume and adding it to a consistency group, a volume type and a consistency group id must be provided. This is because a consistency group can support more than one volume type.

```
$ openstack volume create --type volume_type_1 --consistency-group \
  1de80c27-3b2f-47a6-91a7-e867cbe36462 --size 1 cgBronzeVol

+---------------------------------------+--------------------------------------
↪-+
| Field                                 | Value                                 ␣
↪ |
+---------------------------------------+--------------------------------------
↪-+
|              attachments              |                  []                   ␣
↪ |
|           availability_zone           |                 nova                  ␣
↪ |
|                bootable               |                false                  ␣
↪ |
|          consistencygroup_id          | 1de80c27-3b2f-47a6-91a7-
↪e867cbe36462 |
|               created_at              |      2014-12-29T13:16:47.000000       ␣
↪ |
|              description              |                 None                  ␣
↪ |
|               encrypted               |                False                  ␣
↪ |
|                   id                  | 5e6d1386-4592-489f-a56b-
↪9394a81145fe |
|                metadata               |                  {}                   ␣
↪ |
|                  name                 |             cgBronzeVol               ␣
↪ |
|         os-vol-host-attr:host         |      server-1@backend-1#pool-1        ␣
↪ |
|     os-vol-mig-status-attr:migstat    |                 None                  ␣
↪ |
```

(continues on next page)

(continued from previous page)

```
|     os-vol-mig-status-attr:name_id     |                 None                 ␣
↪ |
|      os-vol-tenant-attr:tenant_id      |   1349b21da2a046d8aa5379f0ed447bed ␣
↪ |
|   os-volume-replication:driver_data    |                 None                 ␣
↪ |
| os-volume-replication:extended_status  |                 None                 ␣
↪ |
|           replication_status           |               disabled               ␣
↪ |
|                  size                  |                  1                   ␣
↪ |
|              snapshot_id               |                 None                 ␣
↪ |
|              source_volid              |                 None                 ␣
↪ |
|                 status                 |               creating               ␣
↪ |
|                user_id                 |   93bdea12d3e04c4b86f9a9f172359859 ␣
↪ |
|              volume_type               |             volume_type_1            ␣
↪ |
+----------------------------------------+--------------------------------------
↪-+
```

Create a snapshot for a consistency group:

```
$ cinder cgsnapshot-create 1de80c27-3b2f-47a6-91a7-e867cbe36462

+---------------------+--------------------------------------+
|       Property      |                Value                 |
+---------------------+--------------------------------------+
| consistencygroup_id | 1de80c27-3b2f-47a6-91a7-e867cbe36462 |
|      created_at     |      2014-12-29T13:19:44.000000      |
|     description     |                 None                 |
|          id         | d4aff465-f50c-40b3-b088-83feb9b349e9 |
|         name        |                 None                 |
|        status       |               creating               |
+---------------------+-------------------------------------+
```

Show a snapshot of a consistency group:

```
$ cinder cgsnapshot-show d4aff465-f50c-40b3-b088-83feb9b349e9
```

List consistency group snapshots:

```
$ cinder cgsnapshot-list

+--------------------------------------+--------+----------+
|                  ID                  | Status | Name     |
```

(continues on next page)

(continued from previous page)

```
+--------------------------------------+--------+----------+
| 6d9dfb7d-079a-471e-b75a-6e9185ba0c38 | available  | None |
| aa129f4d-d37c-4b97-9e2d-7efffda29de0 | available  | None |
| bb5b5d82-f380-4a32-b469-3ba2e299712c | available  | None |
| d4aff465-f50c-40b3-b088-83feb9b349e9 | available  | None |
+--------------------------------------+--------+----------+
```

Delete a snapshot of a consistency group:

```
$ cinder cgsnapshot-delete d4aff465-f50c-40b3-b088-83feb9b349e9
```

Delete a consistency group:

Note

The force flag is needed when there are volumes in the consistency group:

```
$ cinder consisgroup-delete --force 1de80c27-3b2f-47a6-91a7-e867cbe36462
```

Modify a consistency group:

```
cinder consisgroup-update
[--name NAME]
[--description DESCRIPTION]
[--add-volumes UUID1,UUID2,......]
[--remove-volumes UUID3,UUID4,......]
CG
```

The parameter CG is required. It can be a name or UUID of a consistency group. UUID1,UUID2, are UUIDs of one or more volumes to be added to the consistency group, separated by commas. Default is None. UUID3,UUID4, are UUIDs of one or more volumes to be removed from the consistency group, separated by commas. Default is None.

```
$ cinder consisgroup-update --name 'new name' \
  --description 'new description' \
  --add-volumes 0b3923f5-95a4-4596-a536-914c2c84e2db,1c02528b-3781-4e32-929c-
↪618d81f52cf3 \
  --remove-volumes 8c0f6ae4-efb1-458f-a8fc-9da2afcc5fb1,a245423f-bb99-4f94-
↪8c8c-02806f9246d8 \
  1de80c27-3b2f-47a6-91a7-e867cbe36462
```

Create a consistency group from the snapshot of another consistency group:

```
$ cinder consisgroup-create-from-src
[--cgsnapshot CGSNAPSHOT]
[--name NAME]
[--description DESCRIPTION]
```

The parameter CGSNAPSHOT is a name or UUID of a snapshot of a consistency group:

```
$ cinder consisgroup-create-from-src \
  --cgsnapshot 6d9dfb7d-079a-471e-b75a-6e9185ba0c38 \
  --name 'new cg' --description 'new cg from cgsnapshot'
```

Create a consistency group from a source consistency group:

```
$ cinder consisgroup-create-from-src
[--source-cg SOURCECG]
[--name NAME]
[--description DESCRIPTION]
```

The parameter `SOURCECG` is a name or UUID of a source consistency group:

```
$ cinder consisgroup-create-from-src \
  --source-cg 6d9dfb7d-079a-471e-b75a-6e9185ba0c38 \
  --name 'new cg' --description 'new cloned cg'
```

Configure and use driver filter and weighing for scheduler

OpenStack Block Storage enables you to choose a volume back end based on back-end specific properties by using the DriverFilter and GoodnessWeigher for the scheduler. The driver filter and weigher scheduling can help ensure that the scheduler chooses the best back end based on requested volume properties as well as various back-end specific properties.

What is driver filter and weigher and when to use it

The driver filter and weigher gives you the ability to more finely control how the OpenStack Block Storage scheduler chooses the best back end to use when handling a volume request. One example scenario where using the driver filter and weigher can be if a back end that utilizes thin-provisioning is used. The default filters use the `free capacity` property to determine the best back end, but that is not always perfect. If a back end has the ability to provide a more accurate back-end specific value you can use that as part of the weighing. Another example of when the driver filter and weigher can prove useful is if a back end exists where there is a hard limit of 1000 volumes. The maximum volume size is 500ăGB. Once 75% of the total space is occupied the performance of the back end degrades. The driver filter and weigher can provide a way for these limits to be checked for.

Enable driver filter and weighing

To enable the driver filter, set the `scheduler_default_filters` option in the `cinder.conf` file to `DriverFilter`. The DriverFilter can also be used along with other filters by adding it to the list if other filters are already present.

To enable the goodness filter as a weigher, set the `scheduler_default_weighers` option in the `cinder.conf` file to `GoodnessWeigher` or add it to the list if other weighers are already present.

You can choose to use the `DriverFilter` without the `GoodnessWeigher` or vice-versa. The filter and weigher working together, however, create the most benefits when helping the scheduler choose an ideal back end.

Important

The GoodnessWeigher can be used along with CapacityWeigher and others, but must be used with caution as it might obfuscate the CapacityWeigher.

Example `cinder.conf` configuration file:

```
scheduler_default_filters = DriverFilter
scheduler_default_weighers = GoodnessWeigher
```

Note

It is useful to use the other filters and weighers available in OpenStack in combination with these custom ones. For example, the `CapacityFilter` and `CapacityWeigher` can be combined with these. Using them together should be done with caution as depending on the defined logic, one might obfuscate the other.

Defining your own filter and goodness functions

You can define your own filter and goodness functions through the use of various properties that OpenStack Block Storage has exposed. Properties exposed include information about the volume request being made, `volume_type` settings, and back-end specific information about drivers. All of these allow for a lot of control over how the ideal back end for a volume request will be decided.

The `filter_function` option is a string defining an equation that will determine whether a back end should be considered as a potential candidate in the scheduler.

The `goodness_function` option is a string defining an equation that will rate the quality of the potential host (0 to 100, 0 lowest, 100 highest).

Important

The drive filter and weigher will use default values for filter and goodness functions for each back end if you do not define them yourself. If complete control is desired then a filter and goodness function should be defined for each of the back ends in the `cinder.conf` file.

Supported operations in filter and goodness functions

Below is a table of all the operations currently usable in custom filter and goodness functions created by you:

Operations	Type
+, -, *, /, ^	standard math
not, and, or, &, |, !	logic
>, >=, <, <=, ==, <>, !=	equality
+, -	sign
x ? a : b	ternary
abs(x), max(x, y), min(x, y)	math helper functions

Caution

Syntax errors you define in filter or goodness strings are thrown at a volume request time.

Available properties when creating custom functions

There are various properties that can be used in either the `filter_function` or the `goodness_function` strings. The properties allow access to volume info, qos settings, extra specs, and so on.

The following properties and their sub-properties are currently available for use:

Host stats for a back end

In order to access these properties, use the following format: `stats.<property>`

host
The hosts name

volume_backend_name
The volume back end name

vendor_name
The vendor name

driver_version
The driver version

storage_protocol
The storage protocol

QoS_support
Boolean signifying whether QoS is supported

total_capacity_gb
The total capacity in GB

allocated_capacity_gb
The allocated capacity in GB

free_capacity_gb
The free capacity in GB

reserved_percentage
The reserved storage percentage

Capabilities specific to a back end

These properties are determined by the specific back end you are creating filter and goodness functions for. Some back ends may not have any properties available here. Once the capabilities vary too much according to the backend, it is better to check its properties reported on the scheduler log. The scheduler reports these capabilities constantly. In order to access these properties, use the following format: `capabilities.<property>`

Requested volume properties

In order to access the volume properties, use the following format: `volume.<property>`

status
: Status for the requested volume

volume_type_id
: The volume type ID

display_name
: The display name of the volume

volume_metadata
: Any metadata the volume has

reservations
: Any reservations the volume has

user_id
: The volumes user ID

attach_status
: The attach status for the volume

display_description
: The volumes display description

id
: The volumes ID

replication_status
: The volumes replication status

snapshot_id
: The volumes snapshot ID

encryption_key_id
: The volumes encryption key ID

source_volid
: The source volume ID

volume_admin_metadata
: Any admin metadata for this volume

source_replicaid
: The source replication ID

consistencygroup_id
: The consistency group ID

size
: The size of the volume in GB

metadata
: General metadata

The property most used from here will most likely be the `size` sub-property.

Extra specs for the requested volume type

View the available properties for volume types by running:

```
$ cinder extra-specs-list
```

Current QoS specs for the requested volume type

View the available properties for volume types by running:

```
$ openstack volume qos list
```

In order to access these properties in a custom string use the following format:

`<property>.<sub_property>`

Driver filter and weigher usage examples

Below are examples for using the filter and weigher separately, together, and using driver-specific properties.

Example `cinder.conf` file configuration for customizing the filter function:

```
[default]
scheduler_default_filters = DriverFilter
enabled_backends = lvm-1, lvm-2

[lvm-1]
volume_driver = cinder.volume.drivers.lvm.LVMVolumeDriver
volume_backend_name = sample_LVM01
filter_function = "volume.size < 10"

[lvm-2]
volume_driver = cinder.volume.drivers.lvm.LVMVolumeDriver
volume_backend_name = sample_LVM02
filter_function = "volume.size >= 10"
```

The above example will filter volumes to different back ends depending on the size of the requested volume. Default OpenStack Block Storage scheduler weighing is done. Volumes with a size less than 10ăGB are sent to lvm-1 and volumes with a size greater than or equal to 10ăGB are sent to lvm-2.

Example `cinder.conf` file configuration for customizing the goodness function:

```
[default]
scheduler_default_weighers = GoodnessWeigher
enabled_backends = lvm-1, lvm-2

[lvm-1]
volume_driver = cinder.volume.drivers.lvm.LVMVolumeDriver
volume_backend_name = sample_LVM01
goodness_function = "(volume.size < 5) ? 100 : 50"

[lvm-2]
```

(continues on next page)

(continued from previous page)

```
volume_driver = cinder.volume.drivers.lvm.LVMVolumeDriver
volume_backend_name = sample_LVM02
goodness_function = "(volume.size >= 5) ? 100 : 25"
```

The above example will determine the goodness rating of a back end based off of the requested volumes size. Default OpenStack Block Storage scheduler filtering is done. The example shows how the ternary if statement can be used in a filter or goodness function. If a requested volume is of size 10ăGB then lvm-1 is rated as 50 and lvm-2 is rated as 100. In this case lvm-2 wins. If a requested volume is of size 3ăGB then lvm-1 is rated 100 and lvm-2 is rated 25. In this case lvm-1 would win.

Example `cinder.conf` file configuration for customizing both the filter and goodness functions:

```
[default]
scheduler_default_filters = DriverFilter
scheduler_default_weighers = GoodnessWeigher
enabled_backends = lvm-1, lvm-2

[lvm-1]
volume_driver = cinder.volume.drivers.lvm.LVMVolumeDriver
volume_backend_name = sample_LVM01
filter_function = "stats.total_capacity_gb < 500"
goodness_function = "(volume.size < 25) ? 100 : 50"

[lvm-2]
volume_driver = cinder.volume.drivers.lvm.LVMVolumeDriver
volume_backend_name = sample_LVM02
filter_function = "stats.total_capacity_gb >= 500"
goodness_function = "(volume.size >= 25) ? 100 : 75"
```

The above example combines the techniques from the first two examples. The best back end is now decided based off of the total capacity of the back end and the requested volumes size.

Example `cinder.conf` file configuration for accessing driver specific properties:

```
[default]
scheduler_default_filters = DriverFilter
scheduler_default_weighers = GoodnessWeigher
enabled_backends = lvm-1,lvm-2,lvm-3

[lvm-1]
volume_group = stack-volumes-lvmdriver-1
volume_driver = cinder.volume.drivers.lvm.LVMVolumeDriver
volume_backend_name = lvmdriver-1
filter_function = "volume.size < 5"
goodness_function = "(capabilities.total_volumes < 3) ? 100 : 50"

[lvm-2]
volume_group = stack-volumes-lvmdriver-2
volume_driver = cinder.volume.drivers.lvm.LVMVolumeDriver
volume_backend_name = lvmdriver-2
filter_function = "volume.size < 5"
```

(continues on next page)

(continued from previous page)

```
goodness_function = "(capabilities.total_volumes < 8) ? 100 : 50"

[lvm-3]
volume_group = stack-volumes-lvmdriver-3
volume_driver = cinder.volume.drivers.LVMVolumeDriver
volume_backend_name = lvmdriver-3
goodness_function = "55"
```

The above is an example of how back-end specific properties can be used in the filter and goodness functions. In this example the LVM drivers `total_volumes` capability is being used to determine which host gets used during a volume request. In the above example, lvm-1 and lvm-2 will handle volume requests for all volumes with a size less than 5ăGB. Both lvm-1 and lvm-2 will have the same priority while lvm-1 contains 3 or less volumes. After that lvm-2 will have priority while it contains 8 or less volumes. The lvm-3 will collect all volumes greater or equal to 5ăGB as well as all volumes once lvm-1 and lvm-2 lose priority.

Rate-limit volume copy bandwidth

When you create a new volume from an image or an existing volume, or when you upload a volume image to the Image service, large data copy may stress disk and network bandwidth. To mitigate slow down of data access from the instances, OpenStack Block Storage supports rate-limiting of volume data copy bandwidth.

Configure volume copy bandwidth limit

To configure the volume copy bandwidth limit, set the `volume_copy_bps_limit` option in the configuration groups for each back end in the `cinder.conf` file. This option takes the integer of maximum bandwidth allowed for volume data copy in byte per second. If this option is set to `0`, the rate-limit is disabled.

While multiple volume data copy operations are running in the same back end, the specified bandwidth is divided to each copy.

Example `cinder.conf` configuration file to limit volume copy bandwidth of `lvmdriver-1` up to 100 MiB/s:

```
[lvmdriver-1]
volume_group=cinder-volumes-1
volume_driver=cinder.volume.drivers.lvm.LVMVolumeDriver
volume_backend_name=LVM
volume_copy_bps_limit=104857600
```

Note

This feature requires libcgroup to set up blkio cgroup for disk I/O bandwidth limit. The libcgroup is provided by the cgroup-tools package in Debian and Ubuntu, or by the libcgroup-tools package in Fedora, Red Hat Enterprise Linux, CentOS, openSUSE, and SUSE Linux Enterprise.

Note

Some back ends which use remote file systems such as NFS are not supported by this feature.

Oversubscription in thin provisioning

OpenStack Block Storage enables you to choose a volume back end based on virtual capacities for thin provisioning using the oversubscription ratio.

A reference implementation is provided for the default LVM driver. The illustration below uses the LVM driver as an example.

Configure oversubscription settings

To support oversubscription in thin provisioning, a flag `max_over_subscription_ratio` is introduced into `cinder.conf`. This is a float representation of the oversubscription ratio when thin provisioning is involved. Default ratio is 20.0, meaning provisioned capacity can be 20 times of the total physical capacity. A ratio of 10.5 means provisioned capacity can be 10.5 times of the total physical capacity. A ratio of 1.0 means provisioned capacity cannot exceed the total physical capacity. A ratio lower than 1.0 is ignored and the default value is used instead.

This parameter also can be set as `max_over_subscription_ratio=auto`. When using auto, Cinder will automatically calculate the `max_over_subscription_ratio` based on the provisioned capacity and the used space. This allows the creation of a larger number of volumes at the beginning of the pools life, and start to restrict the creation as the free space approaches to 0 or the reserved limit.

Note

`max_over_subscription_ratio` can be configured for each back end when multiple-storage back ends are enabled. It is provided as a reference implementation and is used by the LVM driver. However, it is not a requirement for a driver to use this option from `cinder.conf`.

`max_over_subscription_ratio` is for configuring a back end. For a driver that supports multiple pools per back end, it can report this ratio for each pool. The LVM driver does not support multiple pools.

Setting this value to auto. The values calculated by Cinder can dynamically vary according to the pools provisioned capacity and consumed space.

The existing `reserved_percentage` flag is used to prevent over provisioning. This flag represents the percentage of the back-end capacity that is reserved.

Note

There is a change on how `reserved_percentage` is used. It was measured against the free capacity in the past. Now it is measured against the total capacity.

Capabilities

Drivers can report the following capabilities for a back end or a pool:

```
thin_provisioning_support = True(or False)
thick_provisioning_support = True(or False)
provisioned_capacity_gb = PROVISIONED_CAPACITY
max_over_subscription_ratio = MAX_RATIO
```

Where `PROVISIONED_CAPACITY` is the apparent allocated space indicating how much capacity has been provisioned and `MAX_RATIO` is the maximum oversubscription ratio. For the LVM driver, it is `max_over_subscription_ratio` in `cinder.conf`.

Two capabilities are added here to allow a back end or pool to claim support for thin provisioning, or thick provisioning, or both.

The LVM driver reports `thin_provisioning_support=True` and `thick_provisioning_support=False` if the `lvm_type` flag in `cinder.conf` is `thin`. Otherwise it reports `thin_provisioning_support=False` and `thick_provisioning_support=True`.

Volume type extra specs

If volume type is provided as part of the volume creation request, it can have the following extra specs defined:

```
'capabilities:thin_provisioning_support': '<is> True' or '<is> False'
'capabilities:thick_provisioning_support': '<is> True' or '<is> False'
```

Note

`capabilities` scope key before `thin_provisioning_support` and `thick_provisioning_support` is not required. So the following works too:

```
'thin_provisioning_support': '<is> True' or '<is> False'
'thick_provisioning_support': '<is> True' or '<is> False'
```

The above extra specs are used by the scheduler to find a back end that supports thin provisioning, thick provisioning, or both to match the needs of a specific volume type.

Volume replication extra specs

OpenStack Block Storage has the ability to create volume replicas. Administrators can define a storage policy that includes replication by adjusting the cinder volume driver. Volume replication for OpenStack Block Storage helps safeguard OpenStack environments from data loss during disaster recovery.

To enable replication when creating volume types, configure the cinder volume with `capabilities:replication="<is> True"`.

Each volume created with the replication capability set to `True` generates a copy of the volume on a storage back end.

One use case for replication involves an OpenStack cloud environment installed across two data centers located nearby each other. The distance between the two data centers in this use case is the length of a city.

At each data center, a cinder host supports the Block Storage service. Both data centers include storage back ends.

Depending on the storage requirements, there can be one or two cinder hosts. The administrator accesses the `/etc/cinder/cinder.conf` configuration file and sets `capabilities:replication="<is> True"`.

If one data center experiences a service failure, administrators can redeploy the VM. The VM will run using a replicated, backed up volume on a host in the second data center.

Capacity filter

In the capacity filter, `max_over_subscription_ratio` is used when choosing a back end if `thin_provisioning_support` is True and `max_over_subscription_ratio` is greater than 1.0.

Capacity weigher

In the capacity weigher, virtual free capacity is used for ranking if `thin_provisioning_support` is True. Otherwise, real free capacity will be used as before.

Image-Volume cache

OpenStack Block Storage has an optional Image cache which can dramatically improve the performance of creating a volume from an image. The improvement depends on many factors, primarily how quickly the configured back end can clone a volume.

When a volume is first created from an image, a new cached image-volume will be created that is owned by the Block Storage Internal Tenant. Subsequent requests to create volumes from that image will clone the cached version instead of downloading the image contents and copying data to the volume.

The cache itself is configurable per back end and will contain the most recently used images.

Configure the Internal Tenant

The Image-Volume cache requires that the Internal Tenant be configured for the Block Storage services. This project will own the cached image-volumes so they can be managed like normal users including tools like volume quotas. This protects normal users from having to see the cached image-volumes, but does not make them globally hidden.

To enable the Block Storage services to have access to an Internal Tenant, set the following options in the `cinder.conf` file:

```
cinder_internal_tenant_project_id = PROJECT_ID
cinder_internal_tenant_user_id = USER_ID
```

An example `cinder.conf` configuration file:

```
cinder_internal_tenant_project_id = b7455b8974bb4064ad247c8f375eae6c
cinder_internal_tenant_user_id = f46924c112a14c80ab0a24a613d95eef
```

Note

The actual user and project that are configured for the Internal Tenant do not require any special privileges. They can be the Block Storage service project or can be any normal project and user.

Configure the Image-Volume cache

To enable the Image-Volume cache, set the following configuration option in the `cinder.conf` file:

```
image_volume_cache_enabled = True
```

Note

If you use Ceph as a back end, set the following configuration option in the `cinder.conf` file:

```
[ceph]
image_volume_cache_enabled = True
```

This can be scoped per back end definition or in the default options.

There are optional configuration settings that can limit the size of the cache. These can also be scoped per back end or in the default options in the `cinder.conf` file:

```
image_volume_cache_max_size_gb = SIZE_GB
image_volume_cache_max_count = MAX_COUNT
```

By default they will be set to 0, which means unlimited.

For example, a configuration which would limit the max size to 200 GB and 50 cache entries will be configured as:

```
image_volume_cache_max_size_gb = 200
image_volume_cache_max_count = 50
```

Note

As mentioned above, the *internal tenant* configured as the cache owner does not require any special permissions and is subject to quotas like any other user. Hence, it is possible that the quotas for the internal tenant may need to be adjusted to allow the internal tenant to hold at least `image_volume_cache_max_count` volumes not exceeding `image_volume_cache_max_size_gb` total size. Thus, although the default value for these image volume cache settings is `0` (unlimited), in practice, these will be limited by the quotas that apply to the internal tenant.

See *Manage Block Storage service quotas* for more information.

Notifications

Cache actions will trigger Telemetry messages. There are several that will be sent.

- `image_volume_cache.miss` - A volume is being created from an image which was not found in the cache. Typically this will mean a new cache entry would be created for it.
- `image_volume_cache.hit` - A volume is being created from an image which was found in the cache and the fast path can be taken.
- `image_volume_cache.evict` - A cached image-volume has been deleted from the cache.

Managing cached Image-Volumes

In normal usage there should be no need for manual intervention with the cache. The entries and their backing Image-Volumes are managed automatically.

If needed, you can delete these volumes manually to clear the cache. By using the standard volume deletion APIs, the Block Storage service will clean up correctly.

Volume-backed image

OpenStack Block Storage can quickly create a volume from an image that refers to a volume storing image data (Image-Volume). Compared to the other stores such as file and swift, creating a volume from a Volume-backed image performs better when the block storage driver supports efficient volume cloning.

If the image is set to public in the Image service, the volume data can be shared among projects.

Configure the Volume-backed image

Volume-backed image feature requires locations information from the cinder store of the Image service. To enable the Image service to use the cinder store, add `cinder` to the `stores` option in the `glance_store` section of the `glance-api.conf` file:

```
stores = file, http, swift, cinder
```

To expose locations information, set the following options in the `DEFAULT` section of the `glance-api.conf` file:

```
show_multiple_locations = True
```

To enable the Block Storage services to create a new volume by cloning Image- Volume, set the following options in the `DEFAULT` section of the `cinder.conf` file. For example:

```
allowed_direct_url_schemes = cinder
```

To enable the **`openstack image create --volume <volume>`** command to create an image that refers an `Image-Volume`, set the following options in each back-end section of the `cinder.conf` file:

```
image_upload_use_cinder_backend = True
```

By default, the **`openstack image create --volume <volume>`** command creates the Image-Volume in the current project. To store the Image-Volume into the internal project, set the following options in each back-end section of the `cinder.conf` file:

```
image_upload_use_internal_tenant = True
```

To make the Image-Volume in the internal project accessible from the Image service, set the following options in the `glance_store` section of the `glance-api.conf` file:

- `cinder_store_auth_address`
- `cinder_store_user_name`
- `cinder_store_password`
- `cinder_store_project_name`

Creating a Volume-backed image

To register an existing volume as a new Volume-backed image, use the following commands:

```
$ openstack image create --disk-format raw --container-format bare IMAGE_NAME

$ glance location-add <image-uuid> --url cinder://<volume-uuid>
```

If the `image_upload_use_cinder_backend` option is enabled, the following command creates a new Image-Volume by cloning the specified volume and then registers its location to a new image. The disk format and the container format must be raw and bare (default). Otherwise, the image is uploaded to the default store of the Image service.

```
$ openstack image create --volume SOURCE_VOLUME IMAGE_NAME
```

Get capabilities

When an administrator configures `volume type` and `extra specs` of storage on the back end, the administrator has to read the right documentation that corresponds to the version of the storage back end. Deep knowledge of storage is also required.

OpenStack Block Storage enables administrators to configure `volume type` and `extra specs` without specific knowledge of the storage back end.

Note

- `Volume Type`: A group of volume policies.
- `Extra Specs`: The definition of a volume type. This is a group of policies. For example, provision type, QOS that will be used to define a volume at creation time.
- `Capabilities`: What the current deployed back end in Cinder is able to do. These correspond to extra specs.

Usage of cinder client

When an administrator wants to define new volume types for their OpenStack cloud, the administrator would fetch a list of `capabilities` for a particular back end using the cinder client.

First, get a list of the services:

```
$ openstack volume service list
+------------------+-------------------+------+---------+-------+------------
↪---------------+
| Binary           | Host              | Zone | Status  | State | Updated At ␣
↪               |
+------------------+-------------------+------+---------+-------+------------
↪---------------+
| cinder-scheduler | controller        | nova | enabled | up    | 2016-10-
↪24T13:53:35.000000 |
| cinder-volume    | block1@ABC-driver | nova | enabled | up    | 2016-10-
↪24T13:53:35.000000 |
| cinder-backup    | controller        | nova | enabled | up    | 2016-10-
↪24T13:53:35.000000 |
+------------------+-------------------+------+---------+-------+------------
↪---------------+
```

With one of the listed hosts, pass that to `get-capabilities`, then the administrator can obtain volume stats and also back end `capabilities` as listed below.

```
$ cinder get-capabilities block1@ABC-driver
+---------------------+----------------------------------------------+
|     Volume stats    |                    Value                     |
+---------------------+----------------------------------------------+
|     description     |                     None                     |
|     display_name    |   Capabilities of Cinder Vendor ABC driver   |
|    driver_version   |                    2.0.0                     |
|      namespace      | OS::Storage::Capabilities::block1@ABC-driver |
|      pool_name      |                     None                     |
| replication_targets |                      []                      |
|   storage_protocol  |                    iSCSI                     |
|     vendor_name     |                  Vendor ABC                  |
|      visibility     |                     pool                     |
| volume_backend_name |                  ABC-driver                  |
+---------------------+----------------------------------------------+
+----------------------+-----------------------------------------------------+
|  Backend properties  |                        Value                        |
+----------------------+-----------------------------------------------------+
|      compression     | {u'type':u'boolean', u'title':u'Compression',  ...} |
| ABC:compression_type | {u'enum':u'['lossy', 'lossless', 'special']',  ...} |
|         qos          | {u'type':u'boolean', u'title':u'QoS',          ...} |
|     replication      | {u'type':u'boolean', u'title':u'Replication',  ...} |
|  thin_provisioning   | {u'type':u'boolean', u'title':u'Thin Provisioning'} |
|      ABC:minIOPS     | {u'type':u'integer', u'title':u'Minimum IOPS QoS',} |
|      ABC:maxIOPS     | {u'type':u'integer', u'title':u'Maximum IOPS QoS',} |
|     ABC:burstIOPS    | {u'type':u'integer', u'title':u'Burst IOPS QoS',..} |
+----------------------+-----------------------------------------------------+
```

Disable a service

When an administrator wants to disable a service, identify the Binary and the Host of the service. Use the :command:' openstack volume service set' command combined with the Binary and Host to disable the service:

1. Determine the binary and host of the service you want to remove initially.

```
$ openstack volume service list
+------------------+---------------------+------+---------+-------+------
↪---------------------+
| Binary           | Host                | Zone | Status  | State |␣
↪Updated At                 |
+------------------+---------------------+------+---------+-------+------
↪---------------------+
| cinder-scheduler | devstack            | nova | enabled | up    | 2016-
↪10-24T13:53:35.000000 |
| cinder-volume    | devstack@lvmdriver-1 | nova | enabled | up    | 2016-
↪10-24T13:53:35.000000 |
| cinder-backup    | devstack            | nova | enabled | up    | 2016-
↪10-24T13:53:35.000000 |
+------------------+---------------------+------+---------+-------+------
↪---------------------+
```

2. Disable the service using the Binary and Host name, placing the Host before the Binary name.

```
$ openstack volume service set --disable HOST_NAME BINARY_NAME
```

3. Remove the service from the database.

```
$ cinder-manage service remove BINARY_NAME HOST_NAME
```

Usage of REST API

New endpoint to `get capabilities` list for specific storage back end is also available. For more details, refer to the Block Storage API reference.

API request:

```
GET /v3/{tenant_id}/capabilities/{hostname}
```

Example of return value:

```
{
  "namespace": "OS::Storage::Capabilities::block1@ABC-driver",
  "volume_backend_name": "ABC-driver",
  "pool_name": "pool",
  "driver_version": "2.0.0",
  "storage_protocol": "iSCSI",
  "display_name": "Capabilities of Cinder Vendor ABC driver",
  "description": "None",
  "visibility": "public",
  "properties": {
```

(continues on next page)

(continued from previous page)


```
    "thin_provisioning": {
      "title": "Thin Provisioning",
      "description": "Sets thin provisioning.",
      "type": "boolean"
    },
    "compression": {
      "title": "Compression",
      "description": "Enables compression.",
      "type": "boolean"
    },
    "ABC:compression_type": {
      "title": "Compression type",
      "description": "Specifies compression type.",
      "type": "string",
      "enum": [
        "lossy", "lossless", "special"
      ]
    },
    "replication": {
      "title": "Replication",
      "description": "Enables replication.",
      "type": "boolean"
    },
    "qos": {
      "title": "QoS",
      "description": "Enables QoS.",
      "type": "boolean"
    },
    "ABC:minIOPS": {
      "title": "Minimum IOPS QoS",
      "description": "Sets minimum IOPS if QoS is enabled.",
      "type": "integer"
    },
    "ABC:maxIOPS": {
      "title": "Maximum IOPS QoS",
      "description": "Sets maximum IOPS if QoS is enabled.",
      "type": "integer"
    },
    "ABC:burstIOPS": {
      "title": "Burst IOPS QoS",
      "description": "Sets burst IOPS if QoS is enabled.",
      "type": "integer"
    },
  }
}
```

Usage of volume type access extension

Some volume types should be restricted only. For example, test volume types where you are testing a new technology or ultra high performance volumes (for special cases) where you do not want most users to be able to select these volumes. An administrator/operator can then define private volume types using cinder client. Volume type access extension adds the ability to manage volume type access. Volume types are public by default. Private volume types can be created by setting the `--private` parameter at creation time. Access to a private volume type can be controlled by adding or removing a project from it. Private volume types without projects are only visible by users with the admin role/context.

Create a public volume type by setting `--public` parameter:

```
$ openstack volume type create vol_Type1 --description test1 --public
+-------------+--------------------------------------+
| Field       | Value                                |
+-------------+--------------------------------------+
| description | test1                                |
| id          | b7dbed9e-de78-49f8-a840-651ae7308592 |
| is_public   | True                                 |
| name        | vol_Type1                            |
+-------------+--------------------------------------+
```

Create a private volume type by setting `--private` parameter:

```
$ openstack volume type create vol_Type2 --description test2 --private
+-------------+--------------------------------------+
| Field       | Value                                |
+-------------+--------------------------------------+
| description | test2                                |
| id          | 154baa73-d2c4-462f-8258-a2df251b0d39 |
| is_public   | False                                |
| name        | vol_Type2                            |
+-------------+--------------------------------------+
```

Get a list of the volume types:

```
$ openstack volume type list
+--------------------------------------+-------------+
| ID                                   | Name        |
+--------------------------------------+-------------+
| 0a948c84-bad5-4fba-88a2-c062006e4f6b | vol_Type1   |
| 87e5be6f-9491-4ea5-9906-9ac56494bb91 | lvmdriver-1 |
| fd508846-213f-4a07-aaf2-40518fb9a23f | vol_Type2   |
+--------------------------------------+-------------+
```

Get a list of the projects:

```
$ openstack project list
+----------------------------------+--------------------+
| ID                               | Name               |
+----------------------------------+--------------------+
| 4105ead90a854100ab6b121266707f2b | alt_demo           |
```

(continues on next page)

(continued from previous page)

```
| 4a22a545cedd4fcfa9836eb75e558277 | admin             |
| 71f9cdb1a3ab4b8e8d07d347a2e146bb | service           |
| c4860af62ffe465e99ed1bc08ef6082e | demo              |
| e4b648ba5108415cb9e75bff65fa8068 | invisible_to_admin |
+----------------------------------+--------------------+
```

Add volume type access for the given demo project, using its project-id:

```
$ openstack volume type set --project c4860af62ffe465e99ed1bc08ef6082e \
  vol_Type2
```

List the access information about the given volume type:

```
$ openstack volume type show vol_Type2
+--------------------+--------------------------------------+
| Field              | Value                                |
+--------------------+--------------------------------------+
| access_project_ids | c4860af62ffe465e99ed1bc08ef6082e     |
| description        |                                      |
| id                 | fd508846-213f-4a07-aaf2-40518fb9a23f |
| is_public          | False                                |
| name               | vol_Type2                            |
| properties         |                                      |
| qos_specs_id       | None                                 |
+--------------------+--------------------------------------+
```

Remove volume type access for the given project:

```
$ openstack volume type unset --project c4860af62ffe465e99ed1bc08ef6082e \
  vol_Type2
$ openstack volume type show vol_Type2
+--------------------+--------------------------------------+
| Field              | Value                                |
+--------------------+--------------------------------------+
| access_project_ids |                                      |
| description        |                                      |
| id                 | fd508846-213f-4a07-aaf2-40518fb9a23f |
| is_public          | False                                |
| name               | vol_Type2                            |
| properties         |                                      |
| qos_specs_id       | None                                 |
+--------------------+--------------------------------------+
```

User visible extra specs

Starting in Xena, certain volume type `extra specs` (i.e. properties) are considered user visible, meaning their visibility is not restricted to only cloud administrators. This feature provides regular users with more information about the volume types available to them, and lets them make more informed decisions on which volume type to choose when creating volumes.

The following `extra spec` keys are treated as user visible:

- RESKEY:availability_zones
- multiattach
- replication_enabled

Note

- The set of user visible extra specs is a fixed list that is not configurable.
- The feature is entirely policy based, and does not require a new microversion.

Behavior using openstack client

Consider the following volume type, as viewed from an administrators perspective. In this example, multiattach is a user visible extra spec and volume_backend_name is not.

```
# Administrator behavior
[admin@host]$ openstack volume type show vol_type
+--------------------+-------------------------------------------------------+
| Field              | Value                                                 |
+--------------------+-------------------------------------------------------+
| access_project_ids | None                                                  |
| description        | None                                                  |
| id                 | d03a0f33-e695-4f5c-b712-7d92abbf72be                  |
| is_public          | True                                                  |
| name               | vol_type                                              |
| properties         | multiattach='<is> True', volume_backend_name='secret' |
| qos_specs_id       | None                                                  |
+--------------------+-------------------------------------------------------+
```

Here is the output when a regular user executes the same command. Notice only the user visible multiattach property is listed.

```
# Regular user behavior
[user@host]$ openstack volume type show vol_type
+--------------------+--------------------------------------+
| Field              | Value                                |
+--------------------+--------------------------------------+
| access_project_ids | None                                 |
| description        | None                                 |
| id                 | d03a0f33-e695-4f5c-b712-7d92abbf72be |
| is_public          | True                                 |
| name               | vol_type                             |
| properties         | multiattach='<is> True'              |
+--------------------+--------------------------------------+
```

The behavior for listing volume types is similar. Administrators will see all extra specs but regular users will see only user visible extra specs.

```
# Administrator behavior
[admin@host]$ openstack volume type list --long
```

(continues on next page)

(continued from previous page)

```
+--------------------------------------+-------------+-----------+------------
↪---------+-----------------------------------------------------+
| ID                                   | Name        | Is Public |␣
↪Description         | Properties                                           ␣
↪|
+--------------------------------------+-------------+-----------+------------
↪---------+-----------------------------------------------------+
| d03a0f33-e695-4f5c-b712-7d92abbf72be | vol_type    | True      | None       ␣
↪         | multiattach='<is> True', volume_backend_name='secret' |
| 80f38273-f4b9-4862-a4e6-87692eb66a96 | __DEFAULT__ | True      | Default␣
↪Volume Type |                                                     |
+--------------------------------------+-------------+-----------+------------
↪---------+-----------------------------------------------------+

# Regular user behavior
[user@host]$ openstack volume type list --long
+--------------------------------------+-------------+-----------+------------
↪---------+-----------------------+
| ID                                   | Name        | Is Public |␣
↪Description         | Properties            |
+--------------------------------------+-------------+-----------+------------
↪---------+-----------------------+
| d03a0f33-e695-4f5c-b712-7d92abbf72be | vol_type    | True      | None       ␣
↪         | multiattach='<is> True' |
| 80f38273-f4b9-4862-a4e6-87692eb66a96 | __DEFAULT__ | True      | Default␣
↪Volume Type |                       |
+--------------------------------------+-------------+-----------+------------
↪---------+-----------------------+
```

Regular users may view these properties, but they may not modify them. Attempts to modify a user visible property by a non-administrator will fail.

```
[user@host]$ openstack volume type set --property multiattach='<is> False'␣
↪vol_type
Failed to set volume type property: Policy doesn't allow
volume_extension:types_extra_specs:create to be performed. (HTTP 403)
```

Filtering with extra specs

API microversion 3.52 adds support for using `extra specs` to filter the list of volume types. Regular users are able to use that feature to filter for user visible `extra specs`. If a regular user attempts to filter on a non-user visible `extra spec` then an empty list is returned.

```
# Administrator behavior
[admin@host]$ cinder --os-volume-api-version 3.52 type-list \
> --filters extra_specs={"multiattach":"<is> True"}
+--------------------------------------+----------+-------------+-----------+
| ID                                   | Name     | Description | Is_Public |
+--------------------------------------+----------+-------------+-----------+
```

(continues on next page)

(continued from previous page)

```
| d03a0f33-e695-4f5c-b712-7d92abbf72be | vol_type | -           | True      |
+--------------------------------------+----------+-------------+-----------+

[admin@host]$ cinder --os-volume-api-version 3.52 type-list \
> --filters extra_specs={"volume_backend_name":"secret"}
+--------------------------------------+----------+-------------+-----------+
| ID                                   | Name     | Description | Is_Public |
+--------------------------------------+----------+-------------+-----------+
| d03a0f33-e695-4f5c-b712-7d92abbf72be | vol_type | -           | True      |
+--------------------------------------+----------+-------------+-----------+

# Regular user behavior
[user@host]$ cinder --os-volume-api-version 3.52 type-list \
> --filters extra_specs={"multiattach":"<is> True"}
+--------------------------------------+----------+-------------+-----------+
| ID                                   | Name     | Description | Is_Public |
+--------------------------------------+----------+-------------+-----------+
| d03a0f33-e695-4f5c-b712-7d92abbf72be | vol_type | -           | True      |
+--------------------------------------+----------+-------------+-----------+

[user@host]$ cinder --os-volume-api-version 3.52 type-list \
> --filters extra_specs={"volume_backend_name":"secret"}
+----+------+-------------+-----------+
| ID | Name | Description | Is_Public |
+----+------+-------------+-----------+
+----+------+-------------+-----------+
```

Security considerations

Cloud administrators who do not wish to expose any `extra specs` to regular users may restore the previous behavior by setting the following policies to their pre-Xena default values.

```
"volume_extension:access_types_extra_specs": "rule:admin_api"
"volume_extension:types_extra_specs:index": "rule:admin_api"
"volume_extension:types_extra_specs:show": "rule:admin_api"
```

To restrict regular users from using `extra specs` to filter the list of volume types, modify /etc/cinder/resource_filters.json to restore the *volume_type* entry to its pre-Xena default value.

```
"volume_type": ["is_public"]
```

Generic volume groups

Generic volume group support is available in OpenStack Block Storage (cinder) since the Newton release. The support is added for creating group types and group specs, creating groups of volumes, and creating snapshots of groups. The group operations can be performed using the Block Storage command line.

A group type is a type for a group just like a volume type for a volume. A group type can also have associated group specs similar to extra specs for a volume type.

In cinder, there is a group construct called *consistency group*. Consistency groups only support consistent group snapshots and only a small number of drivers can support it. The following is a list of drivers that support consistency groups and the release when the support was added:

- Juno: EMC VNX
- Kilo: EMC VMAX, IBM (GPFS, Storwize, SVC, and XIV), ProphetStor, Pure
- Liberty: Dell Storage Center, EMC XtremIO, HPE 3Par and LeftHand
- Mitaka: EMC ScaleIO, NetApp Data ONTAP, SolidFire
- Newton: CoprHD, FalconStor, Huawei

Consistency group cannot be extended easily to serve other purposes. A tenant may want to put volumes used in the same application together in a group so that it is easier to manage them together, and this group of volumes may or may not support consistent group snapshot. Generic volume group is introduced to solve this problem.

There is a plan to migrate existing consistency group operations to use generic volume group operations in future releases. More information can be found in Cinder specs.

Note

Only Block Storage V3 API supports groups. You can specify `--os-volume-api-version 3.x` when using the *cinder* command line for group operations where *3.x* contains a microversion value for that command. The generic volume group feature was completed in several patches. As a result, the minimum required microversion is different for group types, groups, and group snapshots APIs.

The following group type operations are supported:

- Create a group type.
- Delete a group type.
- Set group spec for a group type.
- Unset group spec for a group type.
- List group types.
- Show a group type details.
- Update a group.
- List group types and group specs.

The following group and group snapshot operations are supported:

- Create a group, given group type and volume types.

  Note

  A group must have one group type. A group can support more than one volume type. The scheduler is responsible for finding a back end that can support the given group type and volume types.

  A group can only contain volumes hosted by the same back end.

  A group is empty upon its creation. Volumes need to be created and added to it later.

- Show a group.
- List groups.
- Delete a group.
- Modify a group.
- Create a volume and add it to a group.
- Create a snapshot for a group.
- Show a group snapshot.
- List group snapshots.
- Delete a group snapshot.
- Create a group from a group snapshot.
- Create a group from a source group.

The following operations are not allowed if a volume is in a group:

- Volume migration.
- Volume retype.
- Volume deletion.

> **Note**
>
> A group has to be deleted as a whole with all the volumes.

The following operations are not allowed if a volume snapshot is in a group snapshot:

- Volume snapshot deletion.

> **Note**
>
> A group snapshot has to be deleted as a whole with all the volume snapshots.

The details of group type operations are shown in the following. The minimum microversion to support group type and group specs is 3.11:

Create a group type:

```
cinder --os-volume-api-version 3.11 group-type-create
[--description DESCRIPTION]
[--is-public IS_PUBLIC]
NAME
```

> **Note**
>
> The parameter `NAME` is required. The `--is-public IS_PUBLIC` determines whether the group type is accessible to the public. It is `True` by default. By default, the policy on privileges for creating a group type is admin-only.

Show a group type:

```
cinder --os-volume-api-version 3.11 group-type-show
GROUP_TYPE
```

Note

The parameter `GROUP_TYPE` is the name or UUID of a group type.

List group types:

```
cinder --os-volume-api-version 3.11 group-type-list
```

Note

Only admin can see private group types.

Update a group type:

```
cinder --os-volume-api-version 3.11 group-type-update
[--name NAME]
[--description DESCRIPTION]
[--is-public IS_PUBLIC]
GROUP_TYPE_ID
```

Note

The parameter `GROUP_TYPE_ID` is the UUID of a group type. By default, the policy on privileges for updating a group type is admin-only.

Delete group type or types:

```
cinder --os-volume-api-version 3.11 group-type-delete
GROUP_TYPE [GROUP_TYPE ...]
```

Note

The parameter `GROUP_TYPE` is name or UUID of the group type or group types to be deleted. By default, the policy on privileges for deleting a group type is admin-only.

Set or unset group spec for a group type:

```
cinder --os-volume-api-version 3.11 group-type-key
GROUP_TYPE ACTION KEY=VALUE [KEY=VALUE ...]
```

Note

The parameter GROUP_TYPE is the name or UUID of a group type. Valid values for the parameter ACTION are set or unset. KEY=VALUE is the group specs key and value pair to set or unset. For unset, specify only the key. By default, the policy on privileges for setting or unsetting group specs key is admin-only.

List group types and group specs:

```
cinder --os-volume-api-version 3.11 group-specs-list
```

Note

By default, the policy on privileges for seeing group specs is admin-only.

The details of group operations are shown in the following. The minimum microversion to support groups operations is 3.13.

Create a group:

```
cinder --os-volume-api-version 3.13 group-create
[--name NAME]
[--description DESCRIPTION]
[--availability-zone AVAILABILITY_ZONE]
GROUP_TYPE VOLUME_TYPES
```

Note

The parameters GROUP_TYPE and VOLUME_TYPES are required. GROUP_TYPE is the name or UUID of a group type. VOLUME_TYPES can be a list of names or UUIDs of volume types separated by commas without spaces in between. For example, volumetype1,volumetype2,volumetype3..

Show a group:

```
cinder --os-volume-api-version 3.13 group-show
GROUP
```

Note

The parameter GROUP is the name or UUID of a group.

List groups:

```
cinder --os-volume-api-version 3.13 group-list
[--all-tenants [<0|1>]]
```

Note

`--all-tenants` specifies whether to list groups for all tenants. Only admin can use this option.

Create a volume and add it to a group:

```
cinder --os-volume-api-version 3.13 create
--volume-type VOLUME_TYPE
--group-id GROUP_ID SIZE
```

Note

When creating a volume and adding it to a group, the parameters `VOLUME_TYPE` and `GROUP_ID` must be provided. This is because a group can support more than one volume type.

Delete a group:

```
cinder --os-volume-api-version 3.13 group-delete
[--delete-volumes]
GROUP [GROUP ...]
```

Note

`--delete-volumes` allows or disallows groups to be deleted if they are not empty. If the group is empty, it can be deleted without `--delete-volumes`. If the group is not empty, the flag is required for it to be deleted. When the flag is specified, the group and all volumes in the group will be deleted.

Modify a group:

```
cinder --os-volume-api-version 3.13 group-update
[--name NAME]
[--description DESCRIPTION]
[--add-volumes UUID1,UUID2,......]
[--remove-volumes UUID3,UUID4,......]
GROUP
```

Note

The parameter `UUID1,UUID2,......` is the UUID of one or more volumes to be added to the group, separated by commas. Similarly the parameter `UUID3,UUID4,......` is the UUID of one or more volumes to be removed from the group, separated by commas.

The details of group snapshots operations are shown in the following. The minimum microversion to support group snapshots operations is 3.14.

Create a snapshot for a group:

```
cinder --os-volume-api-version 3.14 group-snapshot-create
[--name NAME]
[--description DESCRIPTION]
GROUP
```

Note

The parameter `GROUP` is the name or UUID of a group.

Show a group snapshot:

```
cinder --os-volume-api-version 3.14 group-snapshot-show
GROUP_SNAPSHOT
```

Note

The parameter `GROUP_SNAPSHOT` is the name or UUID of a group snapshot.

List group snapshots:

```
cinder --os-volume-api-version 3.14 group-snapshot-list
[--all-tenants [<0|1>]]
[--status STATUS]
[--group-id GROUP_ID]
```

Note

`--all-tenants` specifies whether to list group snapshots for all tenants. Only admin can use this option. `--status STATUS` filters results by a status. `--group-id GROUP_ID` filters results by a group id.

Delete group snapshot:

```
cinder --os-volume-api-version 3.14 group-snapshot-delete
GROUP_SNAPSHOT [GROUP_SNAPSHOT ...]
```

Note

The parameter `GROUP_SNAPSHOT` specifies the name or UUID of one or more group snapshots to be deleted.

Create a group from a group snapshot or a source group:

```
$ cinder --os-volume-api-version 3.14 group-create-from-src
[--group-snapshot GROUP_SNAPSHOT]
[--source-group SOURCE_GROUP]
```

(continues on next page)

(continued from previous page)

```
[--name NAME]
[--description DESCRIPTION]
```

Note

The parameter GROUP_SNAPSHOT is a name or UUID of a group snapshot. The parameter SOURCE_GROUP is a name or UUID of a source group. Either GROUP_SNAPSHOT or SOURCE_GROUP must be specified, but not both.

Note

To enable the use of encrypted volumes, see the setup instructions in *Create an encrypted volume type*.

Troubleshoot your installation

This section provides useful tips to help you troubleshoot your Block Storage installation.

Troubleshoot the Block Storage configuration

Most Block Storage errors are caused by incorrect volume configurations that result in volume creation failures. To resolve these failures, review these logs:

- `cinder-api` log (`/var/log/cinder/api.log`)
- `cinder-volume` log (`/var/log/cinder/volume.log`)

The `cinder-api` log is useful for determining if you have endpoint or connectivity issues. If you send a request to create a volume and it fails, review the `cinder-api` log to determine whether the request made it to the Block Storage service. If the request is logged and you see no errors or tracebacks, check the `cinder-volume` log for errors or tracebacks.

Note

Create commands are listed in the `cinder-api` log.

These entries in the `cinder.conf` file can be used to assist in troubleshooting your Block Storage configuration.

```
# Print debugging output (set logging level to DEBUG instead
# of default WARNING level). (boolean value)
# debug=false

# Log output to standard error (boolean value)
# use_stderr=true

# Default file mode used when creating log files (string
# value)
```

(continues on next page)

(continued from previous page)

```
# logfile_mode=0644

# format string to use for log messages with context (string
# value)
# logging_context_format_string=%(asctime)s.%(msecs)03d %(levelname)s
# %(name)s [%(request_id)s %(user)s %(tenant)s] %(instance)s%(message)s

# format string to use for log mes #logging_default_format_string=%(asctime)s.
# %(msecs)03d %(process)d %(levelname)s %(name)s [-] %(instance)s%(message)s

# data to append to log format when level is DEBUG (string
# value)
# logging_debug_format_suffix=%(funcName)s %(pathname)s:%(lineno)d

# prefix each line of exception output with this format
# (string value)
# logging_exception_prefix=%(asctime)s.%(msecs)03d %(process)d TRACE %(name)s
# %(instance)s

# list of logger=LEVEL pairs (list value)
# default_log_levels=amqplib=WARN,sqlalchemy=WARN,boto=WARN,suds=INFO,
# keystone=INFO,eventlet.wsgi.server=WARNsages without context
# (string value)

# If an instance is passed with the log message, format it
# like this (string value)
# instance_format="[instance: %(uuid)s]"

# If an instance UUID is passed with the log message, format
# it like this (string value)
#instance_uuid_format="[instance: %(uuid)s] "

# Format string for %%(asctime)s in log records. Default:
# %(default)s (string value)
# log_date_format=%Y-%m-%d %H:%M:%S

# (Optional) Name of log file to output to. If not set,
# logging will go to stdout. (string value)
# log_file=<None>

# (Optional) The directory to keep log files in (will be
# prepended to --log-file) (string value)
# log_dir=<None>
# instance_uuid_format="[instance: %(uuid)s]"

# If this option is specified, the logging configuration file
# specified is used and overrides any other logging options
# specified. Please see the Python logging module
# documentation for details on logging configuration files.
```

(continues on next page)

(continued from previous page)

```
# (string value)
# Use syslog for logging. (boolean value)
# use_syslog=false

# syslog facility to receive log lines (string value)
# syslog_log_facility=LOG_USER
# log_config=<None>
```

These common issues might occur during configuration, and the following potential solutions describe how to address the issues.

Issues with `state_path` and `volumes_dir` settings

Problem

The OpenStack Block Storage uses `tgtd` as the default iSCSI helper and implements persistent targets. This means that in the case of a `tgt` restart, or even a node reboot, your existing volumes on that node will be restored automatically with their original *IQN*.

By default, Block Storage uses a `state_path` variable, which if installing with Yum or APT should be set to `/var/lib/cinder/`. The next part is the `volumes_dir` variable, by default this appends a `volumes` directory to the `state_path`. The result is a file-tree: `/var/lib/cinder/volumes/`.

Solution

In order to ensure nodes are restored to their original IQN, the iSCSI target information needs to be stored in a file on creation that can be queried in case of restart of the `tgt daemon`. While the installer should handle all this, it can go wrong.

If you have trouble creating volumes and this directory does not exist you should see an error message in the `cinder-volume` log indicating that the `volumes_dir` does not exist, and it should provide information about which path it was looking for.

The persistent tgt include file

Problem

The Block Storage service may have issues locating the persistent `tgt include` file. Along with the `volumes_dir` option, the iSCSI target driver also needs to be configured to look in the correct place for the persistent `tgt include `` file. This is an entry in the ``/etc/tgt/conf.d` file that should have been set during the OpenStack installation.

Solution

If issues occur, verify that you have a `/etc/tgt/conf.d/cinder.conf` file. If the file is not present, create it with:

```
# echo 'include /var/lib/cinder/volumes/ *' >> /etc/tgt/conf.d/cinder.conf
```

Failed to create iscsi target error in the cinder-volume.log file

Problem

```
2013-03-12 01:35:43 1248 TRACE cinder.openstack.common.rpc.amqp \
ISCSITargetCreateFailed: \
Failed to create iscsi target for volume \
volume-137641b2-af72-4a2f-b243-65fdccd38780.
```

You might see this error in `cinder-volume.log` after trying to create a volume that is 1 GB.

Solution

To fix this issue, change the content of the `/etc/tgt/targets.conf` file from `include /etc/tgt/conf.d/*.conf` to `include /etc/tgt/conf.d/cinder_tgt.conf`, as follows:

```
include /etc/tgt/conf.d/cinder_tgt.conf
include /etc/tgt/conf.d/cinder.conf
default-driver iscsi
```

Restart `tgt` and `cinder-*` services, so they pick up the new configuration.

Multipath call failed exit

Problem

Multipath call failed exit. This warning occurs in the Compute log if you do not have the optional `multipath-tools` package installed on the compute node. This is an optional package and the volume attachment does work without the multipath tools installed. If the `multipath-tools` package is installed on the compute node, it is used to perform the volume attachment. The IDs in your message are unique to your system.

```
WARNING nova.storage.linuxscsi [req-cac861e3-8b29-4143-8f1b-705d0084e571
    admin admin|req-cac861e3-8b29-4143-8f1b-705d0084e571 admin admin]
    Multipath call failed exit (96)
```

Solution

Run the following command on the compute node to install the `multipath-tools` packages.

```
# apt-get install multipath-tools
```

HTTP bad request in cinder volume log

Problem

These errors appear in the `cinder-volume.log` file:

```
2013-05-03 15:16:33 INFO [cinder.volume.manager] Updating volume status
2013-05-03 15:16:33 DEBUG [hp3parclient.http]
REQ: curl -i https://10.10.22.241:8080/api/v1/cpgs -X GET -H "X-Hp3Par-Wsapi-
```

(continues on next page)

(continued from previous page)

```
↪Sessionkey: 48dc-b69ed2e5
f259c58e26df9a4c85df110c-8d1e8451" -H "Accept: application/json" -H "User-
↪Agent: python-3parclient"

2013-05-03 15:16:33 DEBUG [hp3parclient.http] RESP:{'content-length': 311,
↪'content-type': 'text/plain',
'status': '400'}

2013-05-03 15:16:33 DEBUG [hp3parclient.http] RESP BODY:Second simultaneous␣
↪read on fileno 13 detected.
Unless you really know what you're doing, make sure that only one greenthread␣
↪can read any particular socket.
Consider using a pools.Pool. If you do know what you're doing and want to␣
↪disable this error,
call eventlet.debug.hub_multiple_reader_prevention(False)

2013-05-03 15:16:33 ERROR [cinder.manager] Error during VolumeManager._report_
↪driver_status: Bad request (HTTP 400)
Traceback (most recent call last):
File "/usr/lib/python2.7/dist-packages/cinder/manager.py", line 167, in␣
↪periodic_tasks task(self, context)
File "/usr/lib/python2.7/dist-packages/cinder/volume/manager.py", line 690,␣
↪in _report_driver_status volume_stats =
self.driver.get_volume_stats(refresh=True)
File "/usr/lib/python2.7/dist-packages/cinder/volume/drivers/san/hp/hp_3par_
↪fc.py", line 77, in get_volume_stats stats =
self.common.get_volume_stats(refresh, self.client)
File "/usr/lib/python2.7/dist-packages/cinder/volume/drivers/san/hp/hp_3par_
↪common.py", line 421, in get_volume_stats cpg =
client.getCPG(self.config.hp3par_cpg)
File "/usr/lib/python2.7/dist-packages/hp3parclient/client.py", line 231, in␣
↪getCPG cpgs = self.getCPGs()
File "/usr/lib/python2.7/dist-packages/hp3parclient/client.py", line 217, in␣
↪getCPGs response, body = self.http.get('/cpgs')
File "/usr/lib/python2.7/dist-packages/hp3parclient/http.py", line 255, in␣
↪get return self._cs_request(url, 'GET', **kwargs)
File "/usr/lib/python2.7/dist-packages/hp3parclient/http.py", line 224, in _
↪cs_request **kwargs)
File "/usr/lib/python2.7/dist-packages/hp3parclient/http.py", line 198, in _
↪time_request resp, body = self.request(url, method, **kwargs)
File "/usr/lib/python2.7/dist-packages/hp3parclient/http.py", line 192, in␣
↪request raise exceptions.from_response(resp, body)
HTTPBadRequest: Bad request (HTTP 400)
```

Solution

You need to update your copy of the `hp_3par_fc.py` driver which contains the synchronization code.

Duplicate 3PAR host

Problem

This error may be caused by a volume being exported outside of OpenStack using a host name different from the system name that OpenStack expects. This error could be displayed with the *IQN* if the host was exported using iSCSI:

```
Duplicate3PARHost: 3PAR Host already exists: Host wwn 50014380242B9750 \
already used by host cld4b5ubuntuW(id = 68. The hostname must be called\
'cld4b5ubuntu'.
```

Solution

Change the 3PAR host name to match the one that OpenStack expects. The 3PAR host constructed by the driver uses just the local host name, not the fully qualified domain name (FQDN) of the compute host. For example, if the FQDN was *myhost.example.com*, just *myhost* would be used as the 3PAR host name. IP addresses are not allowed as host names on the 3PAR storage server.

Failed to attach volume after detaching

Problem

Failed to attach a volume after detaching the same volume.

Solution

You must change the device name on the **nova-attach** command. The VM might not clean up after a **nova-detach** command runs. This example shows how the **nova-attach** command fails when you use the `vdb`, `vdc`, or `vdd` device names:

```
# ls -al /dev/disk/by-path/
total 0
drwxr-xr-x 2 root root 200 2012-08-29 17:33 .
drwxr-xr-x 5 root root 100 2012-08-29 17:33 ..
lrwxrwxrwx 1 root root 9 2012-08-29 17:33 pci-0000:00:04.0-virtio-pci-virtio0␣
↪-> ../../vda
lrwxrwxrwx 1 root root 10 2012-08-29 17:33 pci-0000:00:04.0-virtio-pci-
↪virtio0-part1 -> ../../vda1
lrwxrwxrwx 1 root root 10 2012-08-29 17:33 pci-0000:00:04.0-virtio-pci-
↪virtio0-part2 -> ../../vda2
lrwxrwxrwx 1 root root 10 2012-08-29 17:33 pci-0000:00:04.0-virtio-pci-
↪virtio0-part5 -> ../../vda5
lrwxrwxrwx 1 root root 9 2012-08-29 17:33 pci-0000:00:06.0-virtio-pci-virtio2␣
↪-> ../../vdb
lrwxrwxrwx 1 root root 9 2012-08-29 17:33 pci-0000:00:08.0-virtio-pci-virtio3␣
↪-> ../../vdc
```

(continues on next page)

(continued from previous page)

```
lrwxrwxrwx 1 root root 9 2012-08-29 17:33 pci-0000:00:09.0-virtio-pci-virtio4␣
↪-> ../../vdd
lrwxrwxrwx 1 root root 10 2012-08-29 17:33 pci-0000:00:09.0-virtio-pci-
↪virtio4-part1 -> ../../vdd1
```

You might also have this problem after attaching and detaching the same volume from the same VM with the same mount point multiple times. In this case, restart the KVM host.

Failed to attach volume, systool is not installed

Problem

This warning and error occurs if you do not have the required `sysfsutils` package installed on the compute node:

```
WARNING nova.virt.libvirt.utils [req-1200f887-c82b-4e7c-a891-fac2e3735dbb\
admin admin|req-1200f887-c82b-4e7c-a891-fac2e3735dbb admin admin] systool\
is not installed
ERROR nova.compute.manager [req-1200f887-c82b-4e7c-a891-fac2e3735dbb admin\
admin|req-1200f887-c82b-4e7c-a891-fac2e3735dbb admin admin]
[instance: df834b5a-8c3f-477a-be9b-47c97626555c|instance: df834b5a-8c3f-47\
7a-be9b-47c97626555c]
Failed to attach volume 13d5c633-903a-4764-a5a0-3336945b1db1 at /dev/vdk.
```

Solution

Run the following command on the compute node to install the `sysfsutils` packages:

```
# apt-get install sysfsutils
```

Failed to connect volume in FC SAN

Problem

The compute node failed to connect to a volume in a Fibre Channel (FC) SAN configuration. The WWN may not be zoned correctly in your FC SAN that links the compute host to the storage array:

```
ERROR nova.compute.manager [req-2ddd5297-e405-44ab-aed3-152cd2cfb8c2 admin\
demo|req-2ddd5297-e405-44ab-aed3-152cd2cfb8c2 admin demo] [instance: 60ebd\
6c7-c1e3-4bf0-8ef0-f07aa4c3d5f3|instance: 60ebd6c7-c1e3-4bf0-8ef0-f07aa4c3\
d5f3]
Failed to connect to volume 6f6a6a9c-dfcf-4c8d-b1a8-4445ff883200 while\
attaching at /dev/vdjTRACE nova.compute.manager [instance: 60ebd6c7-c1e3-4\
bf0-8ef0-f07aa4c3d5f3|instance: 60ebd6c7-c1e3-4bf0-8ef0-f07aa4c3d5f3]
Traceback (most recent call last):f07aa4c3d5f3\] ClientException: The\
server has either erred or is incapable of performing the requested\
operation.(HTTP 500)(Request-ID: req-71e5132b-21aa-46ee-b3cc-19b5b4ab2f00)
```

Solution

The network administrator must configure the FC SAN fabric by correctly zoning the WWN (port names) from your compute node HBAs.

Cannot find suitable emulator for x86_64

Problem

When you attempt to create a VM, the error shows the VM is in the BUILD then ERROR state.

Solution

On the KVM host, run **cat /proc/cpuinfo**. Make sure the vmx or svm flags are set.

Follow the instructions in the Enable KVM section in the OpenStack Configuration Reference to enable hardware virtualization support in your BIOS.

Non-existent host

Problem

This error could be caused by a volume being exported outside of OpenStack using a host name different from the system name that OpenStack expects. This error could be displayed with the *IQN* if the host was exported using iSCSI.

```
2013-04-19 04:02:02.336 2814 ERROR cinder.openstack.common.rpc.common [-]␣
↪Returning exception Not found (HTTP 404)
NON_EXISTENT_HOST - HOST '10' was not found to caller.
```

Solution

Host names constructed by the driver use just the local host name, not the fully qualified domain name (FQDN) of the Compute host. For example, if the FQDN was **myhost.example.com**, just **myhost** would be used as the 3PAR host name. IP addresses are not allowed as host names on the 3PAR storage server.

Non-existent VLUN

Problem

This error occurs if the 3PAR host exists with the correct host name that the OpenStack Block Storage drivers expect but the volume was created in a different domain.

```
HTTPNotFound: Not found (HTTP 404) NON_EXISTENT_VLUN - VLUN 'osv-
↪DqT7CE3mSrWi4gZJmHAP-Q' was not found.
```

Solution

The hpe3par_domain configuration items either need to be updated to use the domain the 3PAR host currently resides in, or the 3PAR host needs to be moved to the domain that the volume was created in.

Database CPU spikes during operations

Query load upon the database can become a bottleneck that cascades across a deployment and ultimately degrades not only the Cinder service but also the whole OpenStack deployment.

Often, depending on load, query patterns, periodic tasks, and so on and so forth, additional indexes may be needed to help provide hints to the database so it can most efficently attempt to reduce the number of rows which need to be examined in order to return a result set.

Adding indexes

In older releases, before 2023.1 (Antelope), there were some tables that performed poorly in the presence of a large number of deleted resources (volumes, snapshots, backups, etc) which resulted in high CPU loads on the DB servers not only when listing those resources, but also when doing some operations on them. This was resolved by adding appropriate indexes to them.

This example below is specific to MariaDB/MySQL, but the syntax should be easy to modify for operators using PostgreSQL, and it represents the changes that older releases could add to resolve these DB server CPU spikes in such a way that they would not conflict with the ones that Cinder introduced in 2023.1 (Antelope).

```
use cinder;
create index groups_deleted_project_id_idx on groups (deleted, project_id);
create index group_snapshots_deleted_project_id_idx on groups (deleted,␣
↪project_id);
create index volumes_deleted_project_id_idx on volumes (deleted, project_id);
create index volumes_deleted_host_idx on volumes (deleted, host);
create index snapshots_deleted_project_id_idx on snapshots (deleted, project_
↪id);
create index backups_deleted_project_id_idx on backups (deleted, project_id);
```

Availability-zone types

Background

In a newly deployed region environment, the volume types (SSD, HDD or others) may only exist on part of the AZs, but end users have no idea which AZ is allowed for one specific volume type and they cant realize that only when the volume failed to be scheduled to backend. In this case, we have supported availability zone volume type in Rocky cycle which administrators can take advantage of to fix that.

How to config availability zone types?

We decided to use types extra-specs to store this additional info, administrators can turn it on by updating volume types key `RESKEY:availability_zones` as below:

```
"RESKEY:availability_zones": "az1,az2,az3"
```

Its an array list whose items are separated by comma and stored in string. Once the availability zone type is configured, any UI component or client can filter out invalid volume types based on their choice of availability zone:

```
Request example:
/v3/{project_id}/types?extra_specs={'RESKEY:availability_zones':'az1'}
```

Remember, Cinder will always try inexact match for this spec value, for instance, when extra spec `RESKEY:availability_zones` is configured with value `az1,az2`, both `az1` and `az2` are valid inputs for query, also this spec will not be used during performing capability filter, instead it will be only used for choosing suitable availability zones in these two cases below.

1. Create volume, within this feature, now we can specify availability zone via parameter `availability_zone`, volume source (volume, snapshot, group), configuration option `default_availability_zone` and `storage_availability_zone`. When creating new volume, Cinder will try to read the AZ(s) in the priority of:

```
source group > parameter availability_zone > source snapshot (or volume) >
↪volume type > configuration default_availability_zone > storage_
↪availability_zone
```

If there is a conflict between any of them, 400 BadRequest will be raised, also now a AZ list instead of single AZ will be delivered to `AvailabilityZoneFilter`.

2. Retype volume, this flow also has been updated, if new type has configured `RESKEY:availability_zones` Cinder scheduler will validate this as well.

Generalized filters

Background

Cinder introduced generalized resource filters since Pike. Administrator can control the allowed filter keys for **non-admin** user by editing the filter configuration file. Also since this feature, cinder will raise `400 BadRequest` if any invalid query filter is specified.

How do I configure the filter keys?

`resource_query_filters_file` is introduced to cinder to represent the filter config file path, and the config file accepts the valid filter keys for **non-admin** user with json format:

```
{
   "volume": ["name", "status", "metadata"]
}
```

the key `volume` (singular) here stands for the resource you want to apply and the value accepts an list which contains the allowed filters collection, once the configuration file is changed and API service is restarted, cinder will only recognize this filter keys, **NOTE**: the default configuration file will include all the filters we already enabled.

Which filter keys are supported?

Not all the attributes are supported at present, so we add this table below to indicate which filter keys are valid and can be used in the configuration.

Since v3.34 we could use ~ to indicate supporting querying resource by inexact match, for example, if we have a configuration file as below:

```
{
   "volume": ["name~"]
}
```

User can query volume both by `name=volume` and `name~=volume`, and the volumes named `volume123` and `a_volume123` are both valid for second input while neither are valid for first. The supported APIs are marked with * below in the table.

API	Valid filter keys
list volume*	id, group_id, name, status, bootable, migration_status, metadata, host, image_metadata, availability_zone, user_id, volume_type_id, project_id, size, description, replication_status, multiattach
list snapshot*	id, volume_id, user_id, project_id, status, volume_size, name, description, volume_type_id, group_snapshot_id, metadata, availability_zone
list backup*	id, name, status, container, availability_zone, description, volume_id, is_incremental, size, host, parent_id
list group*	id, user_id, status, availability_zone, group_type, name, description, host
list g-snapshot	id, name, description, group_id, group_type_id, status
list attachment*	id, volume_id, instance_id, attach_status, attach_mode, connection_info, mountpoint, attached_host
list message*	id, event_id, resource_uuid, resource_type, request_id, message_level, project_id
get pools	name, volume_type
list types (3.52)	is_public, extra_specs

Basic volume quality of service

Basic volume QoS allows you to define hard performance limits for volumes on a per-volume basis.

Performance parameters for attached volumes are controlled using volume types and associated extra-specs.

As of the 13.0.0 Rocky release, Cinder supports the following options to control volume quality of service, the values of which should be fairly self-explanatory:

For Fixed IOPS per volume.

- *read_iops_sec*
- *write_iops_sec*
- *total_iops_sec*

For Burst IOPS per volume.

- *read_iops_sec_max*
- *write_iops_sec_max*
- *total_iops_sec_max*

For Fixed bandwidth per volume.

- *read_bytes_sec*
- *write_bytes_sec*
- *total_bytes_sec*

For Burst bandwidth per volume.

- *read_bytes_sec_max*
- *write_bytes_sec_max*
- *total_bytes_sec_max*

For burst bucket size.

- *size_iops_sec*

Note that the *total_** and *total_*_max* options for both iops and bytes cannot be used with the equivalent *read* and *write* values.

For example, in order to create a QoS extra-spec with 20000 read IOPs and 10000 write IOPs, you might use the Cinder client in the following way:

```
$ cinder qos-create high-iops consumer="front-end" \
  read_iops_sec=20000 write_iops_sec=10000
+----------+--------------------------------------+
| Property | Value                                |
+----------+--------------------------------------+
| consumer | front-end                            |
| id       | f448f61c-4238-4eef-a93a-2024253b8f75 |
| name     | high-iops                            |
| specs    | read_iops_sec : 20000                |
|          | write_iops_sec : 10000               |
+----------+--------------------------------------+
```

The equivalent OpenStack client command would be:

```
$ openstack volume qos create --consumer "front-end" \
  --property "read_iops_sec=20000" \
  --property "write_iops_sec=10000" \
  high-iops
```

Once this is done, you can associate this QoS with a volume type by using the *qos-associate* Cinder client command.

```
$ cinder qos-associate QOS_ID VOLUME_TYPE_ID
```

or using the *openstack volume qos associate* OpenStack client command.

```
$ openstack volume qos associate QOS_ID VOLUME_TYPE_ID
```

You can now create a new volume and attempt to attach it to a consumer such as Nova. If you login to the Nova compute host, youll be able to see the assigned limits when checking the XML definition of the virtual machine with *virsh dumpxml*.

Note

As of the Nova 18.0.0 Rocky release, front end QoS settings are only supported when using the libvirt driver.

Volume multi-attach: Enable attaching a volume to multiple servers

The ability to attach a volume to multiple hosts/servers simultaneously is a use case desired for active/active or active/standby scenarios.

Support was added in both Cinder and Nova in the Queens release to volume multi-attach with read/write (RW) mode.

Warning

It is the responsibility of the user to ensure that a multiattach or clustered file system is used on the volumes. Otherwise there may be a high probability of data corruption.

In Cinder the functionality is available from microversion 3.50 or higher.

As a prerequisite new Attach/Detach APIs were added to Cinder in Ocata to overcome earlier limitations towards achieving volume multi-attach.

In case you use Cinder together with Nova, compute API calls were switched to using the new block storage volume attachment APIs in Queens, if the required block storage API microversion is available.

For more information on using multiattach volumes with the compute service, refer to the corresponding compute admin guide section.

How to create a multiattach volume

In order to be able to attach a volume to multiple server instances you need to have the multiattach flag set to True in the volume details. Please ensure you have the right role and policy settings before performing the operation.

The only way to create a multiattach volume is by creating a multiattach volume type and using it to create the volume.

Note

For information on back ends that provide the functionality see *Back end support*.

Multiattach volume type

Starting from the Queens release the ability to attach a volume to multiple hosts/servers requires that the volume is of a special type that includes an extra-spec capability setting of `multiattach=<is> True`. You can create the volume type the following way:

```
$ openstack volume type create multiattach
$ openstack volume type set --property multiattach="<is> True"  multiattach
```

Note

Creating a new volume type is an admin-only operation by default. You can change the settings in the cinder policy file if needed. For more information about configuring cinder policies, see *Policy configuration*.

To create the volume you need to use the volume type you created earlier, like this:

```
$ openstack volume create --size 10 --type multiattach my-volume
```

In addition, it is possible to retype a volume to be (or not to be) multiattach capable. Currently however we only allow retyping a volume if its status is `available`.

The reasoning behind the limitation is that some consumers/hypervisors need to make special considerations at attach-time for multiattach volumes (like disable caching) and theres no mechanism currently to update a currently attached volume in a safe way while keeping it attached the whole time.

RO / RW caveats (the secondary RW attachment issue)

By default, secondary volume attachments are made in read/write mode which can be problematic, especially for operations like volume migration.

There might be improvements to provide support to specify the attach-mode for the secondary attachments, for the latest information please take a look into Cinders specs list for the current release.

Back end support

In order to have the feature available, multi-attach needs to be supported by the chosen back end which is indicated through capabilities in the corresponding volume driver.

The reference implementation is available on LVM in the Queens release. You can check the *Driver Support Matrix* for further information on which back end provides the functionality.

Policy rules

You can control the availability of volume multi-attach through policies that you can configure in the cinder policy file. For more information about the cinder policy file, including how to generate a sample file so you can view the default policy settings, see *Policy configuration*.

Multiattach policy

The general policy rule to allow the creation or retyping of multiattach volumes is named `volume:multiattach`.

Multiattach policy for bootable volumes

This is a policy to disallow the ability to create multiple attachments on a volume that is marked as bootable with the name `volume:multiattach_bootable_volume`.

Known issues and limitations

- Retyping an in-use volume from a multiattach-capable type to a non-multiattach-capable type, or vice-versa, is not supported.
- It is not recommended to retype an in-use multiattach volume if that volume has more than one active read/write attachment.
- Encryption is not supported with multiattach-capable volumes.

Default Volume Types

Beginning with the Train release, untyped volumes (that is, volumes with no volume-type) have been disallowed. To facilitate this, a `__DEFAULT__` volume-type was included as part of the Train database migration. Since the Train release, handling of the default volume-type has been improved:

- The default_volume_type configuration option is required to have a value. The default value is `__DEFAULT__`.
- A request to delete the currently configured default_volume_type will fail. (You can delete that volume-type, but you cannot do it while it is the value of the configuration option.)
- There must always be at least one volume-type defined in a Cinder installation. This is enforced by the type-delete call.
- If the default_volume_type is misconfigured (that is, if the value refers to a non-existent volume-type), requests that rely on the default volume-type (for example, a volume-create request that does not specify a volume-type) will result in a HTTP 500 response.

Default types per project

We have overriden the existing Cinder default Volume Type on a per project basis to make it easier to manage complex deployments.

With the introduction of this new default volume type support, well now have 2 different default volume types. From more specific to more generic these are:

- Per project
- Defined in cinder.conf (defaults to `__DEFAULT__` type)

So when a user creates a new volume that has no defined volume type (explicit or in the source), Cinder will look for the appropriate default first by checking if theres one defined in the DB for the specific project and use it, if there isnt one, it will continue like it does today, using the default type from cinder.conf.

Administrators and users must still be careful with the normal Cinder behavior when creating volumes, as Cinder will still only resort to using the default volume type if the user doesnt select one on the request or if theres no volume type in the source, which means that Cinder will not use any of those defaults if we:

- Create a volume providing a volume type
- Create a volume from a snapshot
- Clone a volume
- Create a volume from an image that has cinder_img_volume_type defined in its metadata.

There is a new set of commands in the python-cinderclient to match the new REST API endpoints:

- Set default: `cinder default-type-set <project-id> <type-name>`
- Unset default: `cinder default-type-unset <project-id>`
- List defaults: `cinder default-type-list [--project <project-id>]`

By default the policy restricting access to set, unset, get or list all project default volume type is set to admins only.

API Configuration

Rate limiting

Warning

This is legacy functionality that is poorly tested and may be removed in the future. You may wish to enforce rate limiting through a proxy server instead.

Cinder supports admin-configured API limits. These are disabled by default but can be configured by modifying `api-paste.ini` to enabled the `RateLimitingMiddleware` middleware. For example, given the following composite application definitions in e.g. `/etc/cinder/api-paste.ini`:

```
[composite:openstack_volume_api_v2]
use = call:cinder.api.middleware.auth:pipeline_factory
noauth = cors ... apiv2
keystone = cors ... apiv2
keystone_nolimit = cors ... apiv2

[composite:openstack_volume_api_v3]
use = call:cinder.api.middleware.auth:pipeline_factory
noauth = cors ... apiv3
keystone = cors ... apiv3
keystone_nolimit = cors ... apiv3
```

You can configure rate limiting by adding a new filter to call `RateLimitingMiddleware` and configure the composite applications to use this filter:

```
[composite:openstack_volume_api_v2]
use = call:cinder.api.middleware.auth:pipeline_factory
noauth = cors ... ratelimit apiv2
keystone = cors ... ratelimit apiv2
keystone_nolimit = cors ... ratelimit apiv2

[composite:openstack_volume_api_v3]
use = call:cinder.api.middleware.auth:pipeline_factory
noauth = cors ... ratelimit apiv3
keystone = cors ... ratelimit apiv3
keystone_nolimit = cors ... ratelimit apiv3

[filter:ratelimit]
paste.filter_factory = cinder.api.v2.limits:RateLimitingMiddleware.factory
```

Once configured, restart the **cinder-api** service. Users can then view API limits using the `openstack limits show --rate` command. For example:

```
$ openstack limits show --rate
+--------+----------------+-------+--------+--------+---------------------+
| Verb   | URI            | Value | Remain | Unit   | Next Available      |
+--------+----------------+-------+--------+--------+---------------------+
| POST   | *              |    10 |     10 | MINUTE | 2021-03-23T12:36:09 |
| PUT    | *              |    10 |     10 | MINUTE | 2021-03-23T12:36:09 |
| DELETE | *              |   100 |    100 | MINUTE | 2021-03-23T12:36:09 |
| POST   | */servers      |    50 |     50 | DAY    | 2021-03-23T12:36:09 |
| GET    | *changes-since* |    3 |      3 | MINUTE | 2021-03-23T12:36:09 |
+--------+----------------+-------+--------+--------+---------------------+
```

Note

Rate limits are entirely separate from absolute limits, which track resource utilization and can be seen using the `openstack limits show --absolute` command.

Upgrades

Cinder aims to provide upgrades with minimal downtime.

This should be achieved for both data and control plane. As Cinder doesnt interfere with data plane, its upgrade shouldnt affect any volumes being accessed by virtual machines.

Keeping the control plane running during an upgrade is more difficult. This documents goal is to provide preliminaries and a detailed procedure of such upgrade.

Concepts

Here are the key concepts you need to know before reading the section on the upgrade process:

RPC version pinning

Through careful RPC versioning, newer services are able to talk to older services (and vice-versa). The versions are autodetected using information reported in `services` table. In case of receiving `CappedVersionUnknown` or `ServiceTooOld` exceptions on service start, youre probably having some old orphaned records in that table.

Graceful service shutdown

Many cinder services are python processes listening for messages on an AMQP queue. When the operator sends SIGTERM signal to the process, it stops getting new work from its queue, completes any outstanding work and then terminates. During this process, messages can be left on the queue for when the python process starts back up. This gives us a way to shutdown a service using older code, and start up a service using newer code with minimal impact.

Note

Waiting for completion of long-running operations (e.g. slow volume copy operation) may take a while.

Note

This was tested with RabbitMQ messaging backend and may vary with other backends.

Database upgrades

Cinder has two types of database upgrades in use:

- Schema migrations
- Data migrations

Schema migrations are defined in `cinder/db/migrations/versions`. They are the routines that transform our database structure, which should be additive and able to be applied to a running system before service code has been upgraded.

Data migrations are banned from schema migration scripts and are instead defined in `cinder/db/api.py`. They are kept separate to make DB schema migrations less painful to execute. Instead, the migrations are executed by a background process in a manner that doesnt interrupt running services (you can also execute online data migrations with services turned off if youre doing a cold upgrade). The `cinder-manage db online_data_migrations` utility can be used for this purpose. Before upgrading N to N+1, you need to run this tool in the background until it tells you no more migrations are needed. Note that you wont be able to apply N+1s schema migrations before completing Ns online data migrations.

For information on developing your own schema or data migrations as part of a feature or bugfix, refer to *Database migrations*.

Note

Occasionally we receive reports of database issues during upgrades due to an old version of database software being used (see, for example, Bug #1968746).

Upgrades are tested in the gate using the mysql or mariadb version packaged with any of the Linux distributions supported for that release. Thus, if you are using an earlier version of mysql or mariadb, you may want to do additional research before upgrading to make sure you wont run into an issue caused by us using a newer feature than your database version supports.

To assist you in this, the OpenStack Technical Committee maintains a list of the Linux distributions supported for each release on the Release based Runtimes page.

API load balancer draining

When upgrading API nodes, you can make your load balancer only send new connections to the newer API nodes, allowing for a seamless update of your API nodes.

DB prune deleted rows

Currently resources are soft deleted in the database, so users are able to track instances in the DB that are created and destroyed in production. However, most people have a data retention policy, of say 30 days or 90 days after which they will want to delete those entries. Not deleting those entries affects DB performance as indices grow very large and data migrations take longer as there is more data to migrate. To make pruning easier theres a `cinder-manage db purge <age_in_days>` command that permanently deletes records older than specified age.

Versioned object backports

RPC pinning ensures new services can talk to the older services method signatures. But many of the parameters are objects that may well be too new for the old service to understand. Cinder makes sure to backport an object to a version that it is pinned to before sending.

Minimal Downtime Upgrade Procedure

Plan your upgrade

- Read and ensure you understand the release notes for the next release.
- Make a backup of your database. Cinder does not support downgrading of the database. Hence, in case of upgrade failure, restoring database from backup is the only choice.
- To avoid dependency hell it is advised to have your Cinder services deployed separately in containers or Python venvs.

> **Note**
>
> Cinder is basing version detection on what is reported in the `services` table in the DB. Before upgrade make sure you dont have any orphaned old records there, because these can block starting newer services. You can clean them up using `cinder-manage service remove <binary> <host>` command.

Note that theres an assumption that live upgrade can be performed only between subsequent releases. This means that you cannot upgrade N directly to N+2, you need to upgrade to N+1 first.

The assumed service upgrade order is `cinder-scheduler`, `cinder-volume`, `cinder-backup` and finally `cinder-api`.

Rolling upgrade process

To reduce downtime, the services can be upgraded in a rolling fashion. It means upgrading a few services at a time. To minimise downtime you need to have HA Cinder deployment, so at the moment a service is upgraded, youll keep other service instances running.

Before maintenance window

- First you should execute required DB schema migrations. To achieve that without interrupting your existing installation, install new Cinder code in new venv or a container and run the DB sync (`cinder-manage db sync`). These schema change operations should have minimal or no effect on performance, and should not cause any operations to fail.

- At this point, new columns and tables may exist in the database. These DB schema changes are done in a way that both the N and N+1 release can perform operations against the same schema.

During maintenance window

1. The first service is cinder-scheduler. It is load-balanced by the message queue, so the only thing you need to worry about is to shut it down gracefully (using `SIGTERM` signal) to make sure it will finish all the requests being processed before shutting down. Then you should upgrade the code and restart the service.
2. Repeat first step for all of your cinder-scheduler services.
3. Then you proceed to upgrade cinder-volume services. The problem here is that due to Active/Passive character of this service, youre unable to run multiple instances of cinder-volume managing a single volume backend. This means that there will be a moment when you wont have any cinder-volume in your deployment and you want that disruption to be as short as possible.

   Note

   The downtime here is non-disruptive as long as it doesnt exceed the service heartbeat timeout. If you dont exceed that, then cinder-schedulers will not notice that cinder-volume is gone and the message queue will take care of queuing any RPC messages until cinder-volume is back.

   To make sure its achieved, you can either lengthen the timeout by tweaking `service_down_time` value in `cinder.conf`, or prepare upgraded cinder-volume on another node and do a very quick switch by shutting down older service and starting the new one just after that.

   Also note that in case of A/P HA configuration you need to make sure both primary and secondary c-vol have the same hostname set (you can override it using `host` option in `cinder.conf`), so both will be listening on the same message queue and will accept the same messages.

4. Repeat third step for all cinder-volume services.
5. Now we should proceed with (optional) cinder-backup services. You should upgrade them in the same manner like cinder-scheduler.

   Note

   Backup operations are time consuming, so shutting down a c-bak service without interrupting ongoing requests can take time. It may be useful to disable the service first using `cinder service-disable` command, so it wont accept new requests, and wait a reasonable amount of time until all the in-progress jobs are completed. Then you can proceed with the upgrade. To make sure the backup service finished all the ongoing requests, you can check the service logs.

   Note

   Until Liberty cinder-backup was tightly coupled with cinder-volume service and needed to coexist on the same physical node. This is not true starting with Mitaka version. If youre still keeping that coupling, then your upgrade strategy for cinder-backup should be more similar to how cinder-volume is upgraded.

6. cinder-api services should go last. In HA deployment youre typically running them behind a load balancer (e.g. HAProxy), so you need to take one service instance out of the balancer, shut it down, upgrade the code and dependencies, and start the service again. Then you can plug it back into the load balancer.

   Note

   You may want to start another instance of older c-api to handle the load while youre upgrading your original services.

7. Then you should repeat step 6 for all of the cinder-api services.

After maintenance window

- Once all services are running the new code, double check in the DB that there are no old orphaned records in `services` table (Cinder doesnt remove the records when service is gone or service hostname is changed, so you need to take care of that manually; you should be able to distinguish dead records by looking at when the record was updated). Cinder is basing its RPC version detection on that, so stale records can prevent you from going forward.
- Now all services are upgraded, we need to send the `SIGHUP` signal, so all the services clear any cached service version data. When a new service starts, it automatically detects which version of the services RPC protocol to use, and will downgrade any communication to that version. Be advised that cinder-api service doesnt handle `SIGHUP` so it needs to be restarted. Its best to restart your cinder-api services as last ones, as that way you make sure API will fail fast when user requests new features on a deployment thats not fully upgraded (new features can fail when RPC messages are backported to lowest common denominator). Order of the rest of the services shouldnt matter.
- Now all the services are upgraded, the system is able to use the latest version of the RPC protocol and able to access all the features of the new release.
- At this point, you must also ensure you update the configuration, to stop using any deprecated features or options, and perform any required work to transition to alternative features. All the deprecated options should be supported for one cycle, but should be removed before your next upgrade is performed.
- Since Ocata, you also need to run `cinder-manage db online_data_migrations` command to make sure data migrations are applied. The tool lets you limit the impact of the data migrations by using `--max_count` option to limit number of migrations executed in one run. If this option is used, the exit status will be 1 if any migrations were successful (even if others generated errors, which could be due to dependencies between migrations). The command should be rerun while the exit status is 1. If no further migrations are possible, the exit status will be 2 if some migrations are still generating errors, which requires intervention to resolve. The command should be considered completed successfully only when the exit status is 0. You need to complete all of the migrations before starting upgrade to the next version (e.g. you need to complete Ocatas data migrations before proceeding with upgrade to Pike; you wont be able to execute Pikes DB schema migrations before completing Ocatas data migrations).

3.3 Reference

Contents:

3.3.1 Cinder Service Configuration

Introduction to the Block Storage service

The Block Storage service provides persistent block storage resources that Compute instances can consume. This includes secondary attached storage similar to the Amazon Elastic Block Storage (EBS) offering. In addition, you can write images to a Block Storage device for Compute to use as a bootable persistent instance.

The Block Storage service differs slightly from the Amazon EBS offering. The Block Storage service does not provide a shared storage solution like NFS. With the Block Storage service, you can attach a device to only one instance.

The Block Storage service provides:

- `cinder-api` - a WSGI app that authenticates and routes requests throughout the Block Storage service. It supports the OpenStack APIs only, although there is a translation that can be done through Computes EC2 interface, which calls in to the Block Storage client.
- `cinder-scheduler` - schedules and routes requests to the appropriate volume service. Depending upon your configuration, this may be simple round-robin scheduling to the running volume services, or it can be more sophisticated through the use of the Filter Scheduler. The Filter Scheduler is the default and enables filters on things like Capacity, Availability Zone, Volume Types, and Capabilities as well as custom filters.
- `cinder-volume` - manages Block Storage devices, specifically the back-end devices themselves.
- `cinder-backup` - provides a means to back up a Block Storage volume to OpenStack Object Storage (swift).

The Block Storage service contains the following components:

- **Back-end Storage Devices** - the Block Storage service requires some form of back-end storage that the service is built on. The default implementation is to use LVM on a local volume group named cinder-volumes. In addition to the base driver implementation, the Block Storage service also provides the means to add support for other storage devices to be utilized such as external Raid Arrays or other storage appliances. These back-end storage devices may have custom block sizes when using KVM or QEMU as the hypervisor.
- **Users and Tenants (Projects)** - the Block Storage service can be used by many different cloud computing consumers or customers (tenants on a shared system), using role-based access assignments. Roles control the actions that a user is allowed to perform. In the default configuration, most actions do not require a particular role, but this can be configured by the system administrator in the cinder policy file that maintains the rules.

  Note

  For more information about configuring cinder policies, see *Policy configuration*.

  A users access to particular volumes is limited by tenant, but the user name and password are assigned per user. Key pairs granting access to a volume are enabled per user, but quotas to control resource consumption across available hardware resources are per tenant.

For tenants, quota controls are available to limit:

- The number of volumes that can be created.
- The number of snapshots that can be created.
- The total number of GBs allowed per tenant (shared between snapshots and volumes).

You can revise the default quota values with the Block Storage CLI, so the limits placed by quotas are editable by admin users.

- **Volumes, Snapshots, and Backups** - the basic resources offered by the Block Storage service are volumes and snapshots which are derived from volumes and volume backups:
  - **Volumes** - allocated block storage resources that can be attached to instances as secondary storage or they can be used as the root store to boot instances. Volumes are persistent R/W block storage devices most commonly attached to the compute node through iSCSI.
  - **Snapshots** - a read-only point in time copy of a volume. The snapshot can be created from a volume that is currently in use (through the use of `--force True`) or in an available state. The snapshot can then be used to create a new volume through create from snapshot.
  - **Backups** - an archived copy of a volume currently stored in Object Storage (swift).

Using service tokens

Warning

For all OpenStack releases after 2023-05-10, it is **required** that Nova be configured to send a service token to Cinder and Cinder to receive it. This is required by the fix for CVE-2023-2088. See OSSA-2023-003 for details.

When a user initiates a request whose processing involves multiple services (for example, a boot-from-volume request to the Compute Service will require processing by the Block Storage Service, and may require processing by the Image Service), the users token is handed from service to service. This ensures that the requestor is tracked correctly for audit purposes and also guarantees that the requestor has the appropriate permissions to do what needs to be done by the other services.

There are several instances where we want to differentiate between a request coming from the user to one coming from another OpenStack service on behalf of the user:

- **For security reasons** There are some operations in the Block Storage service, required for normal operations, that could be exploited by a malicious user to gain access to resources belonging to other users. By differentiating when the request comes directly from a user and when from another OpenStack service the Cinder service can protect the deployment.
- To prevent long-running job failures: If the chain of operations takes a long time, the users token may expire before the action is completed, leading to the failure of the users original request.

  One way to deal with this is to set a long token life in Keystone, and this may be what you are currently doing. But this can be problematic for installations whose security policies prefer short user token lives. Beginning with the Queens release, an alternative solution is available. You have the ability to configure some services (particularly Nova and Cinder) to send a service token along with the users token. When properly configured, the Identity Service will validate an expired user token *when it is accompanied by a valid service token*. Thus if the users token expires somewhere during a long running chain of operations among various OpenStack services, the operations can continue.

Note

Theres nothing special about a service token. Its a regular token that has been requested by a service user. And theres nothing special about a service user, its just a user that has been configured in the Identity Service to have specific roles that identify that user as a service.

The key point here is that the service token doesnt need to have an extra long life it can have the same short life as all the other tokens because it will be a **fresh** (and hence valid) token accompanying the (possibly expired) users token.

Configuration

To configure an OpenStack service that supports Service Tokens, like Nova and Cinder, to send a service token along with the users token when it makes a request to another service, you must do the following:

1. Configure the sender services to send the token when calling other OpenStack services.
2. Configure each services user to have a service role in Keystone.
3. Configure the receiver services to expect the token and validate it appropriately on reception.

Send service token

To send the token we need to add to our configuration file the `[service_user]` section and fill it in with the appropriate configuration for your service user (`username`, `project_name`, etc.) and set the `send_service_user_token` option to `true` to tell the service to send the token.

The configuration for the service user is basically the normal keystone user configuration like we would have in the `[keystone_authtoken]` section, but without the 2 configuration options well see in one of the next subsection to configure the reception of service tokens.

In most cases we would use the same user we do in `[keystone_authtoken]`, for example for the nova configuration we would have something like this:

```
[service_user]
send_service_user_token = True

# Copy following options from [keystone_authtoken] section
project_domain_name = Default
project_name = service
user_domain_name = Default
password = abc123
username = nova
auth_url = http://192.168.121.66/identity
auth_type = password
```

Service role

A service role is nothing more than a Keystone role that allows a deployment to identify a service without the need to make them admins, that way there is no change in the privileges but we are able to identify that the request is coming from another service and not a user.

The default service role is `service`, but we can use a different name or even have multiple service roles. For simplicitys sake we recommend having just one, `service`.

We need to make sure that the user configured in the `[service_user]` section for a project has a service role.

Assuming our users are `nova` and `cinder` from the `service` project and the service role is going to be the default `service`, we first check if the role exists or not:

```
$ openstack role show service
```

If it doesnt, we need to create it

```
$ openstack role create service
```

Check if the users have the roles assigned or not:

```
$ openstack role assignment list --user cinder --project service --names
$ openstack role assignment list --user nova --project service --names
```

And if they are not we assign the role to those users

```
$ openstack role add --user cinder --project service service
$ openstack role add --user nova --project service service
```

More information on creating service users can be found in the Keystone documentation

Receive service token

Now we need to make the services validate the service token on reception, this part is crucial.

The 2 configuration options in `[keystone_authoken]` related to receiving service tokens are `service_token_roles` and `service_token_roles_required`.

The `service_token_roles` contains a list of roles that we consider to belong to services. The service user must belong to at least one of them to be considered a valid service token. The value defaults to `service`, so we dont need to set it if thats the value we are using.

Now we need to tell the keystone middleware to actually validate the service token and confirm that its not only a valid token, but that it has one of the roles set in `service_token_roles`. We do this by setting `service_token_roles_required` to `true`.

So we would have something like this in our `[keystone_authtoken]` section:

```
[keystone_authtoken]
service_token_roles = service
service_token_roles_required = true
```

Troubleshooting

If youve configured this feature and are still having long-running job failures, there are basically three degrees of freedom to take into account: (1) each source service, (2) each receiving service, and (3) the Identity Service (Keystone).

1. Each source service (basically, Nova and Cinder) must have the `[service_user]` section in the **source service** configuration file filled in as described in the *Configuration* section above.

   > **Note**
   >
   > As of the 2023.1 release, Glance does not have the ability to pass service tokens. It can receive them, though. The place where you may still see a long running failure is when Glance is using a backend that requires Keystone validation (for example, the Swift backend) and the user token has expired.

2. There are several things to pay attention to in Keystone:

   - When `service_token_roles_required` is enabled you must make sure that any service user who will be contacting that receiving service (and for whom you want to enable service token usage) has one of the roles specified in the receiving servicess `service_token_roles` setting. (This is a matter of creating and assigning roles using the Identity Service API, its not a configuration file issue.)
   - Even with a service token, an expired user token cannot be used indefinitely. Theres a Keystone configuration setting that controls this: `[token]/allow_expired_window` in the **Keystone** configuration file. The default setting is 2 days, so some security teams may want to lower this just on general principles. You need to make sure its not set too low to be completely ineffective.
   - If you are using Fernet tokens, you need to be careful with your Fernet key rotation period. Whoever sets up the key rotation has to pay attention to the `[token]/allow_expired_window` setting as well as the obvious `[token]/expiration` setting. If keys get rotated faster than `expiration` + `allow_expired_window` seconds, an expired user token might not be decryptable, even though the request using it is being made within `allow_expired_window` seconds.

To summarize, you need to be aware of:

- Keystone: must allow a decent sized `allow_expired_window` (default is 2 days)
- Each source service: must be configured to be able to create and send service tokens (default is OFF)
- Each receiving service: has to be configured to accept service tokens (default is ON) and require role verification (default is OFF)

Volume drivers

To use different volume drivers for the cinder-volume service, use the parameters described in these sections.

These volume drivers are included in the Block Storage repository. To set a volume driver, use the `volume_driver` flag.

The default is:

```
volume_driver = cinder.volume.drivers.lvm.LVMVolumeDriver
```

Note that some third party storage systems may maintain more detailed configuration documentation elsewhere. Contact your vendor for more information if needed.

Driver Configuration Reference

Ceph RADOS Block Device (RBD)

If you use KVM, QEMU or Hyper-V as your hypervisor, you can configure the Compute service to use Ceph RADOS block devices (RBD) for volumes.

Ceph is a massively scalable, open source, distributed storage system. It is comprised of an object store, block store, and a POSIX-compliant distributed file system. The platform can auto-scale to the exabyte level and beyond. It runs on commodity hardware, is self-healing and self-managing, and has no single point of failure. Due to its open-source nature, you can install and use this portable storage platform in public or private clouds.

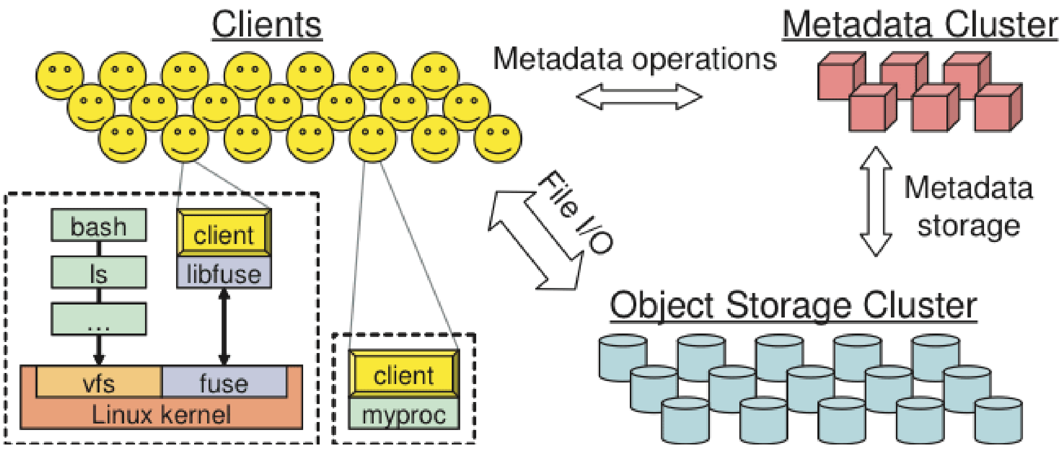


Fig. 1: Ceph architecture

Note

Supported Ceph versions

The current release cycle model for Ceph targets a new release yearly on 1 March, with there being at most two active stable releases at any time.

For a given OpenStack release, *Cinder supports the current Ceph active stable releases plus the two prior releases.*

For example, at the time of the OpenStack Wallaby release in April 2021, the Ceph active supported releases are Pacific and Octopus. The Cinder Wallaby release therefore supports Ceph Pacific, Octopus, Nautilus, and Mimic.

Additionally, it is expected that the version of the Ceph client available to Cinder or any of its associated libraries (os-brick, cinderlib) is aligned with the Ceph server version. Mixing server and client versions is *unsupported* and may lead to anomalous behavior.

The minimum requirements for using Ceph with Hyper-V are Ceph Pacific and Windows Server 2016.

RADOS

Ceph is based on Reliable Autonomic Distributed Object Store (RADOS). RADOS distributes objects across the storage cluster and replicates objects for fault tolerance. RADOS contains the following major components:

Object Storage Device (OSD) Daemon
: The storage daemon for the RADOS service, which interacts with the OSD (physical or logical storage unit for your data). You must run this daemon on each server in your cluster. For each OSD, you can have an associated hard drive disk. For performance purposes, pool your hard drive disk with raid arrays, or logical volume management (LVM). By default, the following pools are created: data, metadata, and RBD.

Meta-Data Server (MDS)
: Stores metadata. MDSs build a POSIX file system on top of objects for Ceph clients. However, if you do not use the Ceph file system, you do not need a metadata server.

Monitor (MON)
: A lightweight daemon that handles all communications with external applications and clients. It also provides a consensus for distributed decision making in a Ceph/RADOS cluster. For instance, when you mount a Ceph shared on a client, you point to the address of a MON server. It checks the state and the consistency of the data. In an ideal setup, you must run at least three `ceph-mon` daemons on separate servers.

Ways to store, use, and expose data

To store and access your data, you can use the following storage systems:

RADOS
: Use as an object, default storage mechanism.

RBD
: Use as a block device. The Linux kernel RBD (RADOS block device) driver allows striping a Linux block device over multiple distributed object store data objects. It is compatible with the KVM RBD image.

CephFS
: Use as a file, POSIX-compliant file system.

Ceph exposes RADOS; you can access it through the following interfaces:

RADOS Gateway
: OpenStack Object Storage and Amazon-S3 compatible RESTful interface (see RADOS_Gateway).

librados
: and its related C/C++ bindings

RBD and QEMU-RBD
: Linux kernel and QEMU block devices that stripe data across multiple objects.

RBD pool

The RBD pool used by the Cinder backend is configured with option `rbd_pool`, and by default the driver expects exclusive management access to that pool, as in being the only system creating and deleting resources in it, since thats the recommended deployment choice.

Pool sharing is strongly discouraged, and if we were to share the pool with other services, within OpenStack (Nova, Glance, another Cinder backend) or outside of OpenStack (oVirt), then the stats returned by the driver to the scheduler would not be entirely accurate.

The inaccuracy would be that the actual size in use by the cinder volumes would be lower than the reported one, since it would be also including the used space by the other services.

We can set the `rbd_exclusive_cinder_pool` configuration option to `false` to fix this inaccuracy, but this has a performance impact.

Warning

Setting `rbd_exclusive_cinder_pool` to `false` will increase the burden on the Cinder driver and the Ceph cluster, since a request will be made for each existing image, to retrieve its size, during the stats gathering process.

For deployments with large amount of volumes it is recommended to leave the default value of `true`, and accept the inaccuracy, as it should not be particularly problematic.

Driver options

The following table contains the configuration options supported by the Ceph RADOS Block Device driver.

Table 1: Description of Ceph storage configuration options

Configuration option = Default value	Description
deferred_ = 0	(Integer) Time delay in seconds before a volume is eligible for permanent removal after being tagged for deferred deletion.
deferred_ = 60	(Integer) Number of seconds between runs of the periodic task to purge volumes tagged for deletion.
enable_de = False	(Boolean) Enable deferred deletion. Upon deletion, volumes are tagged for deletion but will only be removed asynchronously at a later time.
rados_con = -1	(Integer) Timeout value (in seconds) used when connecting to ceph cluster. If value < 0, no timeout is set and default librados value is used.
rados_con = 5	(Integer) Interval value (in seconds) between connection retries to ceph cluster.
rados_con = 3	(Integer) Number of retries if connection to ceph cluster failed.
rbd_ceph_ = <>	(String) Path to the ceph configuration file
rbd_clust = ceph	(String) The name of ceph cluster
rbd_concu = 3	(Integer(min=0)) Number of flatten operations that will run concurrently on this volume service.
rbd_exclu = True	(Boolean) Set to False if the pool is shared with other usages. On exclusive use driver wont query images provisioned size as they will match the value calculated by the Cinder core code for allocated_capacity_gb. This reduces the load on the Ceph cluster as well as on the volume service. On non exclusive use driver will query the Ceph cluster for per image used disk, this is an intensive operation having an independent request for each image.
rbd_flatt = False	(Boolean) Flatten volumes created from snapshots to remove dependency from volume to snapshot
rbd_max_c = 5	(Integer) Maximum number of nested volume clones that are taken before a flatten occurs. Set to 0 to disable cloning. Note: lowering this value will not affect existing volumes whose clone depth exceeds the new value.
rbd_pool = rbd	(String) The RADOS pool where RBD volumes are stored
rbd_secre = None	(String) The libvirt uuid of the secret for the rbd_user volumes. Defaults to the cluster FSID.
rbd_store = 4	(Integer) Volumes will be chunked into objects of this size (in megabytes).
rbd_user = None	(String) The RADOS client name for accessing RBD volumes - only set when using cephx authentication
replicati = 5	(Integer) Timeout value (in seconds) used when connecting to ceph cluster to do a demotion/promotion of volumes. If value < 0, no timeout is set and default librados value is used.
report_dy = True	(Boolean) Set to True for driver to report total capacity as a dynamic value (used + current free) and to False to report a static value (quota max bytes if defined and global size of cluster if not).

RBD Mirroring

The cinder RBD driver supports mirroring between multiple clusters. You can configure it on the cinder side with the usual replication configuration. Refer to the *documentation* for more information.

You will also have to configure replication on the Ceph side. To do so you may refer to the Ceph documentation.

Note that with the RBD driver in cinder you need to configure the pool replication option in image mode. For instance, if your pool is named `volumes`, the command would be: `rbd mirror pool enable volumes image`.

RBD QoS

Currently, the Cinder RBD driver supports the following QoS options compatible with Ceph Octopus release and above:

Cinder Value	Ceph Mapping
`total_iops_sec`	`rbd_qos_iops_limit`
`read_iops_sec`	`rbd_qos_read_iops_limit`
`write_iops_sec`	`rbd_qos_write_iops_limit`
`total_bytes_sec`	`rbd_qos_bps_limit`
`read_bytes_sec`	`rbd_qos_read_bps_limit`
`write_bytes_sec`	`rbd_qos_write_bps_limit`
`total_iops_sec_max`	`rbd_qos_bps_burst`
`read_iops_sec_max`	`rbd_qos_read_iops_burst`
`write_iops_sec_max`	`rbd_qos_write_iops_burst`
`total_bytes_sec_max`	`rbd_qos_bps_burst`
`read_bytes_sec_max`	`rbd_qos_read_bps_burst`
`write_bytes_sec_max`	`rbd_qos_write_bps_burst`

For more information on QoS settings you may refer to Ceph QoS documentation.

LVM

The default volume back end uses local volumes managed by LVM.

This driver supports different transport protocols to attach volumes, currently iSCSI and iSER.

Set the following in your `cinder.conf` configuration file, and use the following options to configure for iSCSI transport:

```
volume_driver = cinder.volume.drivers.lvm.LVMVolumeDriver
target_protocol = iscsi
```

Use the following options to configure for the iSER transport:

```
volume_driver = cinder.volume.drivers.lvm.LVMVolumeDriver
target_protocol = iser
```

Table 2: Description of LVM configuration options

Configuration option = Default value	Description
`lvm_conf_file` = `/etc/cinder/lvm.conf`	(String) LVM conf file to use for the LVM driver in Cinder; this setting is ignored if the specified file does not exist (You can also specify None to not use a conf file even if one exists).
`lvm_mirrors` = `0`	(Integer) If >0, create LVs with multiple mirrors. Note that this requires lvm_mirrors + 2 PVs with available space
`lvm_share_target` = `False`	(Boolean) Whether to share the same target for all LUNs or not (currently only supported by nvmet.
`lvm_suppress_fd_` = `False`	(Boolean) Suppress leaked file descriptor warnings in LVM commands.
`lvm_type` = `auto`	(String(choices=[default, thin, auto])) Type of LVM volumes to deploy; (default, thin, or auto). Auto defaults to thin if thin is supported.
`volume_group` = `cinder-volumes`	(String) Name for the VG that will contain exported volumes

Caution

When extending an existing volume which has a linked snapshot, the related logical volume is deactivated. This logical volume is automatically reactivated unless `auto_activation_volume_list` is defined in LVM configuration file `lvm.conf`. See the `lvm.conf` file for more information.

If auto activated volumes are restricted, then include the cinder volume group into this list:

```
auto_activation_volume_list = [ "existingVG", "cinder-volumes" ]
```

This note does not apply for thinly provisioned volumes because they do not need to be deactivated.

NFS driver

The Network File System (NFS) is a distributed file system protocol originally developed by Sun Microsystems in 1984. An NFS server `exports` one or more of its file systems, known as `shares`. An NFS client can mount these exported shares on its own file system. You can perform file actions on this mounted remote file system as if the file system were local.

How the NFS driver works

The NFS driver, and other drivers based on it, work quite differently than a traditional block storage driver.

The NFS driver does not actually allow an instance to access a storage device at the block level. Instead, files are created on an NFS share and mapped to instances, which emulates a block device. This works in a similar way to QEMU, which stores instances in the `/var/lib/nova/instances` directory.

Enable the NFS driver and related options

To use Cinder with the NFS driver, first set the `volume_driver` in the `cinder.conf` configuration file:

```
volume_driver=cinder.volume.drivers.nfs.NfsDriver
```

The following table contains the options supported by the NFS driver.

Table 3: Description of NFS storage configuration options

Configuration option = Default value	Description
`nfs_mount_attempts` = `3`	(Integer) The number of attempts to mount NFS shares before raising an error. At least one attempt will be made to mount an NFS share, regardless of the value specified.
`nfs_mount_options` = `None`	(String) Mount options passed to the NFS client. See the NFS(5) man page for details.
`nfs_mount_point_bas` = `$state_path/mnt`	(String) Base dir containing mount points for NFS shares.
`nfs_qcow2_volumes` = `False`	(Boolean) Create volumes as QCOW2 files rather than raw files.
`nfs_shares_config` = `/etc/cinder/nfs_shares`	(String) File with the list of available NFS shares.
`nfs_snapshot_suppor` = `False`	(Boolean) Enable support for snapshots on the NFS driver. Platforms using libvirt <1.2.7 will encounter issues with this feature.
`nfs_sparsed_volumes` = `True`	(Boolean) Create volumes as sparsed files which take no space. If set to False volume is created as regular file. In such case volume creation takes a lot of time.

Note

As of the Icehouse release, the NFS driver (and other drivers based off it) will attempt to mount shares using version 4.1 of the NFS protocol (including pNFS). If the mount attempt is unsuccessful due to a lack of client or server support, a subsequent mount attempt that requests the default behavior of the **`mount.nfs`** command will be performed. On most distributions, the default behavior is to attempt mounting first with NFS v4.0, then silently fall back to NFS v3.0 if necessary. If the `nfs_mount_options` configuration option contains a request for a specific version of NFS to be used, or if specific options are specified in the shares configuration file specified by the `nfs_shares_config` configuration option, the mount will be attempted as requested with no subsequent attempts.

How to use the NFS driver

Creating an NFS server is outside the scope of this document.

Configure with one NFS server

This example assumes access to the following NFS server and mount point:

- 192.168.1.200:/storage

This example demonstrates the usage of this driver with one NFS server.

Set the `nas_host` option to the IP address or host name of your NFS server, and the `nas_share_path` option to the NFS export path:

```
nas_host = 192.168.1.200
nas_share_path = /storage
```

Configure with multiple NFS servers

Note

You can use the multiple NFS servers with cinder multi back ends feature. Configure the *enabled_backends* option with multiple values, and use the `nas_host` and `nas_share` options for each back end as described above.

The below example is another method to use multiple NFS servers, and demonstrates the usage of this driver with multiple NFS servers. Multiple servers are not required. One is usually enough.

This example assumes access to the following NFS servers and mount points:

- 192.168.1.200:/storage
- 192.168.1.201:/storage
- 192.168.1.202:/storage

1. Add your list of NFS servers to the file you specified with the `nfs_shares_config` option. For example, if the value of this option was set to `/etc/cinder/shares.txt` file, then:

   ```
   # cat /etc/cinder/shares.txt
   192.168.1.200:/storage
   192.168.1.201:/storage
   192.168.1.202:/storage
   ```

   Comments are allowed in this file. They begin with a `#`.

2. Configure the `nfs_mount_point_base` option. This is a directory where `cinder-volume` mounts all NFS shares stored in the `shares.txt` file. For this example, `/var/lib/cinder/nfs` is used. You can, of course, use the default value of `$state_path/mnt`.

3. Start the `cinder-volume` service. `/var/lib/cinder/nfs` should now contain a directory for each NFS share specified in the `shares.txt` file. The name of each directory is a hashed name:

```
# ls /var/lib/cinder/nfs/
...
46c5db75dc3a3a50a10bfd1a456a9f3f
...
```

4. You can now create volumes as you normally would:

```
$ openstack volume create --size 5
# ls /var/lib/cinder/nfs/46c5db75dc3a3a50a10bfd1a456a9f3f
volume-a8862558-e6d6-4648-b5df-bb84f31c8935
```

This volume can also be attached and deleted just like other volumes.

NFS driver notes

- `cinder-volume` manages the mounting of the NFS shares as well as volume creation on the shares. Keep this in mind when planning your OpenStack architecture. If you have one master NFS server, it might make sense to only have one `cinder-volume` service to handle all requests to that NFS server. However, if that single server is unable to handle all requests, more than one `cinder-volume` service is needed as well as potentially more than one NFS server.
- Because data is stored in a file and not actually on a block storage device, you might not see the same IO performance as you would with a traditional block storage driver. Please test accordingly.
- Despite possible IO performance loss, having volume data stored in a file might be beneficial. For example, backing up volumes can be as easy as copying the volume files.

Note

Regular IO flushing and syncing still stands.

DataCore SANsymphony volume driver

DataCore SANsymphony volume driver provides OpenStack Compute instances with access to the SANsymphony(TM) Software-defined Storage Platform.

When volumes are created in OpenStack, the driver creates corresponding virtual disks in the SANsymphony server group. When a volume is attached to an instance in OpenStack, a Linux host is registered and the corresponding virtual disk is served to the host in the SANsymphony server group.

Requirements

- DataCore server group running SANsymphony software version 10 PSP6 or later.
- OpenStack Integration has been tested with the OpenStack environment installed on Ubuntu 20.04. For the list of qualified Linux host operating system types, refer to the Linux Host Configuration Guide on the DataCore Technical Support Web page.
- If using multipath I/O, ensure that iSCSI ports are logged in on all OpenStack Compute nodes. (All Fibre Channel ports will be logged in automatically.)

Python dependencies

- `websocket-client>=0.32.0`

  Install this package using pip:

  ```
  $ sudo pip install "websocket-client>=0.32.0"
  ```

Configuration

The volume driver can be configured by editing the `cinder.conf` file. The options below can be configured either per server group or as extra specifications in a volume type configuration.

Configuration options and default values:

- `datacore_disk_pools = None`

  Sets the pools to use for the DataCore OpenStack Cinder Volume Driver. This option acts like a filter and any number of pools may be specified. The list of specified pools will be used to select the storage sources needed for virtual disks; one for single or two for mirrored. Selection is based on the pools with the most free space.

  This option may also be specified as an extra specification of a volume type.

- `datacore_disk_type = single`

  Sets the SANsymphony virtual disk type (single or mirrored). **Single** virtual disks are created by default. Specify **mirrored** to override this behavior. Mirrored virtual disks require two DataCore Servers in the server group.

  This option may also be specified as an extra specification of a volume type.

- `datacore_storage_profile = Normal`

  Sets the storage profile of the virtual disk. The default setting is Normal. Other valid values include the standard storage profiles (Critical, High, Low, and Archive) and the names of custom profiles that have been created.

  This option may also be specified as an extra specification of a volume type.

- `datacore_api_timeout = 300`

  Sets the number of seconds to wait for a response from a DataCore API call.

  This option is used in the server group back-end configuration only.

- `datacore_disk_failed_delay = 300`

  Sets the number of seconds to wait for the SANsymphony virtual disk to come out of the Failed state.

  This option is used in the server group back-end configuration only.

- `datacore_iscsi_unallowed_targets = []`

  Sets a list of iSCSI targets that cannot be used to attach to the volume. By default, the DataCore iSCSI volume driver attaches a volume through all target ports with the Front-end role enabled, unlike the DataCore Fibre Channel volume driver that attaches a volume only through target ports connected to initiator.

To prevent the DataCore iSCSI volume driver from using some front-end targets in volume attachment, specify this option and list the iqn and target machine for each target as the value, such as `<iqn:target name>, <iqn:target name>, <iqn:target name>`. For example, `<iqn.2000-08.com.company:Server1-1, iqn.2000-08.com.company:Server2-1, iqn.2000-08.com.company:Server3-1>`.

This option is used in the server group back-end configuration only.

- `use_chap_auth = False`

  Sets the CHAP authentication for the iSCSI targets that are used to serve the volume. This option is disabled by default and will allow hosts (OpenStack Compute nodes) to connect to iSCSI storage back-ends without authentication. To enable CHAP authentication, which will prevent hosts (OpenStack Compute nodes) from connecting to back-ends without authentication, set this option to **True**.

  In addition, specify the location where the DataCore volume driver will store dynamically created CHAP secrets by setting the **datacore_iscsi_chap_storage** option.

  This option is used in the server group back-end configuration only. The driver will enable CHAP only for involved target ports, therefore, not all DataCore Servers may have CHAP configured. *Before enabling CHAP, ensure that there are no SANsymphony volumes attached to any instances.*

- `datacore_iscsi_chap_storage = /var/lib/cinder/.datacore_chap`

  Sets the path to the iSCSI CHAP authentication password storage file. **datacore_iscsi_chap_storage** is only used when **use_chap_auth = True** and **chap_password** is not set. Default **datacore_iscsi_chap_storage** value is $state_path/.datacore_chap.

  CHAP secrets are passed from OpenStack Block Storage to compute in clear text. This communication should be secured to ensure that CHAP secrets are not compromised. This can be done by setting up file permissions. Before changing the CHAP configuration, ensure that there are no SANsymphony volumes attached to any instances.

  This option is used in the server group back-end configuration only.

Configuration Examples

Examples of option configuration in the `cinder.conf` file.

- An example using **datacore_disk_pools**, **datacore_disk_type**, and **datacore_storage_profile** to create a mirrored virtual disk with a High priority storage profile using specific pools:

```
volume_driver = cinder.volume.drivers.datacore.iscsi.ISCSIVolumeDriver

san_ip = <DataCore Server IP or DNS name>

san_login = <User Name>

san_password = <Password>

datacore_disk_type = mirrored

datacore_disk_pools = Disk pool 1, Disk pool 2

datacore_storage_profile = High
```

- An example using **datacore_iscsi_unallowed_targets** to prevent the volume from using the specified targets:

```
volume_driver = cinder.volume.drivers.datacore.iscsi.ISCSIVolumeDriver

san_ip = <DataCore Server IP or DNS name>

san_login = <User Name>

san_password = <Password>

datacore_iscsi_unallowed_targets = iqn.2000-08.com.datacore:mns-ssv-10-1,
↪iqn.2000-08.com.datacore:mns-ssvdev-01-1
```

- An example using **use_chap_auth** and **chap_username** and **chap_password** to enable CHAP authentication:

```
volume_driver = cinder.volume.drivers.datacore.iscsi.ISCSIVolumeDriver

use_chap_auth = True

chap_username = user1

chap_password = user1_password
```

- An example using **use_chap_auth** and **datacore_iscsi_chap_storage** to enable CHAP authentication and provide the path to the CHAP password storage file:

```
volume_driver = cinder.volume.drivers.datacore.iscsi.ISCSIVolumeDriver

use_chap_auth = True

datacore_iscsi_chap_storage = /var/lib/cinder/.datacore_chap
```

DataCore volume driver stores CHAP secrets in clear text, and the password file must be secured by setting up file permissions. The following example shows how to create a password file and set up permissions. It assumes that the cinder-volume service is running under the user *cinder*. Please note that following steps are only required if the user wants to change the default **datacore_iscsi_chap_storage** location.

```
$ sudo mkdir /opt/user_dir/cinder -p

$ sudo /bin/sh -c "> /opt/user_dir/cinder/.datacore_chap"

$ sudo chown cinder:cinder /opt/user_dir/cinder

$ sudo chown cinder:cinder /opt/user_dir/cinder/.datacore_chap

$ sudo chmod -v 600 /opt/user_dir/cinder/.datacore_chap
```

CHAP will be enabled in SANsymphony after setting **use_chap_auth = True**. **chap_username** and **chap_password** will be used if mentioned, if not iSCSI initiator PortName will be used as chap_username with a random password, and the credentials will be stored in **datacore_iscsi_chap_storage** location.

Creating Volume Types

Volume types can be created with the DataCore disk type specified in the datacore:disk_type extra specification. In the following example, a volume type named mirrored_disk is created and the disk type is set to mirrored.

```
$ cinder type-create mirrored_disk

$ cinder type-key mirrored_disk set datacore:disk_type=mirrored
```

In addition, volume specifications can also be declared as extra specifications for volume types. The example below sets additional configuration options for the volume type mirrored_disk; storage profile will be set to High and virtual disks will be created from Disk pool 1, Disk pool 2, or Disk pool 3.

```
$ cinder type-key mirrored_disk set datacore:storage_profile=High

$ cinder type-key mirrored_disk set "datacore:disk_pools=Disk pool 1, Disk␣
↪pool 2, Disk pool 3"
```

Configuring Multiple Storage Back Ends

OpenStack Block Storage can be configured to use several back-end storage solutions. Multiple back-end configuration allows you to configure different storage configurations for SANsymphony server groups. The configuration options for a group must be defined in the group.

To enable multiple back ends:

1. In the `cinder.conf` file, set the **enabled_backends** option to identify the groups. One name is associated with each server group back-end configuration. In the example below there are two groups, `datacore-1` and `datacore-2`:

   ```
   [DEFAULT]

   enabled_backends = datacore-1, datacore-2
   ```

2. Define the back-end storage used by each server group in a separate section (for example `[datacore-1]`):

   ```
   [datacore-1]

   volume_driver = cinder.volume.drivers.datacore.iscsi.ISCSIVolumeDriver

   volume_backend_name = DataCore_iSCSI

   san_ip    = <ip_or_dns_name>

   san_login = <user_name>

   san_password = <password>
   ```

(continues on next page)

(continued from previous page)

```
use_chap_auth = True

chap_username = <chap_username>

chap_password = <chap_password>

datacore_iscsi_chap_storage = /var/lib/cinder/.datacore_chap

datacore_iscsi_unallowed_targets = iqn.2000-08.com.datacore:mns-ssv-10-1

datacore_disk_type = mirrored

[datacore-2]

volume_driver = cinder.volume.drivers.datacore.fc.FibreChannelVolumeDriver

volume_backend_name = DataCore_FibreChannel

san_ip   = <ip_or_dns_name>

san_login = <user_name>

san_password = <password>

datacore_disk_type = mirrored

datacore_disk_pools = Disk pool 1, Disk pool 2

datacore_storage_profile = High
```

3. Create the volume types

```
$ cinder type-create datacore_iscsi

$ cinder type-create datacore_fc
```

4. Add an extra specification to link the volume type to a back-end name:

```
$ cinder type-key datacore_iscsi set volume_backend_name=DataCore_iSCSI

$ cinder type-key datacore_fc set volume_backend_name=DataCore_
↪FibreChannel
```

See Configure multiple-storage back ends for additional information.

Detaching Volumes and Terminating Instances

Notes about the expected behavior of SANsymphony software when detaching volumes and terminating instances in OpenStack:

1. When a volume is detached from a host in OpenStack, the virtual disk will be unserved from the host in SANsymphony, but the virtual disk will not be deleted.
2. If all volumes are detached from a host in OpenStack, the host will remain registered and all virtual disks will be unserved from that host in SANsymphony. The virtual disks will not be deleted.
3. If an instance is terminated in OpenStack, the virtual disk for the instance will be unserved from the host and either be deleted or remain as unserved virtual disk depending on the option selected when terminating.

Support

In the event that a support bundle is needed, the administrator should save the files from the `/var/log` folder on the Linux host and attach to DataCore Technical Support incident manually.

Datera drivers

Datera iSCSI driver

The Datera Data Services Platform (DSP) is a scale-out storage software that turns standard, commodity hardware into a RESTful API-driven, intent-based policy controlled storage fabric for large-scale clouds. The Datera DSP integrates seamlessly with the Block Storage service. It provides storage through the iSCSI block protocol framework over the iSCSI block protocol. Datera supports all of the Block Storage services.

System requirements, prerequisites, and recommendations

Prerequisites

- All nodes must have access to Datera DSP through the iSCSI block protocol.
- All nodes accessing the Datera DSP must have the following packages installed:
  - Linux I/O (LIO)
  - open-iscsi
  - open-iscsi-utils
  - wget

Table 4: Description of Datera configuration options

Configuration option = Default value	Description
`datera_503` = `5`	(Integer) Interval between 503 retries
`datera_503` = `120`	(Integer) Timeout for HTTP 503 retry messages
`datera_deb` = `False`	(Boolean) True to set function arg and return logging
`datera_deb` = `False`	(Boolean) ONLY FOR DEBUG/TESTING PURPOSES True to set replica_count to 1
`datera_dis` = `False`	(Boolean) Set to True to disable sending additional metadata to the Datera backend
`datera_dis` = `False`	(Boolean) Set to True to disable profiling in the Datera driver
`datera_dis` = `False`	(Boolean) Set to True to disable automatic template override of the size attribute when creating from a template
`datera_ena` = `False`	(Boolean) Set to True to enable Datera backend image caching
`datera_ima` = `None`	(String) Cinder volume type id to use for cached volumes
`datera_lda` = `None`	(String) LDAP authentication server
`datera_ten` = `None`	(String) If set to Map > OpenStack project ID will be mapped implicitly to Datera tenant ID If set to None > Datera tenant ID will not be used during volume provisioning If set to anything else > Datera tenant ID will be the provided value
`datera_vol` = `{}`	(Dict of String) Settings here will be used as volume-type defaults if the volume-type setting is not provided. This can be used, for example, to set a very low total_iops_max value if none is specified in the volume-type to prevent accidental overusage. Options are specified via the following format, WITHOUT ANY DF: PREFIX: datera_volume_type_defaults=iops_per_gb:100,bandwidth_per_gb:200etc.
`datera_api` = `7717`	(String) Datera API port. **DEPRECATED**
`datera_api` = `2.2`	(String) Datera API version. **DEPRECATED**

Configuring the Datera volume driver

Modify the `/etc/cinder/cinder.conf` file for Block Storage service.

- Enable the Datera volume driver:

```
[DEFAULT]
# ...
enabled_backends = datera
# ...
```

- Optional. Designate Datera as the default back-end:

```
default_volume_type = datera
```

- Create a new section for the Datera back-end definition. The `VIP` can be either the Datera Management Network VIP or one of the Datera iSCSI Access Network VIPs depending on the network segregation requirements. For a complete list of parameters that can be configured, please see the section Volume Driver Cinder.conf Options

```
[datera]
volume_driver = cinder.volume.drivers.datera.datera_iscsi.DateraDriver
san_ip = <VIP>
san_login = admin
san_password = password
datera_tenant_id =
volume_backend_name = datera
datera_volume_type_defaults=replica_count:3
```

Enable the Datera volume driver

- Verify the OpenStack control node can reach the Datera `VIP`:

```
$ ping -c 4 <VIP>
```

- Start the Block Storage service on all nodes running the `cinder-volume` services:

```
$ service cinder-volume restart
```

Configuring one (or more) Datera specific volume types

There are extra volume type parameters that can be used to define Datera volume types with specific QoS policies (R/W IOPS, R/W bandwidth) and/or placement policies (replica count, type of media, IP pool to use, etc.)

For a full list of supported options please see the Volume-Type ExtraSpecs section in the driver documentation. See more examples in the Usage section.

```
# Create 2 replica volume type
$ openstack volume type create datera_2way --property volume_backend_
↪name=datera --property DF:replica_count=2

# Create volume type with limited write IOPS
$ openstack volume type create datera_iops --property volume_backend_
↪name=datera --property DF:write_iops_max=5000
```

Supported operations

- Create, delete, attach, detach, manage, unmanage, and list volumes.
- Create, list, and delete volume snapshots.
- Create a volume from a snapshot.
- Copy an image to a volume.

- Copy a volume to an image.
- Clone a volume.
- Extend a volume.
- Support for naming convention changes.

Configuring multipathing

Enabling multipathing is strongly reccomended for reliability and availability reasons. Please refer to the following file for an example of configuring multipathing in Linux 3.x kernels. Some parameters in different Linux distributions may be different.

Dell PowerFlex Storage driver

Overview

Dell PowerFlex (formerly named Dell ScaleIO/VxFlex OS) is a software-only solution that uses existing servers local disks and LAN to create a virtual SAN that has all of the benefits of external storage, but at a fraction of the cost and complexity. Using the driver, Block Storage hosts can connect to a PowerFlex Storage cluster.

The Dell PowerFlex Cinder driver is designed and tested to work with both PowerFlex and with ScaleIO. The *configuration options* are identical for both PowerFlex and ScaleIO.

Official PowerFlex documentation

To find the PowerFlex documentation:

1. Go to the PowerFlex product documentation page.
2. On the page, search for the relevant PowerFlex version.

Supported PowerFlex or VxFlex OS Versions

The Dell PowerFlex Block Storage driver has been tested against the following versions of VxFlex OS and PowerFlex and found to be compatible:

- PowerFlex 3.6.0
- PowerFlex 4.x

At the time of publication, the Dell PowerFlex Block Storage driver has been tested against the following version series of VxFlex OS and PowerFlex and found to be compatible: 3.6, 4.0, 4.5,and 4.6. You can find the current list of compatible versions at https://elabnavigator.dell.com/eln/elnhome.

Please consult the *Official PowerFlex documentation* to determine supported operating systems for each version of PowerFlex or VxFlex OS.

Deployment prerequisites

- The PowerFlex Gateway must be installed and accessible in the network. For installation steps, refer to the Preparing the installation Manager and the Gateway section in PowerFlex Deployment Guide. See *Official PowerFlex documentation*.
- PowerFlex Storage Data Client (SDC) must be installed on all OpenStack nodes.

Note

Ubuntu users must follow the specific instructions in the PowerFlex OS Deployment Guide for Ubuntu environments. See the `Deploying on Ubuntu Servers` section in PowerFlex Deployment Guide. See *Official PowerFlex documentation*.

Supported operations

- Create, delete, clone, attach, detach, migrate, manage, and unmanage volumes
- Create, delete, manage, and unmanage volume snapshots
- Create a volume from a snapshot
- Revert a volume to a snapshot
- Copy an image to a volume
- Copy a volume to an image
- Extend a volume
- Get volume statistics
- Create, list, update, and delete consistency groups
- Create, list, update, and delete consistency group snapshots
- OpenStack replication v2.1 support
- Cinder volume active/active support

PowerFlex Block Storage driver configuration

This section explains how to configure and connect the block storage nodes to a PowerFlex storage cluster.

Edit the `cinder.conf` file by adding the configuration below under a new section (for example, `[powerflex]`) and change the `enable_backends` setting (in the `[DEFAULT]` section) to include this new back end. The configuration file is usually located at `/etc/cinder/cinder.conf`.

For a configuration example, refer to the example *cinder.conf* .

PowerFlex driver name

Configure the driver name by adding the following parameter:

```
volume_driver = cinder.volume.drivers.dell_emc.powerflex.driver.
↪PowerFlexDriver
```

PowerFlex Gateway server IP

The PowerFlex Gateway provides a REST interface to PowerFlex.

Configure the Gateway server IP address by adding the following parameter:

```
san_ip = <PowerFlex GATEWAY IP>
```

PowerFlex Storage Pools

Multiple Storage Pools and Protection Domains can be listed for use by the virtual machines. The list should include every Protection Domain and Storage Pool pair that you would like Cinder to utilize.

To retrieve the available Storage Pools, use the command **scli --query_all** and search for available Storage Pools.

Configure the available Storage Pools by adding the following parameter:

```
powerflex_storage_pools = <Comma-separated list of protection domain:storage␣
↪pool name>
```

PowerFlex user credentials

Block Storage requires a PowerFlex user with administrative privileges. Dell recommends creating a dedicated OpenStack user account that has an administrative user role.

Refer to the PowerFlex User Guide for details on user account management.

Configure the user credentials by adding the following parameters:

```
san_login = <POWERFLEX_USER>
san_password = <POWERFLEX_PASSWD>
```

Oversubscription

Configure the oversubscription ratio by adding the following parameter under the separate section for PowerFlex:

```
powerflex_max_over_subscription_ratio = <OVER_SUBSCRIPTION_RATIO>
```

Note

The default value for `powerflex_max_over_subscription_ratio` is 10.0.

Oversubscription is calculated correctly by the Block Storage service only if the extra specification `provisioning:type` appears in the volume type regardless of the default provisioning type. Maximum oversubscription value supported for PowerFlex is 10.0.

Default provisioning type

If provisioning type settings are not specified in the volume type, the default value is set according to the `san_thin_provision` option in the configuration file. The default provisioning type will be `thin` if the option is not specified in the configuration file. To set the default provisioning type `thick`, set the `san_thin_provision` option to `false` in the configuration file, as follows:

```
san_thin_provision = false
```

The configuration file is usually located in `/etc/cinder/cinder.conf`. For a configuration example, see: *cinder.conf* .

Configuration example

cinder.conf example file

You can update the `cinder.conf` file by editing the necessary parameters as follows:

```
[DEFAULT]
enabled_backends = powerflex

[powerflex]
volume_driver = cinder.volume.drivers.dell_emc.powerflex.driver.
↪PowerFlexDriver
volume_backend_name = powerflex
san_ip = GATEWAY_IP
powerflex_storage_pools = Domain1:Pool1,Domain2:Pool2
san_login = POWERFLEX_USER
san_password = POWERFLEX_PASSWD
san_thin_provision = false
```

Connector configuration

Before using attach/detach volume operations PowerFlex connector must be properly configured. On each node where PowerFlex SDC is installed do the following:

1. Create `/opt/emc/scaleio/openstack/connector.conf` if it does not exist.

```
$ mkdir -p /opt/emc/scaleio/openstack
$ touch /opt/emc/scaleio/openstack/connector.conf
```

2. For each PowerFlex section in the `cinder.conf` create the same section in the `/opt/emc/scaleio/openstack/connector.conf` and populate it with passwords. Example:

```
[powerflex]
san_password = POWERFLEX_PASSWD
replicating_san_password = REPLICATION_SYSTEM_POWERFLEX_PASSWD # if␣
↪applicable

[powerflex-new]
san_password = SIO2_PASSWD
replicating_san_password = REPLICATION_SYSTEM_SIO2_PASSWD # if applicable
```

Configuration options

The PowerFlex driver supports these configuration options:

Table 5: Description of PowerFlex configuration options

Configuration option = Default value	Description
powerflex_allow_mig = False	(Boolean) Allow volume migration during rebuild.
powerflex_allow_non = False	(Boolean) Allow volumes to be created in Storage Pools when zero padding is disabled. This option should not be enabled if multiple tenants will utilize volumes from a shared Storage Pool.
powerflex_max_over_ = 10.0	(Float) max_over_subscription_ratio setting for the driver. Maximum value allowed is 10.0.
powerflex_rest_serv = 443	(Port(min=0, max=65535)) Gateway REST server port.
powerflex_round_vol = True	(Boolean) Round volume sizes up to 8GB boundaries. PowerFlex/VxFlex OS requires volumes to be sized in multiples of 8GB. If set to False, volume creation will fail for volumes not sized properly
powerflex_server_ap = None	(String) PowerFlex/ScaleIO API version. This value should be left as the default value unless otherwise instructed by technical support.
powerflex_storage_p = None	(String) Storage Pools. Comma separated list of storage pools used to provide volumes. Each pool should be specified as a protection_domain_name:storage_pool_name value
powerflex_unmap_vol = False	(Boolean) Unmap volumes before deletion.
rest_api_connect_ti = 30	(Integer(min=1)) Use this value to specify connect timeout value (in seconds) for rest call.
rest_api_read_timec = 30	(Integer(min=1)) Use this value to specify read timeout value (in seconds) for rest call.
vxflexos_allow_migr = False	(Boolean) renamed to powerflex_allow_migration_during_rebuild. **DEPRECATED**
vxflexos_allow_non_ = False	(Boolean) renamed to powerflex_allow_non_padded_volumes. **DEPRECATED**
vxflexos_max_over_s = 10.0	(Float) renamed to powerflex_max_over_subscription_ratio. **DEPRECATED**
vxflexos_rest_serve = 443	(Port(min=0, max=65535)) renamed to powerflex_rest_server_port. **DEPRECATED**
vxflexos_round_volu = True	(Boolean) renamed to powerflex_round_volume_capacity. **DEPRECATED**
vxflexos_server_api = None	(String) renamed to powerflex_server_api_version. **DEPRECATED**
vxflexos_storage_pc = None	(String) renamed to powerflex_storage_pools. **DEPRECATED**
vxflexos_unmap_volu = False	(Boolean) renamed to powerflex_round_volume_capacity. **DEPRECATED**

Volume Types

Volume types can be used to specify characteristics of volumes allocated via the PowerFlex Driver. These characteristics are defined as `Extra Specs` within `Volume Types`.

PowerFlex Protection Domain and Storage Pool

When multiple storage pools are specified in the Cinder configuration, users can specify which pool should be utilized by adding the `pool_name` Extra Spec to the volume type extra-specs and setting the value to the requested protection_domain:storage_pool.

```
$ openstack volume type create powerflex_type_1
$ openstack volume type set --property volume_backend_name=powerflex␣
↪powerflex_type_1
$ openstack volume type set --property pool_name=Domain2:Pool2 powerflex_type_
↪1
```

PowerFlex thin provisioning support

The Block Storage driver supports creation of thin-provisioned and thick-provisioned volumes. The provisioning type settings can be added as an extra specification of the volume type, as follows:

```
$ openstack volume type create powerflex_type_thick
$ openstack volume type set --property provisioning:type=thick powerflex_type_
↪thick
```

PowerFlex QoS support

QoS support for the PowerFlex driver includes the ability to set the following capabilities:

maxIOPS
The QoS I/O rate limit. If not set, the I/O rate will be unlimited. The setting must be larger than 10.

maxIOPSperGB
The QoS I/O rate limit. The limit will be calculated by the specified value multiplied by the volume size. The setting must be larger than 10.

maxBWS
The QoS I/O bandwidth rate limit in KBs. If not set, the I/O bandwidth rate will be unlimited. The setting must be a multiple of 1024.

maxBWSperGB
The QoS I/O bandwidth rate limit in KBs. The limit will be calculated by the specified value multiplied by the volume size. The setting must be a multiple of 1024.

The QoS keys above must be created and associated with a volume type. For example:

```
$ openstack volume qos create qos-limit-iops --consumer back-end --property␣
↪maxIOPS=5000
$ openstack volume type create powerflex_limit_iops
$ openstack volume qos associate qos-limit-iops powerflex_limit_iops
```

The driver always chooses the minimum between the QoS keys value and the relevant calculated value of `maxIOPSperGB` or `maxBWSperGB`.

Since the limits are per SDC, they will be applied after the volume is attached to an instance, and thus to a compute node/SDC.

PowerFlex compression support

Starting from version 3.0, PowerFlex supports volume compression. By default driver will create volumes without compression. In order to create a compressed volume, a volume type which enables compression support needs to be created first:

```
$ openstack volume type create powerflex_compressed
$ openstack volume type set --property provisioning:type=compressed powerflex_
↪compressed
```

If a volume with this type is scheduled to a storage pool which doesnt support compression, then `thin` provisioning will be used. See table below for details.

provisioning:type	storage pool supports compression	
	yes (PowerFlex 3.0 FG pool)	no (other pools)
compressed	thin with compression	thin
thin	thin	thin
thick	thin	thick
not set	thin	thin

Note

PowerFlex 3.0 Fine Granularity storage pools dont support thick provisioned volumes.

You can add property `compression_support='<is> True'` to volume type to limit volumes allocation only to data pools which supports compression.

```
$ openstack volume type set  --property compression_support='<is> True'␣
↪powerflex_compressed
```

PowerFlex replication support

Starting from version 3.5, PowerFlex supports volume replication.

Prerequisites

- PowerFlex replication components must be installed on source and destination systems.
- Source and destination systems must have the same configuration for Protection Domains and their Storage Pools (i.e. names, zero padding, etc.).
- Source and destination systems must be paired and have at least one Replication Consistency Group created.

See *Official PowerFlex documentation* for instructions.

Configure replication

1. Enable replication in `cinder.conf` file.

   To enable replication feature for storage backend `replication_device` must be set as below:

   ```
   [DEFAULT]
   enabled_backends = powerflex

   [powerflex]
   volume_driver = cinder.volume.drivers.dell_emc.powerflex.driver.PowerFlexDriver
   volume_backend_name = powerflex
   san_ip = GATEWAY_IP
   powerflex_storage_pools = Domain1:Pool1,Domain2:Pool2
   san_login = POWERFLEX_USER
   san_password = POWERFLEX_PASSWD
   san_thin_provision = false
   replication_device = backend_id:powerflex_repl,
                        san_ip: REPLICATION_SYSTEM_GATEWAY_IP,
                        san_login: REPLICATION_SYSTEM_POWERFLEX_USER,
                        san_password: REPLICATION_SYSTEM_POWERFLEX_PASSWD
   ```

   - Only one replication device is supported for storage backend.
   - The following parameters are optional for replication device:
     - REST API port - `powerflex_rest_server_port`.
     - SSL certificate verification - `driver_ssl_cert_verify` and `driver_ssl_cert_path`.

   For more information see *Configuration options*.

2. Create volume type for volumes with replication enabled.

   ```
   $ openstack volume type create powerflex_replicated
   $ openstack volume type set --property replication_enabled='<is> True' powerflex_replicated
   ```

3. Set PowerFlex Replication Consistency Group name for volume type.

   ```
   $ openstack volume type set --property powerflex:replication_cg=<replication_cg name> \
     powerflex_replicated
   ```

4. Set Protection Domain and Storage Pool if multiple Protection Domains are specified.

   PowerFlex Replication Consistency Group is created between source and destination Protection Domains. If more than one Protection Domain is specified in `cinder.conf` you should set `pool_name` property for volume type with appropriate Protection Domain and Storage Pool. See *PowerFlex Protection Domain and Storage Pool*.

Failover host

In the event of a disaster, or where there is a required downtime the administrator can issue the failover host command:

```
$ cinder failover-host cinder_host@powerflex --backend_id powerflex_repl
```

After issuing Cinder failover-host command Cinder will switch to configured replication device, however to get existing instances to use this target and new paths to volumes it is necessary to first shelve Nova instances and then unshelve them, this will effectively restart the Nova instance and re-establish data paths between Nova instances and the volumes.

```
$ nova shelve <server>
$ nova unshelve [--availability-zone <availability_zone>] <server>
```

If the primary system becomes available, the administrator can initiate failback operation using `--backend_id default`:

```
$ cinder failover-host cinder_host@powerflex --backend_id default
```

PowerFlex storage-assisted volume migration

Starting from version 3.0, PowerFlex supports storage-assisted volume migration.

Known limitations

- Migration between different backends is not supported.
- For migration from Medium Granularity (MG) to Fine Granularity (FG) storage pool zero padding must be enabled on the MG pool.
- For migration from MG to MG pool zero padding must be either enabled or disabled on both pools.

In the above cases host-assisted migration will be perfomed.

Migrate volume

Volume migration is performed by issuing the following command:

```
$ cinder migrate <volume> <host>
```

Note

Volume migration has a timeout of 3600 seconds (1 hour). It is done to prevent from endless waiting for migration to complete if something unexpected happened. If volume still is in migration after timeout has expired, volume status will be changed to `maintenance` to prevent future operations with this volume. The corresponding warning will be logged.

In this situation the status of the volume should be checked on the storage side. If volume migration succeeded, its status can be changed manually:

```
$ cinder reset-state --state available <volume>
```

Using PowerFlex Storage with a containerized overcloud

1. Create a file with below contents:

```
parameter_defaults:
  NovaComputeOptVolumes:
    - /opt/emc/scaleio:/opt/emc/scaleio
  CinderVolumeOptVolumes:
    - /opt/emc/scaleio:/opt/emc/scaleio
  GlanceApiOptVolumes:
    - /opt/emc/scaleio:/opt/emc/scaleio
```

   Name it whatever you like, e.g. `powerflex_volumes.yml`.

2. Use `-e` to include this customization file to deploy command.

3. Install the Storage Data Client (SDC) on all nodes after deploying the overcloud.

Dell PowerMax iSCSI and FC drivers

The Dell PowerMax drivers, `PowerMaxISCSIDriver` and `PowerMaxFCDriver`, support the use of Dell PowerMax and VMAX storage arrays with the Cinder Block Storage project. They both provide equivalent functions and differ only in support for their respective host attachment methods.

The drivers perform volume operations by communicating with the back-end PowerMax storage management software. They use the Requests HTTP library to communicate with a Unisphere for PowerMax instance, using a RESTAPI interface in the backend to perform PowerMax and VMAX storage operations.

Note

DEPRECATION NOTICE: The VMAX Hybrid series will not be supported from the `Z` release of OpenStack. Also, any All Flash array running HyperMaxOS 5977 will no longer be supported from the `Z` release onwards.

Note

While `PowerMax` will be used throughout this document, it will be used to collectively categorize the following supported arrays, PowerMax 2000, 8000, 2500, 8500, VMAX All Flash 250F, 450F, 850F and 950F and *VMAX-Hybrid*.

System requirements and licensing

The Dell PowerMax Cinder driver supports the *VMAX-Hybrid* series, VMAX All-Flash series and the PowerMax v3 and v4 arrays.

Download Solutions Enabler and Unisphere from the Dells support web site (login is required). See the *Dell Solutions Enabler Installation and Configuration Guide* and *Dell Unisphere for PowerMax Installation Guide* at the Dell Support site.

Note

At the time each OpenStack release, *support-matrix-table* was the recommended PowerMax management software and OS combinations. Please reach out your local PowerMax representative to see if these versions are still valid.

Starting with Antelope, the PowerMax OS version is now aligned with the Unisphere version scheme.

Table 6: PowerMax Management software and OS for OpenStack release

OpenStack release	Unisphere for PowerMax	PowerMax OS	Supported Arrays
Dalmatian	10.1.0	10.1.0 (6079.225)	PowerMax 2500,8500
		5978.711	PowerMax 2000,8000 VMAX 250F, 450F, 850F, 950F
Caracal	10.1.0	10.1.0 (6079.225)	PowerMax 2500,8500
		5978.711	PowerMax 2000,8000 VMAX 250F, 450F, 850F, 950F
Bobcat	10.0.1	10.0.1 (6079.175)	PowerMax 2500,8500
		5978.711	PowerMax 2000,8000 VMAX 250F, 450F, 850F, 950F
Antelope	10.0.1	10.0.1 (6079.175)	PowerMax 2500,8500
		5978.711	PowerMax 2000,8000 VMAX 250F, 450F, 850F, 950F
Zed	9.2.2	5978.711	PowerMax 2000,8000 VMAX 250F, 450F, 850F, 950F
Yoga	9.2.2	5978.711	PowerMax 2000,8000 VMAX 250F, 450F, 850F, 950F VMAX 100K, 200K, 400K (Hybrid)
Xena	9.2.2	5978.711	PowerMax 2000,8000 VMAX 250F, 450F, 850F, 950F VMAX 100K, 200K, 400K (Hybrid)
Wallaby	9.2.1	5978.711	PowerMax 2000,8000 VMAX 250F, 450F, 850F, 950F VMAX 100K, 200K, 400K (Hybrid)
Victoria	9.2.0	5978.669	PowerMax 2000,8000 VMAX 250F, 450F, 850F, 950F VMAX 100K, 200K, 400K (Hybrid)
Ussuri	9.1.x	5978.479	PowerMax 2000,8000 VMAX 250F, 450F, 850F, 950F VMAX 100K, 200K, 400K (Hybrid)
Train	9.1.x	5978.444	PowerMax 2000,8000 VMAX 250F, 450F, 850F, 950F VMAX 100K, 200K, 400K (Hybrid)
Stein	9.0.x	5978.221	PowerMax 2000,8000 VMAX 250F, 450F, 850F, 950F VMAX 100K, 200K, 400K (Hybrid)

Note

A Hybrid array can only run HyperMax OS 5977, and is still supported until the Z release of OpenStack. Some functionality will not be available in older versions of the OS. If in any doubt, please contact your local PowerMax representative.

Note

Newer versions of Unisphere for PowerMax and PowerMax OS are not retrospectively tested on older versions of OpenStack. If it is necessary to upgrade, the older REST endpoints will be used. For example, in Ussuri, if upgrading to Unisphere for PowerMax 9.2, the older 91 endpoints will be used.

Required PowerMax software suites for OpenStack

The storage system requires a Unisphere for PowerMax (SMC) eLicense.

PowerMax

There are two licenses for the PowerMax 2000 and 8000:

- Essentials software package
- Pro software package

The Dell PowerMax cinder driver requires the Pro software package.

All Flash

For full functionality including SRDF for the VMAX All Flash, the FX package, or the F package plus the SRDF a la carte add on is required.

Hybrid

There are five Dell Software Suites sold with the *VMAX-Hybrid* arrays:

- Base Suite
- Advanced Suite
- Local Replication Suite
- Remote Replication Suite
- Total Productivity Pack

The Dell PowerMax Cinder driver requires the Advanced Suite and the Local Replication Suite or the Total Productivity Pack (it includes the Advanced Suite and the Local Replication Suite) for the VMAX Hybrid.

Using PowerMax Remote Replication functionality will also require the Remote Replication Suite.

Note

Each are licensed separately. For further details on how to get the relevant license(s), reference eLicensing Support below.

eLicensing support

To activate your entitlements and obtain your PowerMax license files, visit the Service Center on Dell Support, as directed on your License Authorization Code (LAC) letter emailed to you.

- For help with missing or incorrect entitlements after activation (that is, expected functionality remains unavailable because it is not licensed), contact your EMC account representative or authorized reseller.
- For help with any errors applying license files through Solutions Enabler, contact the Dell Customer Support Center.
- If you are missing a LAC letter or require further instructions on activating your licenses through the Online Support site, contact EMCs worldwide Licensing team at `licensing@emc.com` or call:

  North America, Latin America, APJK, Australia, New Zealand: SVC4EMC (800-782-4362) and follow the voice prompts.

  EMEA: +353 (0) 21 4879862 and follow the voice prompts.

PowerMax for OpenStack Cinder customer support

If you require help or assistance with PowerMax and Cinder please open a Service Request (SR) through standard support channels at Dell Support. When opening a SR please include the following information:

- Array Model & uCode level
- Unisphere for PowerMax version
- Solutions Enabler Version
- OpenStack host Operating System (Ubuntu, RHEL, etc.)
- OpenStack version (Usurri, Train, etc.)
- PowerMax for Cinder driver version, this can be located in the comments in the PowerMax driver file: `{cinder_install_dir}/cinder/volume/drivers/dell_emc/powermax/fc.py`
- Cinder logs
- Detailed description of the issue you are encountering

Supported operations

PowerMax drivers support these operations:

- Create, list, delete, attach, and detach volumes
- Create, list, and delete volume snapshots
- Copy an image to a volume
- Copy a volume to an image
- Clone a volume
- Extend a volume
- Retype a volume (Host and storage assisted volume migration)
- Create a volume from a snapshot
- Create and delete generic volume group

- Create and delete generic volume group snapshot
- Modify generic volume group (add and remove volumes)
- Create generic volume group from source
- Live Migration
- Volume replication SRDF/S, SRDF/A and SRDF Metro
- Quality of service (QoS)
- Manage and unmanage volumes and snapshots
- List Manageable Volumes/Snapshots
- Backup create, delete, list, restore and show

PowerMax drivers also support the following features:

- Dynamic masking view creation
- Dynamic determination of the target iSCSI IP address
- iSCSI multipath support
- Oversubscription
- Service Level support
- SnapVX support
- Compression support(All Flash and PowerMax)
- Deduplication support(PowerMax)
- CHAP Authentication
- Multi-attach support
- Volume Metadata in logs
- Encrypted Volume support
- Extending attached volume
- Replicated volume retype support
- Retyping attached(in-use) volume
- Unisphere High Availability(HA) support
- Online device expansion of a metro device
- Rapid TDEV deallocation of deletes
- Multiple replication devices
- PowerMax array and storage group tagging
- Short host name and port group templates
- Snap id support
- Seamless Live Migration from SMI-S support
- Port group & port performance load balancing

- Cinder volume active/active support

Note

In certain cases, when creating a volume from a source snapshot or source volume, subsequent operations using the volumes may fail due to a missing snap_name exception. A manual refresh on the connected Unisphere instance or waiting until another operation automatically refreshes the connected Unisphere instance, will alleviate this issue.

PowerMax naming conventions

Note

`shortHostName` will be altered using the following formula, if its length exceeds 16 characters. This is because the storage group and masking view names cannot exceed 64 characters:

```
if len(shortHostName) > 16:
    1. Perform md5 hash on the shortHostName
    2. Convert output of 1. to hex
    3. Take last 6 characters of shortHostName and append output of 2.
    4. If the length of output of 3. exceeds 16 characters, join the
       first 8 characters and last 8 characters.
```

Note

`portgroup_name` will be altered using the following formula, if its length exceeds 12 characters. This is because the storage group and masking view names cannot exceed 64 characters:

```
if len(portgroup_name) > 12:
    1. Perform md5 hash on the portgroup_name
    2. Convert output of 1. to hex
    3. Take last 6 characters of portgroup_name and append output of 2.
    4. If the length of output of 3. exceeds 12 characters, join the
       first 6 characters and last 6 characters.
```

Masking view names

Masking views are dynamically created by the PowerMax FC and iSCSI drivers using the following naming conventions. `[protocol]` is either `I` for volumes attached over iSCSI or `F` for volumes attached over Fibre Channel.

```
OS-[shortHostName]-[protocol]-[portgroup_name]-MV
```

Initiator group names

For each host that is attached to PowerMax volumes using the drivers, an initiator group is created or re-used (per attachment type). All initiators of the appropriate type known for that host are included in the group. At each new attach volume operation, the PowerMax driver retrieves the initiators (either

WWNNs or IQNs) from OpenStack and adds or updates the contents of the Initiator Group as required. Names are of the following format. `[protocol]` is either `I` for volumes attached over iSCSI or `F` for volumes attached over Fibre Channel.

```
OS-[shortHostName]-[protocol]-IG
```

Note

Hosts attaching to OpenStack managed PowerMax storage cannot also attach to storage on the same PowerMax that are not managed by OpenStack.

FA port groups

PowerMax array FA ports to be used in a new masking view are retrieved from the port group provided as the extra spec on the volume type, or chosen from the list provided in the Dell configuration file.

Storage group names

As volumes are attached to a host, they are either added to an existing storage group (if it exists) or a new storage group is created and the volume is then added. Storage groups contain volumes created from a pool, attached to a single host, over a single connection type (iSCSI or FC). `[protocol]` is either `I` for volumes attached over iSCSI or `F` for volumes attached over Fibre Channel. PowerMax Cinder driver utilizes cascaded storage groups - a `parent` storage group which is associated with the masking view, which contains `child` storage groups for each configured SRP/slo/workload/compression-enabled or disabled/replication-enabled or disabled combination.

PowerMax, VMAX All Flash and *VMAX-Hybrid*

Parent storage group:

```
OS-[shortHostName]-[protocol]-[portgroup_name]-SG
```

Child storage groups:

```
OS-[shortHostName]-[SRP]-[ServiceLevel/Workload]-[portgroup_name]-CD-RE
```

Note

CD and RE are only set if compression is explicitly disabled or replication explicitly enabled. See the compression *11. All Flash compression support* and replication *Volume replication support* sections below.

Note

For VMAX All Flash with PowerMax OS (5978) or greater, workload if set will be ignored and set to NONE.

Table 7: Replication storage group naming conventions

Default storage group	Attached child storage group	Management Group	Replication Type
OS-[SRP]-[SL]-[WL]-SG	OS-[HOST]-[SRP]-[SL/WL]-[PG]	N/A	None
OS-[SRP]-[SL]-[WL]-RE-SG	OS-[HOST]-[SRP]-[SL/WL]-[PG]-RE	N/A	Synchronous
OS-[SRP]-[SL]-[WL]-RA-SG	OS-[HOST]-[SRP]-[SL/WL]-[PG]-RA	OS-[RDFG]-Asynchronous-rdf-sg	Asynchronous
OS-[SRP]-[SL]-[WL]-RM-SG	OS-[HOST]-[SRP]-[SL/WL]-[PG]-RM	OS-[RDFG]-Metro-rdf-sg	Metro

PowerMax driver integration

1. Prerequisites

1. Download Solutions Enabler from Dell Support and install it.

   You can install Solutions Enabler on a non-OpenStack host. Supported platforms include different flavors of Windows, Red Hat, and SUSE Linux. Solutions Enabler can be installed on a physical server, or as a Virtual Appliance (a VMware ESX server VM). Additionally, starting with HYPERMAX OS Q3 2015, you can manage VMAX3 arrays using the Embedded Management (eManagement) container application. See the `Dell Solutions Enabler 9.2.1 Installation and Configuration Guide` on Dell Support for more details.

   > **Note**
   >
   > You must discover storage arrays before you can use the PowerMax drivers. Follow instructions in `Dell Solutions Enabler 9.2.1 Installation and Configuration Guide` on Dell Support for more details.

2. Download Unisphere from Dell Support and install it.

   Unisphere can be installed in local, remote, or embedded configurations - i.e., on the same server running Solutions Enabler; on a server connected to the Solutions Enabler server; or using the eManagement container application (containing Solutions Enabler and Unisphere for PowerMax). See `Dell Solutions Enabler 9.2.1 Installation and Configuration Guide` at Dell Support.

3. Pay attention to the number of Gatekeepers device to have in your environment. It may vary depending on simultaneous call to Unisphere.

2. FC zoning with PowerMax

Zone Manager is required when there is a fabric between the host and array. This is necessary for larger configurations where pre-zoning would be too complex and open-zoning would raise security concerns.

3. iSCSI with PowerMax

- Make sure the `open-iscsi` package (or distro equivalent) is installed on all Compute nodes.

Note

You can only ping the PowerMax iSCSI target ports when there is a valid masking view. An attach operation creates this masking view.

4. Configure block storage in cinder.conf

Table 8: Description of PowerMax configuration options

Configuration option = Default value	Description
`initiator_check` = `False`	(Boolean) Use this value to enable the initiator_check.
`interval` = `3`	(Integer) Use this value to specify length of the interval in seconds.
`load_balance` = `False`	(Boolean) Enable/disable load balancing for a PowerMax backend.
`load_balance_real` = `False`	(Boolean) Enable/disable real-time performance metrics for Port level load balancing for a PowerMax backend.
`load_data_format` = `Avg`	(String) Performance data format, not applicable for real-time metrics. Available options are avg and max.
`load_look_back` = `60`	(Integer) How far in minutes to look back for diagnostic performance metrics in load calculation, minimum of 0 maximum of 1440 (24 hours).
`load_look_back_re` = `1`	(Integer) How far in minutes to look back for real-time performance metrics in load calculation, minimum of 1 maximum of 10.
`port_group_load_m` = `PercentBusy`	(String) Metric used for port group load calculation.
`port_load_metric` = `PercentBusy`	(String) Metric used for port load calculation.
`powermax_array` = `None`	(String) Serial number of the array to connect to.
`powermax_array_ta` = `None`	(List of String) List of user assigned name for storage array.
`powermax_port_gro` = `portGroupName`	(String) User defined override for port group name.
`powermax_port_gro` = `None`	(List of String) List of port groups containing frontend ports configured prior for server connection.
`powermax_service_` = `None`	(String) Service level to use for provisioning storage. Setting this as an extra spec in pool_name is preferable.
`powermax_short_ho` = `shortHostName`	(String) User defined override for short host name.
`powermax_srp` = `None`	(String) Storage resource pool on array to use for provisioning.
`rest_api_connect_` = `30`	(Integer(min=1)) Use this value to specify connect timeout value (in seconds) for rest call.
`rest_api_read_tim` = `30`	(Integer(min=1)) Use this value to specify read timeout value (in seconds) for rest call.
`retries` = `200`	(Integer) Use this value to specify number of retries.
`snapvx_unlink_sym` = `False`	(Boolean) Enable SnapVx unlink symforce, which forces the operation to execute when normally it is rejected.
`u4p_failover_auto` = `True`	(Boolean) If the driver should automatically failback to the primary instance of Unisphere when a successful connection is re-established.
`u4p_failover_back` = `1`	(Integer) A backoff factor to apply between attempts after the second try (most errors are resolved immediately by a second try without a delay). Retries will sleep for: {backoff factor} * (2 ^ ({number of total retries} - 1)) seconds.
`u4p_failover_retr` = `3`	(Integer) The maximum number of retries each connection should attempt. Note, this applies only to failed DNS lookups, socket connections and connection timeouts, never to requests where data has made it to the server.
`u4p_failover_targ` = `None`	(Dict of String) Dictionary of Unisphere failover target info.
`u4p_failover_time` = `20.0`	(Integer) How long to wait for the server to send data before giving up.

Note

san_api_port is 8443 by default but can be changed if necessary. For the purposes of this documentation the default is assumed so the tag will not appear in any of the cinder.conf extracts below.

Note

PowerMax PortGroups must be pre-configured to expose volumes managed by the array. Port groups can be supplied in cinder.conf, or can be specified as an extra spec storagetype:portgroupname on a volume type. If a port group is set on a volume type as an extra specification it takes precedence over any port groups set in cinder.conf. For more information on port and port group selection please see the section port group & port load balancing.

Note

PowerMax SRP cannot be changed once configured and in-use. SRP renaming on the PowerMax array is not supported.

Note

Service Level can be added to cinder.conf when the backend is the default case and there is no associated volume type. This not a recommended configuration as it is too restrictive. Workload is NONE for PowerMax and any All Flash with PowerMax OS (5978) or greater.

PowerMax parameter	cinder.conf parameter	Default	Required
ServiceLevel	powermax_service_level	None	No

To configure PowerMax block storage, add the following entries to /etc/cinder/cinder.conf:

```
enabled_backends = CONF_GROUP_ISCSI, CONF_GROUP_FC

[CONF_GROUP_ISCSI]
volume_driver = cinder.volume.drivers.dell_emc.powermax.iscsi.
↪PowerMaxISCSIDriver
volume_backend_name = POWERMAX_ISCSI
powermax_port_groups = [OS-ISCSI-PG]
san_ip = 10.10.10.10
san_login = my_username
san_password = my_password
powermax_array = 000123456789
powermax_srp = SRP_1


[CONF_GROUP_FC]
```

(continues on next page)

(continued from previous page)

```
volume_driver = cinder.volume.drivers.dell_emc.powermax.fc.PowerMaxFCDriver
volume_backend_name = POWERMAX_FC
powermax_port_groups = [OS-FC-PG]
san_ip = 10.10.10.10
san_login = my_username
san_password = my_password
powermax_array = 000123456789
powermax_srp = SRP_1
```

In this example, two back-end configuration groups are enabled: `CONF_GROUP_ISCSI` and `CONF_GROUP_FC`. Each configuration group has a section describing unique parameters for connections, drivers and the `volume_backend_name`.

5. SSL support

1. Get the CA certificate of the Unisphere server. This pulls the CA cert file and saves it as `.pem` file:

   ```
   # openssl s_client -showcerts \
                      -connect my_unisphere_host:8443 \
                      </dev/null 2>/dev/null \
                      | openssl x509 -outform PEM > my_unisphere_host.pem
   ```

   Where `my_unisphere_host` is the hostname of the unisphere instance and `my_unisphere_host.pem` is the name of the `.pem` file.

2. Add this path to `cinder.conf` under the PowerMax backend stanza and set SSL verify to `True`

   ```
   driver_ssl_cert_verify = True
   driver_ssl_cert_path = /path/to/my_unisphere_host.pem
   ```

   `OR` follow the steps 3-6 below if you would like to add the CA cert to the system certificate bundle instead of specifying the path to cert:

3. OPTIONAL: Copy the `.pem` cert to the system certificate directory and convert to `.crt`:

   ```
   # cp my_unisphere_host.pem /usr/share/ca-certificates/ca_cert.crt
   ```

4. OPTIONAL: Update CA certificate database with the following command. Ensure you select to enable the cert from step 3 when prompted:

   ```
   # sudo dpkg-reconfigure ca-certificates
   ```

5. OPTIONAL: Set a system environment variable to tell the Requests library to use the system cert bundle instead of the default Certifi bundle:

   ```
   # export REQUESTS_CA_BUNDLE = /etc/ssl/certs/ca-certificates.crt
   ```

6. OPTIONAL: Set cert verification to `True` under the PowerMax backend stanza in `cinder.conf`:

   ```
   # driver_ssl_cert_verify = True
   ```

7. Ensure `driver_ssl_cert_verify` is set to `True` in `cinder.conf` backend stanzas if steps 3-6 are followed, otherwise ensure both `driver_ssl_cert_path` and `driver_ssl_cert_verify` are set in `cinder.conf` backend stanzas.

6. Create volume types

Once `cinder.conf` has been updated, Openstack CLI commands need to be issued in order to create and associate OpenStack volume types with the declared `volume_backend_names`.

Additionally, each volume type will need an associated `pool_name` - an extra specification indicating the service level/ workload combination to be used for that volume type.

Note

The `pool_name` is an additional property which has to be set and is of the format: `<ServiceLevel>+<SRP>+<Array ID>`. This can be obtained from the output of the `cinder get-pools--detail`. Workload is NONE for PowerMax or any All Flash with PowerMax OS (5978) or greater.

There is also the option to assign a port group to a volume type by setting the `storagetype:portgroupname` extra specification.

```
$ openstack volume type create POWERMAX_ISCSI_SILVER
$ openstack volume type set --property volume_backend_name=ISCSI_backend \
                            --property pool_name=Silver+SRP_1+000123456789 \
                            --property storagetype:portgroupname=OS-PG2 \
                            POWERMAX_ISCSI_SILVER
$ openstack volume type create POWERMAX_FC_DIAMOND
$ openstack volume type set --property volume_backend_name=FC_backend \
                            --property pool_name=Gold+SRP_1+000123456789 \
                            --property storagetype:portgroupname=OS-PG1 \
                            POWERMAX_FC_GOLD
```

By issuing these commands, the Block Storage volume type `POWERMAX_ISCSI_SILVER` is associated with the `ISCSI_backend`, a Silver Service Level.

The type `POWERMAX_FC_DIAMOND` is associated with the `FC_backend`, a Diamond Service Level.

The `ServiceLevel` manages the underlying storage to provide expected performance. Setting the `ServiceLevel` to `None` means that non-FAST managed storage groups will be created instead (storage groups not associated with any service level).

```
openstack volume type set --property pool_name=None+SRP_1+000123456789
```

Note

PowerMax and *VMAX-Hybrid* support `Diamond`, `Platinum`, `Gold`, `Silver`, `Bronze`, `Optimized`, and `None` service levels. VMAX All Flash running HyperMax OS (5977) supports `Diamond` and `None`. *VMAX-Hybrid* and All Flash support `DSS_REP`, `DSS`, `OLTP_REP`, `OLTP`, and `None` workloads, the latter up until ucode 5977. Please refer to Stein PowerMax online documentation if you wish to use `workload`. There is no support for workloads in PowerMax OS (5978) or greater. These will be silently ignored if set for VMAX All-Flash arrays which have been upgraded to PowerMax OS (5988).

7. Interval and retries

By default, `interval` and `retries` are `3` seconds and `200` retries respectively. These determine how long (`interval`) and how many times (`retries`) a user is willing to wait for a single Rest call, `3*200=600seconds`. Depending on usage, these may need to be overridden by the user in `cinder.conf`. For example, if performance is a factor, then the `interval` should be decreased to check the job status more frequently, and if multiple concurrent provisioning requests are issued then `retries` should be increased so calls will not timeout prematurely.

In the example below, the driver checks every 3 seconds for the status of the job. It will continue checking for 200 retries before it times out.

Add the following lines to the PowerMax backend in `cinder.conf`:

```
[CONF_GROUP_ISCSI]
volume_driver = cinder.volume.drivers.dell_emc.powermax.iscsi.
↪PowerMaxISCSIDriver
volume_backend_name = POWERMAX_ISCSI
powermax_port_groups = [OS-ISCSI-PG]
san_ip = 10.10.10.10
san_login = my_username
san_password = my_password
powermax_array = 000123456789
powermax_srp = SRP_1
interval = 1
retries = 700
```

8. CHAP authentication support

This supports one-way initiator CHAP authentication functionality into the PowerMax backend. With CHAP one-way authentication, the storage array challenges the host during the initial link negotiation process and expects to receive a valid credential and CHAP secret in response. When challenged, the host transmits a CHAP credential and CHAP secret to the storage array. The storage array looks for this credential and CHAP secret which stored in the host initiators initiator group (IG) information in the ACLX database. Once a positive authentication occurs, the storage array sends an acceptance message to the host. However, if the storage array fails to find any record of the credential/secret pair, it sends a rejection message, and the link is closed.

Assumptions, restrictions and prerequisites

1. The host initiator IQN is required along with the credentials the host initiator will use to log into the storage array with. The same credentials should be used in a multi node system if connecting to the same array.
2. Enable one-way CHAP authentication for the iSCSI initiator on the storage array using `SYMCLI`. Template and example shown below. For the purpose of this setup, the credential/secret used would be `my_username/my_password` with iSCSI initiator of `iqn.1991-05.com.company.lcseb130`

```
# symaccess -sid <SymmID> -iscsi <iscsi> \
            {enable chap | disable chap | set chap} \
             -cred <Credential> -secret <Secret>

# symaccess -sid 128 \
            -iscsi iqn.1991-05.com.company.lcseb130 \
            set chap -cred my_username -secret my_password
```

Settings and configuration

1. Set the configuration in the PowerMax backend group in `cinder.conf` using the following parameters and restart cinder.

Configuration options	Value required for CHAP	Required for CHAP
`use_chap_auth`	`True`	Yes
`chap_username`	`my_username`	Yes
`chap_password`	`my_password`	Yes

```
[POWERMAX_ISCSI]
volume_driver = cinder.volume.drivers.dell_emc.powermax.iscsi.
↪PowerMaxISCSIDriver
volume_backend_name = POWERMAX_ISCSI
san_ip = 10.10.10.10
san_login = my_u4v_username
san_password = my_u4v_password
powermax_srp = SRP_1
powermax_array = 000123456789
powermax_port_groups = [OS-ISCSI-PG]
use_chap_auth = True
chap_username = my_username
chap_password = my_password
```

Usage

1. Using `SYMCLI`, enable CHAP authentication for a host initiator as described above, but do not set `use_chap_auth`, `chap_username` or `chap_password` in `cinder.conf`. Create a bootable volume.

```
openstack volume create --size 1 \
                        --image <image_name> \
                        --type <volume_type> \
                        test
```

2. Boot instance named `test_server` using the volume created above:

```
openstack server create --volume test \
                        --flavor m1.small \
                        --nic net-id=private \
                        test_server
```

3. Verify the volume operation succeeds but the boot instance fails as CHAP authentication fails.
4. Update `cinder.conf` with `use_chap_auth` set to `true` and `chap_username` and `chap_password` set with the correct credentials.
5. Rerun `openstack server create`
6. Verify that the boot instance operation ran correctly and the volume is accessible.
7. Verify that both the volume and boot instance operations ran successfully and the user is able to access the volume.

9. QoS (Quality of Service) support

Quality of service (QoS) has traditionally been associated with network bandwidth usage. Network administrators set limitations on certain networks in terms of bandwidth usage for clients. This enables them to provide a tiered level of service based on cost. The Nova/Cinder QoS offer similar functionality based on volume type setting limits on host storage bandwidth per service offering. Each volume type is tied to specific QoS attributes some of which are unique to each storage vendor. In the hypervisor, the QoS limits the following:

- Limit by throughput - Total bytes/sec, read bytes/sec, write bytes/sec
- Limit by IOPS - Total IOPS/sec, read IOPS/sec, write IOPS/sec

QoS enforcement in Cinder is done either at the hyper-visor (front-end), the storage subsystem (back-end), or both. This section focuses on QoS limits that are enforced by either the PowerMax backend and the hyper-visor front end interchangeably or just back end (Vendor Specific). The PowerMax driver offers support for Total bytes/sec limit in throughput and Total IOPS/sec limit of IOPS.

The PowerMax driver supports the following attributes that are front end/back end agnostic

- `total_iops_sec` - Maximum IOPs (in I/Os per second). Valid values range from 100 IO/Sec to 100000 IO/sec.
- `total_bytes_sec` - Maximum bandwidth (throughput) in bytes per second. Valid values range from 1048576 bytes (1MB) to 104857600000 bytes (100,000MB)

The PowerMax driver offers the following attribute that is vendor specific to the PowerMax and dependent on the `total_iops_sec` and/or `total_bytes_sec` being set.

- `Dynamic Distribution` - Enables/Disables dynamic distribution of host I/O limits. Possible values are:
  - `Always` - Enables full dynamic distribution mode. When enabled, the configured host I/O limits will be dynamically distributed across the configured ports, thereby allowing the limits on each individual port to adjust to fluctuating demand.
  - `OnFailure` - Enables port failure capability. When enabled, the fraction of configured host I/O limits available to a configured port will adjust based on the number of ports currently online.
  - `Never` - Disables this feature (Default).

USE CASE 1 - Default values

Prerequisites - PowerMax

- Host I/O Limit (MB/Sec) - No Limit
- Host I/O Limit (IO/Sec) - No Limit
- Set Dynamic Distribution - N/A

Table 9: **Prerequisites - Block Storage (Cinder) back-end (storage group)**

Key	Value
total_iops_sec	500
total_bytes_sec	104857600 (100MB)
DistributionType	Always

1. Create QoS Specs with the prerequisite values above:

```
$ openstack volume qos create --consumer back-end \
                              --property total_iops_sec=500 \
                              --property total_bytes_sec=104857600 \
                              --property DistributionType=Always \
                              my_qos
```

2. Associate QoS specs with specified volume type:

```
$ openstack volume qos associate my_qos my_volume_type
```

3. Create volume with the volume type indicated above:

```
$ openstack volume create --size 1 --type my_volume_type my_volume
```

Outcome - PowerMax (storage group)

- Host I/O Limit (MB/Sec) - 100
- Host I/O Limit (IO/Sec) - 500
- Set Dynamic Distribution - Always

Outcome - Block Storage (Cinder)

Volume is created against volume type and QoS is enforced with the parameters above.

USE CASE 2 - Pre-set limits

Prerequisites - PowerMax

- Host I/O Limit (MB/Sec) - 2000
- Host I/O Limit (IO/Sec) - 2000
- Set Dynamic Distribution - Never

Table 10: **Prerequisites - Block Storage (Cinder) back-end (storage group)**

Key	Value
total_iops_sec	500
total_bytes_sec	104857600 (100MB)
DistributionType	Always

1. Create QoS specifications with the prerequisite values above. The consumer in this use case is both for front-end and back-end:

```
$ openstack volume qos create --consumer back-end \
                              --property total_iops_sec=500 \
                              --property total_bytes_sec=104857600 \
                              --property DistributionType=Always \
                              my_qos
```

2. Associate QoS specifications with specified volume type:

```
$ openstack volume qos associate my_qos my_volume_type
```

3. Create volume with the volume type indicated above:

```
$ openstack volume create --size 1 --type my_volume_type my_volume
```

4. Attach the volume created in step 3 to an instance

```
$ openstack server add volume my_instance my_volume
```

Outcome - PowerMax (storage group)

- Host I/O Limit (MB/Sec) - 100
- Host I/O Limit (IO/Sec) - 500
- Set Dynamic Distribution - Always

Outcome - Block Storage (Cinder)

Volume is created against volume type and QoS is enforced with the parameters above.

Outcome - Hypervisor (Nova)

Libvirt includes an extra xml flag within the <disk> section called iotune that is responsible for rate limitation. To confirm that, first get the OS-EXT-SRV-ATTR:instance_name value of the server instance, for example instance-00000003.

```
$ openstack server show <serverid>

+-------------------------------------+-------------------------------------
↪-------------------------+
| Field                               | Value                               ␣
↪                         |
+-------------------------------------+-------------------------------------
↪-------------------------+
```

(continues on next page)

(continued from previous page)

```
| OS-DCF:diskConfig                    | AUTO                                      ␣
↪                                   |
| OS-EXT-AZ:availability_zone          | nova                                      ␣
↪                                   |
| OS-EXT-SRV-ATTR:host                 | myhost                                    ␣
↪                                   |
| OS-EXT-SRV-ATTR:hypervisor_hostname | myhost                                    ␣
↪                                   |
| OS-EXT-SRV-ATTR:instance_name        | instance-00000003                         ␣
↪                                   |
| OS-EXT-STS:power_state               | Running                                   ␣
↪                                   |
| OS-EXT-STS:task_state                | None                                      ␣
↪                                   |
| OS-EXT-STS:vm_state                  | active                                    ␣
↪                                   |
| OS-SRV-USG:launched_at               | 2017-11-02T08:15:42.000000                ␣
↪                                   |
| OS-SRV-USG:terminated_at             | None                                      ␣
↪                                   |
| accessIPv4                           |                                           ␣
↪                                   |
| accessIPv6                           |                                           ␣
↪                                   |
| addresses                            |␣
↪private=fd21:99c2:73f3:0:f816:3eff:febe:30ed, 10.0.0.3           |
| config_drive                         |                                           ␣
↪                                   |
| created                              | 2017-11-02T08:15:34Z                      ␣
↪                                   |
| flavor                               | m1.tiny (1)                               ␣
↪                                   |
| hostId                               |␣
↪e7b8312581f9fbb8508587d45c0b6fb4dc86102c632ed1f3a6a49d42         |
| id                                   | 0ef0ff4c-dbda-4dc7-b8ed-45d2fc2f31db ␣
↪                                   |
| image                                | cirros-0.3.5-x86_64-disk (b7c220f5-
↪2408-4296-9e58-fc5a41cb7e9d) |
| key_name                             | myhostname                                ␣
↪                                   |
| name                                 | myhostname                                ␣
↪                                   |
| progress                             | 0                                         ␣
↪                                   |
| project_id                           | bae4b97a0d8b42c28a5add483981e5db          ␣
↪                                   |
| properties                           |                                           ␣
↪                                   |
| security_groups                      | name='default'                            ␣
```

(continues on next page)

(continued from previous page)

```
↪                                |
| status                                | ACTIVE                                      ␣
↪                                |
| updated                               | 2017-11-02T08:15:42Z                        ␣
↪                                |
| user_id                               | 7bccf456740546799a7e20457f13c38b            ␣
↪                                |
| volumes_attached                      |                                             ␣
↪                                |
+---------------------------------------+---------------------------------------------
↪--------------------------+
```

We then run the following command using the `OS-EXT-SRV-ATTR:instance_name` retrieved above.

```
$ virsh dumpxml instance-00000003 | grep -1 "total_bytes_sec\|total_iops_sec"
```

The output of the command contains the XML below. It is found between the `<disk>` start and end tag.

```
<iotune>
   <total_bytes_sec>104857600</total_bytes_sec>
   <total_iops_sec>500</total_iops_sec>
</iotune>
```

USE CASE 3 - Pre-set limits

Prerequisites - PowerMax

- Host I/O Limit (MB/Sec) - `100`
- Host I/O Limit (IO/Sec) - `500`
- Set Dynamic Distribution - `Always`

Table 11: **Prerequisites - Block Storage (Cinder) back end (storage group)**

Key	Value
total_iops_sec	500
total_bytes_sec	104857600 (100MB)
DistributionType	OnFailure

1. Create QoS specifications with the prerequisite values above:

```
$ openstack volume qos create --consumer back-end \
                              --property total_iops_sec=500 \
                              --property total_bytes_sec=104857600 \
                              --property DistributionType=OnFailure \
                              my_qos
```

2. Associate QoS specifications with specified volume type:

```
$ openstack volume qos associate my_qos my_volume_type
```

3. Create volume with the volume type indicated above:

```
$ openstack volume create --size 1 --type my_volume_type my_volume
```

Outcome - PowerMax (storage group)

- Host I/O Limit (MB/Sec) - `100`
- Host I/O Limit (IO/Sec) - `500`
- Set Dynamic Distribution - `OnFailure`

Outcome - Block Storage (Cinder)

Volume is created against volume type and QOS is enforced with the parameters above.

USE CASE 4 - Default values

Prerequisites - PowerMax

- Host I/O Limit (MB/Sec) - `No Limit`
- Host I/O Limit (IO/Sec) - `No Limit`
- Set Dynamic Distribution - `N/A`

Table 12: **Prerequisites - Block Storage (Cinder) back end (storage group)**

Key	Value
DistributionType	Always

1. Create QoS specifications with the prerequisite values above:

```
$ openstack volume qos create --consumer back-end \
                              --property DistributionType=Always \
                              my_qos
```

2. Associate QoS specifications with specified volume type:

```
$ openstack volume qos associate my_qos my_volume_type
```

3. Create volume with the volume type indicated above:

```
$ openstack volume create --size 1 --type my_volume_type my_volume
```

Outcome - PowerMax (storage group)

- Host I/O Limit (MB/Sec) - `No Limit`
- Host I/O Limit (IO/Sec) - `No Limit`
- Set Dynamic Distribution - `N/A`

Outcome - Block Storage (Cinder)

Volume is created against volume type and there is no QoS change.

10. Multi-pathing support

- Install `open-iscsi` on all nodes on your system if on an iSCSI setup.
- Do not install EMC PowerPath as they cannot co-exist with native multi-path software
- Multi-path tools must be installed on all Nova compute nodes

On Ubuntu:

```
# apt-get install multipath-tools      #multipath modules
# apt-get install sysfsutils sg3-utils #file system utilities
# apt-get install scsitools            #SCSI tools
```

On openSUSE and SUSE Linux Enterprise Server:

```
# zipper install multipath-tools      #multipath modules
# zipper install sysfsutils sg3-utils #file system utilities
# zipper install scsitools            #SCSI tools
```

On Red Hat Enterprise Linux and CentOS:

```
# yum install iscsi-initiator-utils   #ensure iSCSI is installed
# yum install device-mapper-multipath #multipath modules
# yum install sysfsutils sg3-utils    #file system utilities
```

Multipath configuration file

The multi-path configuration file may be edited for better management and performance. Log in as a privileged user and make the following changes to `/etc/multipath.conf` on the Compute (Nova) node(s).

```
devices {
# Device attributed for EMC PowerMax
    device {
            vendor "EMC"
            product "SYMMETRIX"
            path_grouping_policy multibus
            getuid_callout "/lib/udev/scsi_id --page=pre-spc3-83 --
↪whitelisted --device=/dev/%n"
            path_selector "round-robin 0"
            path_checker tur
            features "0"
            hardware_handler "0"
            prio const
            rr_weight uniform
            no_path_retry 6
            rr_min_io 1000
            rr_min_io_rq 1
```

(continues on next page)

(continued from previous page)

```
    }
}
```

You may need to reboot the host after installing the MPIO tools or restart iSCSI and multi-path services.

On Ubuntu iSCSI:

```
# service open-iscsi restart
# service multipath-tools restart
```

On Ubuntu FC

```
# service multipath-tools restart
```

On openSUSE, SUSE Linux Enterprise Server, Red Hat Enterprise Linux, and CentOS iSCSI:

```
# systemctl restart open-iscsi
# systemctl restart multipath-tools
```

On openSUSE, SUSE Linux Enterprise Server, Red Hat Enterprise Linux, and CentOS FC:

```
# systemctl restart multipath-tools
```

```
$ lsblk
NAME                                          MAJ:MIN RM   SIZE RO TYPE ␣
↪MOUNTPOINT
sda                                             8:0    0     1G  0 disk
..360000970000196701868533030303235 (dm-6)  252:6    0     1G  0 mpath
sdb                                             8:16   0     1G  0 disk
..360000970000196701868533030303235 (dm-6)  252:6    0     1G  0 mpath
vda                                           253:0    0     1T  0 disk
```

OpenStack configurations

On Compute (Nova) node, add the following flag in the `[libvirt]` section of `nova.conf` and `nova-cpu.conf`:

```
volume_use_multipath = True
```

On Cinder controller node, multi-path for image transfer can be enabled in `cinder.conf` for each backend section or in `[backend_defaults]` section as a common configuration for all backends.

```
use_multipath_for_image_xfer = True
```

Restart `nova-compute` and `cinder-volume` services after the change.

Verify you have multiple initiators available on the compute node for I/O

1. Create a 3GB PowerMax volume.
2. Create an instance from image out of native LVM storage or from PowerMax storage, for example, from a bootable volume

3. Attach the 3GB volume to the new instance:

```
# multipath -ll
mpath102 (360000970000196700531533030383039) dm-3 EMC,SYMMETRIX
size=3G features='1 queue_if_no_path' hwhandler='0' wp=rw
'-+- policy='round-robin 0' prio=1 status=active
33:0:0:1 sdb 8:16 active ready running
'- 34:0:0:1 sdc 8:32 active ready running
```

4. Use the `lsblk` command to see the multi-path device:

```
# lsblk
NAME                                        MAJ:MIN RM   SIZE RO TYPE
sdb                                           8:0    0     3G  0 disk
..360000970000196700531533030383039 (dm-6) 252:6    0     3G  0 mpath
sdc                                           8:16   0     3G  0 disk
..360000970000196700531533030383039 (dm-6) 252:6    0     3G  0 mpath
vda
```

11. All Flash compression support

On an All Flash array, the creation of any storage group has a compressed attribute by default. Setting compression on a storage group does not mean that all the devices will be immediately compressed. It means that for all incoming writes compression will be considered. Setting compression `off` on a storage group does not mean that all the devices will be uncompressed. It means all the writes to compressed tracks will make these tracks uncompressed.

Note

This feature is only applicable for All Flash arrays, 250F, 450F, 850F and 950F and PowerMax 2000 and 8000. It was first introduced Solutions Enabler 8.3.0.11 or later and is enabled by default when associated with a Service Level. This means volumes added to any newly created storage groups will be compressed.

Use case 1 - Compression disabled create, attach, detach, and delete volume

1. Create a new volume type called `POWERMAX_COMPRESSION_DISABLED`.
2. Set an extra spec `volume_backend_name`.
3. Set a new extra spec `storagetype:disablecompression = True`.
4. Create a new volume.
5. Check in Unisphere or SYMCLI to see if the volume exists in storage group `OS-<srp>-<servicelevel>-<workload>-CD-SG`, and compression is disabled on that storage group.
6. Attach the volume to an instance. Check in Unisphere or SYMCLI to see if the volume exists in storage group `OS-<shorthostname>-<srp>-<servicelevel/workload>-<portgroup>-CD`, and compression is disabled on that storage group.
7. Detach volume from instance. Check in Unisphere or symcli to see if the volume exists in storage group `OS-<srp>-<servicelevel>-<workload>-CD-SG`, and compression is disabled on that

storage group.

8. Delete the volume. If this was the last volume in the `OS-<srp>-<servicelevel>-<workload>-CD-SG` storage group, it should also be deleted.

Use case 2 - Retype from compression disabled to compression enabled

1. Repeat steps 1-4 of Use case 1.
2. Create a new volume type. For example `POWERMAX_COMPRESSION_ENABLED`.
3. Set extra spec `volume_backend_name` as before.
4. Set the new extra specs compression as `storagetype:disablecompression = False` or DO NOT set this extra spec.
5. Retype from volume type `POWERMAX_COMPRESSION_DISABLED` to `POWERMAX_COMPRESSION_ENABLED`.
6. Check in Unisphere or symcli to see if the volume exists in storage group `OS-<srp>-<servicelevel>-<workload>-SG`, and compression is enabled on that storage group.

Note

If extra spec `storagetype:disablecompression` is set on a *VMAX-Hybrid*, it is ignored because compression is not an available feature on a *VMAX-Hybrid*.

12. Oversubscription support

Please refer to the official OpenStack over-subscription documentation for further information on using over-subscription with PowerMax.

13. Live migration support

Non-live migration (sometimes referred to simply as migration). The instance is shut down for a period of time to be moved to another hyper-visor. In this case, the instance recognizes that it was rebooted.

Live migration (or true live migration). Almost no instance downtime. Useful when the instances must be kept running during the migration. The different types of live migration are:

- **Shared storage-based live migration** Both hyper-visors have access to shared storage.
- **Block live migration** No shared storage is required. Incompatible with read-only devices such as CD-ROMs and Configuration Drive (config_drive).
- **Volume-backed live migration** Instances are backed by volumes rather than ephemeral disk. For PowerMax volume-backed live migration, shared storage is required.

The PowerMax driver supports shared volume-backed live migration.

Architecture

In PowerMax, A volume cannot belong to two or more FAST storage groups at the same time. To get around this limitation we leverage both cascaded storage groups and a temporary non-FAST storage group.

A volume can remain live if moved between masking views that have the same initiator group and port groups which preserves the host path.

During live migration, the following steps are performed by the PowerMax driver on the volume:

1. Within the originating masking view, the volume is moved from the FAST storage group to the non-FAST storage group within the parent storage group.
2. The volume is added to the FAST storage group within the destination parent storage group of the destination masking view. At this point the volume belongs to two storage groups.
3. One of two things happen:
   - If the connection to the destination instance is successful, the volume is removed from the non-FAST storage group in the originating masking view, deleting the storage group if it contains no other volumes.
   - If the connection to the destination instance fails, the volume is removed from the destination storage group, deleting the storage group, if empty. The volume is reverted back to the original storage group.

Live migration configuration

Please refer to the official OpenStack documentation on configuring migrations and live migration usage for more information.

> **Note**
>
> OpenStack Oslo uses an open standard for messaging middleware known as `AMQP`. This messaging middleware (the RPC messaging system) enables the OpenStack services that run on multiple servers to talk to each other. By default, the RPC messaging client is set to timeout after 60 seconds, meaning if any operation you perform takes longer than 60 seconds to complete the operation will timeout and fail with the ERROR message `Messaging Timeout: Timed out waiting for a reply to message ID [message_id]`
>
> If this occurs, increase the `rpc_response_timeout` flag value in `cinder.conf` and `nova.conf` on all Cinder and Nova nodes and restart the services.
>
> What to change this value to will depend entirely on your own environment, you might only need to increase it slightly, or if your environment is under heavy network load it could need a bit more time than normal. Fine tuning is required here, change the value and run intensive operations to determine if your timeout value matches your environment requirements.
>
> At a minimum please set `rpc_response_timeout` to `240`, but this will need to be raised if high concurrency is a factor. This should be sufficient for all Cinder backup commands also.

System configuration

`NOVA-INST-DIR/instances/` (for example, `/opt/stack/data/nova/instances`) has to be mounted by shared storage. Ensure that `NOVA-INST-DIR` (set with `state_path` in the `nova.conf` file) is the same on all hosts.

1. Configure your DNS or `/etc/hosts` and ensure it is consistent across all hosts. Make sure that the three hosts can perform name resolution with each other. As a test, use the ping command to ping each host from one another.

```
$ ping HostA
$ ping HostB
$ ping HostC
```

2. Export NOVA-INST-DIR/instances from HostA, and ensure it is readable and writable by the Compute user on HostB and HostC. Please refer to the relevant OS documentation for further details, for example Ubuntu NFS Documentation

3. On all compute nodes, enable the execute/search bit on your shared directory to allow qemu to be able to use the images within the directories. On all hosts, run the following command:

```
$ chmod o+x NOVA-INST-DIR/instances
```

Note

If migrating from compute to controller, make sure to run step two above on the controller node to export the instance directory.

Use case

For our use case shown below, we have three hosts with host names HostA, HostB and HostC. HostA is the controller node while HostB and HostC are the compute nodes. The following were also used in live migration.

- 2GB bootable volume using the CirrOS image.
- Instance created using the 2GB volume above with a flavor m1.small using 2048 RAM, 20GB of Disk and 1 VCPU.

1. Create a bootable volume.

```
$ openstack volume create --size 2 \
                          --image cirros-0.3.5-x86_64-disk \
                          --volume_lm_1
```

2. Launch an instance using the volume created above on HostB.

```
$ openstack server create --volume volume_lm_1 \
                          --flavor m1.small \
                          --nic net-id=private \
                          --security-group default \
                          --availability-zone nova:HostB \
                          server_lm_1
```

3. Confirm on HostB has the instance created by running:

```
$ openstack server show server_lm_1 | grep "hypervisor_hostname\|instance_
↪name"
  | OS-EXT-SRV-ATTR:hypervisor_hostname | HostB
  | OS-EXT-SRV-ATTR:instance_name | instance-00000006
```

4. Confirm, through virsh using the instance_name returned in step 3 (instance-00000006), on HostB that the instance is created using:

```
$ virsh list --all

Id   Name                      State
--------------------------------
1    instance-00000006         Running
```

5. Migrate the instance from HostB to HostA with:

```
$ openstack server migrate --os-compute-api-version 2.30 \
                           --live-migration --host HostA \
                           server_lm_1
```

6. Run the command on step 3 above when the instance is back in available status. The hypervisor should be on Host A.

7. Run the command on Step 4 on Host A to confirm that the instance is created through `virsh`.

14. Multi-attach support

PowerMax cinder driver supports the ability to attach a volume to multiple hosts/servers simultaneously. Please see the official OpenStack multi-attach documentation for configuration information.

Multi-attach architecture

In PowerMax, a volume cannot belong to two or more FAST storage groups at the same time. This can cause issues when we are attaching a volume to multiple instances on different hosts. To get around this limitation, we leverage both cascaded storage groups and non-FAST storage groups (i.e. a storage group with no service level, workload, or SRP specified).

Note

If no service level is assigned to the volume type, no extra work on the backend is required the volume is attached to and detached from each host as normal.

Example use case

Volume `Multi-attach-Vol-1` (with a multi-attach capable volume type, and associated with a Diamond Service Level) is attached to Instance `Multi-attach-Instance-A` on HostA. We then issue the command to attach `Multi-attach-Vol-1` to `Multi-attach-Instance-B` on HostB:

1. In the `HostA` masking view, the volume is moved from the FAST managed storage group to the non-FAST managed storage group within the parent storage group.

2. The volume is attached as normal on `HostB` i.e., it is added to a FAST managed storage group within the parent storage group of the `HostB` masking view. The volume now belongs to two masking views, and is exposed to both `HostA` and `HostB`.

We then decide to detach the volume from `Multi-attach-Instance-B` on `HostB`:

1. The volume is detached as normal from `HostB` i.e., it is removed from the FAST managed storage group within the parent storage group of the `HostB` masking view this includes cleanup of the associated elements if required. The volume now belongs to one masking view, and is no longer exposed to `HostB`.

2. In the HostA masking view, the volume is returned to the FAST managed storage group from the non-FAST managed storage group within the parent storage group. The non-FAST managed storage group is cleaned up, if required.

15. Volume encryption support

Encryption is supported through the use of OpenStack Barbican. Only front-end encryption is supported, back-end encryption is handled at the hardware level with Data at Rest Encryption (D@RE).

For further information on OpenStack Barbican including setup and configuration please refer to the following official Barbican documentation.

16. Volume metadata

Volume metadata is returned to the user in both the Cinder Volume logs and with volumes and snapshots created in Cinder via the UI or CLI.

16.1 Volume metadata in logs

If debug is enabled in the default section of `cinder.conf`, PowerMax Cinder driver will log additional volume information in the Cinder volume log, on each successful operation. The facilitates bridging the gap between OpenStack and the Array by tracing and describing the volume from a VMAX/ PowerMax view point.

```
+------------------------------------+----------------------------------------
↪-----------------+
| Key                                | Value                                  ␣
↪                 |
+------------------------------------+----------------------------------------
↪-----------------+
| service_level                      | Gold                                   ␣
↪                 |
| is_compression_disabled            | no                                     ␣
↪                 |
| powermax_cinder_driver_version     | 3.2.0                                  ␣
↪                 |
| identifier_name                    | OS-819470ab-a6d4-49cc-b4db-
↪6f85e82822b7                 |
| openstack_release                  | 13.0.0.0b3.dev3                        ␣
↪                 |
| volume_id                          | 819470ab-a6d4-49cc-b4db-6f85e82822b7   ␣
↪                 |
| storage_model                      | PowerMax_8000                          ␣
↪                 |
| successful_operation               | delete                                 ␣
↪                 |
| default_sg_name                    | OS-DEFAULT_SRP-Gold-NONE-SG            ␣
↪                 |
| device_id                          | 01C03                                  ␣
↪                 |
| unisphere_for_powermax_version     | V9.0.0.9                               ␣
↪                 |
```

(continues on next page)

(continued from previous page)

```
| workload                       | NONE                                      ␣
↪                 |
| openstack_version              | 13.0.0                                    ␣
↪                 |
| volume_updated_time            | 2018-08-03 03:13:53                       ␣
↪                 |
| platform                       | Linux-4.4.0-127-generic-x86_64-with-
↪Ubuntu-16.04-xenial |
| python_version                 | 2.7.12                                    ␣
↪                 |
| volume_size                    | 20                                        ␣
↪                 |
| srp                            | DEFAULT_SRP                               ␣
↪                 |
| openstack_name                 | 90_Test_Vol56                             ␣
↪                 |
| storage_firmware_version       | 5978.143.144                              ␣
↪                 |
| serial_number                  | 000123456789                              ␣
↪                 |
+--------------------------------+---------------------------------------------
↪-----------------+
```

16.2 Metadata in the UI and CLI

By default metadata will be set on all volume and snapshot objects created in Cinder. This information represents the state of the object on the backend PowerMax and will change when volume attributes are changed by performing actions on them such as re-type or attaching to an instance.

```
demo@openstack-controller:~$ cinder show powermax-volume

+--------------------------------+---------------------------------------------
↪----------------+
| Property                       | Value                                     ␣
↪                |
+--------------------------------+---------------------------------------------
↪----------------+
| metadata                       | ArrayID : 000123456789                    ␣
↪                |
|                                | ArrayModel : PowerMax_8000                ␣
↪                |
|                                | CompressionDisabled : False               ␣
↪                |
|                                | Configuration : TDEV                      ␣
↪                |
|                                | DeviceID : 0012F                          ␣
↪                |
|                                | DeviceLabel : OS-d87edb98-60fd-49dd-bb0f-
↪cc388cf6f3f4      |
```

(continues on next page)

(continued from previous page)

```
|                              | Emulation : FBA                           ␣
↪                |
|                              | ReplicationEnabled : False                ␣
↪                |
|                              | ServiceLevel : Diamond                    ␣
↪                |
|                              | Workload : None                           ␣
↪                |
| name                         | powermax-volume                           ␣
↪                |
+------------------------------+-----------------------------------------
↪----------------+
```

17. Unisphere High Availability (HA) support

This feature facilitates high availability of Unisphere for PowerMax servers, allowing for one or more backup unisphere instances in the event of a loss in connection to the primary Unisphere instance. The PowerMax driver will cycle through the list of failover instances, trying each until a successful connection is made. The ordering is first in, first out (FIFO), so the first `u4p_failover_target` specified in `cinder.conf` will be the first selected, the second `u4p_failover_target` in `cinder.conf` will be the second selected, and so on until all failover targets are exhausted.

Requirements

- All required instances of Unisphere for PowerMax are set up and configured for the array(s)
- Array(s) are locally registered with the instance of Unisphere that will be used as a failover instance. There are two failover types, local and remote:
  - *Local failover* - Primary Unisphere is unreachable, failover to secondary local instance of Unisphere to resume normal operations at primary site.
  - *Remote failover* - Complete loss of primary site so primary instance of Unisphere is unreachable, failover to secondary instance of Unisphere at remote site to resume operations with the R2 array.

Note

Replication must be configured in advance for remote failover to work successfully. Human intervention will also be required to failover from R1 array to R2 array in Cinder using `cinder failover-host` command (see *Volume replication support* for replication setup details).

Note

The remote target array must be registered as local to the remote instance of Unisphere

Configuration

The following configuration changes need to be made in `cinder.conf` under the PowerMax backend stanza in order to support the failover to secondary Unisphere. Cinder services will need to be restarted for changes to take effect.

```
[POWERMAX_1]
...

u4p_failover_timeout = 30
u4p_failover_retries = 3
u4p_failover_backoff_factor = 1
u4p_failover_autofailback = True
u4p_failover_target = san_ip:10.10.10.12,
                      san_api_port: 8443,
                      san_login:my_username,
                      san_password:my_password,
                      driver_ssl_cert_verify: False,
u4p_failover_target = san_ip:10.10.10.13,
                      san_api_port: 8443
                      san_login:my_username,
                      san_password:my_password,
                      driver_ssl_cert_verify: True,
                      driver_ssl_cert_path: /path/to/my_unisphere_host.pem
```

Note

`u4p_failover_target` key value pairs will need to be on the same line (separated by commas) in `cinder.conf`. They are displayed on separated lines above for readability.

Note

To add more than one Unisphere failover target create additional `u4p_failover_target` details for the Unisphere instance. These will be cycled through in a first-in, first-out (FIFO) basis, the first failover target in `cinder.conf` will be the first backup instance of Unisphere used by the PowerMax driver.

18. Rapid TDEV deallocation

The PowerMax driver can now leverage the enhanced volume delete feature-set made available in the PowerMax 5978 Foxtail uCode release. These enhancements allow volume deallocation & deletion to be combined into a single call. Previously, volume deallocation & deletion were split into separate tasks; now a single REST call is dispatched and a response code on the projected outcome of their request is issued rapidly allowing other task execution to proceed without the delay. No additional configuration is necessary, the system will automatically determine when to use either the rapid or legacy compliant volume deletion sequence based on the connected PowerMax arrays metadata.

19. PowerMax online (in-use) device expansion

uCode Level		Supported In-Use Volume Extend Operations		
R1 uCode Level	R2 uCode Level	Sync	Async	Metro
5978.711	5978.711	Y	Y	Y
5978.711	5978.669	Y	Y	Y
5978.711	5978.444	Y	Y	Y
5978.711	5978.221	Y	Y	N
5978.669	5978.669	Y	Y	Y
5978.669	5978.444	Y	Y	Y
5978.669	5978.221	Y	Y	N
5978.444	5978.444	Y	Y	Y
5978.444	5978.221	Y	Y	N
5978.221	5978.221	Y	Y	N

Assumptions, restrictions and prerequisites

- ODE in the context of this document refers to extending a volume where it is in-use, that is, attached to an instance.
- The `allow_extend` is only applicable on *VMAX-Hybrid* arrays or All Flash arrays with HyperMax OS. If included elsewhere, it is ignored.
- Where one array is a lower uCode than the other, the environment is limited to functionality of that of the lowest uCode level, i.e. if R1 is 5978.444 and R2 is 5978.221, expanding a metro volume is not supported, both R1 and R2 need to be on 5978.444 uCode at a minimum.

20. PowerMax array and storage group tagging

Unisphere for PowerMax 9.1 and later supports tagging of storage groups and arrays, so the user can give their own tag for ease of searching and/or grouping.

Assumptions, restrictions and prerequisites

- The storage group tag(s) is associated with a volume type extra spec key `storagetype:storagegrouptags`.
- The array tag is associated with the backend stanza using key `powermax_array_tag_list`. It expects a list of one or more comma separated values, for example `powermax_array_tag_list=[value1,value2, value3]`
- They can be one or more values in a comma separated list.
- There is a 64 characters limit of letters, numbers, - and _.
- 8 tags are allowed per storage group and array.
- Tags cannot be modified once a volume has been created with that volume type. This is an OpenStack constraint.
- Tags can be modified on the backend stanza, but none will ever be removed, only added.
- There is no restriction on creating or deleting tags of OpenStack storage groups or arrays outside of OpenStack, for example Unisphere for PowerMax UI. The max number of 8 tags will apply

however, as this is a Unisphere for PowerMax limit.

Set a storage group tag on a volume type:

```
$ openstack volume type set --property␣
↪storagetype:storagegrouptags=myStorageGroupTag1,myStorageGroupTag2
```

Set an array tag on the PowerMax backend:

```
[POWERMAX_ISCSI]
volume_driver = cinder.volume.drivers.dell_emc.powermax.iscsi.
↪PowerMaxISCSIDriver
volume_backend_name = POWERMAX_ISCSI
san_ip = 10.10.10.10
san_login = my_u4v_username
san_password = my_u4v_password
powermax_srp = SRP_1
powermax_array = 000123456789
powermax_port_groups = [OS-ISCSI-PG]
powermax_array_tag_list = [openstack1, openstack2]
```

21. PowerMax short host name and port group name override

This functionality allows the user to customize the short host name and port group name that are contained in the PowerMax driver storage groups and masking views names. For current functionality please refer to *PowerMax naming conventions* for more details.

As the storage group name and masking view name are limited to 64 characters the short host name needs to be truncated to 16 characters or less and port group needs to be truncated to 12 characters or less. This functionality offers a little bit more flexibility to determine how these truncated components should look.

Note: Once the port group and short host name have been overridden with any new format, it is not possible to return to the default format or change to another format if any volumes are in an attached state. This is because there is no way to determine the overridden format once `powermax_short_host_name_template` or ``powermax_port_group_name_template` have been removed or changed.

Assumptions, restrictions, and prerequisites

- Backward compatibility with old format is preserved.
- `cinder.conf` will have 2 new configuration options, `short_host_name_template` and `port_group_name_template`.
- If a storage group, masking view or initiator group in the old naming convention already exists, this remains and any new attaches will use the new naming convention where the label for the short host name and/or port group has been customized by the user.
- Only the short host name and port group name components can be renamed within the storage group, initiator group and masking view names.

- If the `powermax_short_host_name_template` and `powermax_port_group_name_template` do not adhere to the rules, then the operation will fail early and gracefully with a clear description as to the problem.
- The templates cannot be changed once volumes have been attached using the new configuration.
- If only one of the templates are configured, then the other will revert to the default option.
- The UUID is generated from the MD5 hash of the full short host name and port group name
- If `userdef` is used, the onus is on the user to make sure it will be unique among all short host names (controller and compute nodes) and unique among port groups.

Table 13: Short host name templates

powermax_short_host_na	Description	Rule
shortHostName	This is the default option	Existing functionality, if over 16 characters then see *PowerMax naming conventions*, otherwise short host name
shortHostName[:x])uuid[:x] e.g. shortHostName[:6]uuid[:9]	First x characters of the short host name and x uuid characters created from md5 hash of short host name	Must be less than 16 characters
shortHostName[:x]userdef e.g. shortHostName[:6]-testHost	First x characters of the short host name and a user defined x char name. NB - the responsibility is on the user for uniqueness	Must be less than 16 characters
shortHostName[-x:]uuid[:x] e.g. shortHostName[-6:]uuid[:9]	Last x characters of the short host name and x uuid characters created from md5 hash of short host name	Must be less than 16 characters
shortHostName[-x:]userdef e.g. shortHostName[-6:]-testHost	Last x characters of the short host name and a user defined x char name. NB - the responsibility is on the user for uniqueness	Must be less than 16 characters

Table 14: Port group name templates

powermax_port_group_n…	Description	Rule
portGroupName	This is the default option	Existing functionality, if over 12 characters then see *PowerMax naming conventions*, otherwise port group name
portGroupName[:x])uuid[:x] e.g. portGroupName[:6]uuid[:5]	First x characters of the port group name and x uuid characters created from md5 hash of port group name	Must be less than 12 characters
portGroupName[:x]userdef e.g. portGroupName[:6]-test	First x characters of the port group name and a user defined x char name. NB - the responsibility is on the user for uniqueness	Must be less than 12 characters
portGroupName[-x:]uuid[:x] e.g. portGroupName[-6:]uuid[:5]	Last x characters of the port group name and x uuid characters created from md5 hash of port group name	Must be less than 12 characters
portGroupName[-x:]userdef e.g. portGroupName[-6:]-test	Last x characters of the port group name and a user defined x char name. NB - the responsibility is on the user for uniqueness	Must be less than 12 characters

21. Snap ids replacing generations

Snap ids were introduced to the PowerMax in microcde 5978.669.669 and Unisphere for PowerMax 9.2. Generations existed previously and could cause stale data if deleted out of sequence, even though we locked against this occurence. This happened when the newer generation(s) inherited its deleted predecessors generation number. So in a series of 0, 1, 2 and 3 generations, if generation 1 gets deleted, generation 2 now becomes generation 1 and generation 3 becomes generation 2 and so on down the line. Snap ids are unique to each snapVX and will not change once assigned at creation so out of sequence deletions are no longer an issue. Generations will remain for arrays with microcode less than 5978.669.669.

Cinder supported operations

Volume replication support

Note

A mix of `RDF1+TDEV` and `TDEV` volumes should not exist in the same storage group. This can happen on a cleanup operation after breaking the pair and a TDEV remains in the storage group on either the local or remote array. If this happens, remove the volume from the storage group so that further replicated volume operations can continue. For example, Remove `TDEV` from `OS-[SRP]-[SL]-[WL]-RA-SG`.

Note

Replication storage groups should exist on both local and remote array but never on just one. For example, if OS-[SRP]-[SL]-[WL]-RA-SG exists on local array A it must also exist on remote array B. If this condition does not hold, further replication operations will fail. This applies to management storage groups in the case of `Asynchronous` and `Metro` modes also. See *Replication storage group naming conventions*.

Note

The number of devices in replication storage groups in both local and remote arrays should be same. This also applies to management storage groups in `Asynchronous` and `Metro` modes. See *Replication storage group naming conventions*.

Configure a single replication target

1. Configure an SRDF group between the chosen source and target arrays for the PowerMax Cinder driver to use. The source array must correspond with the `powermax_array` entry in `cinder.conf`.
2. Select both the director and the ports for the SRDF emulation to use on both sides. Bear in mind that network topology is important when choosing director endpoints. Supported modes are `Synchronous`, `Asynchronous`, and `Metro`.

   Note

   If the source and target arrays are not managed by the same Unisphere server (that is, the target array is remotely connected to server - for example, if you are using embedded management), in the event of a full disaster scenario (i.e. the primary array is completely lost and all connectivity to it is gone), the Unisphere server would no longer be able to contact the target array. In this scenario, the volumes would be automatically failed over to the target array, but administrator intervention would be required to either; configure the target (remote) array as local to the current Unisphere server (if it is a stand-alone server), or enter the details of a second Unisphere server to the `cinder.conf`, which is locally connected to the target array (for example, the embedded management Unisphere server of the target array), and restart the Cinder volume service.

   Note

   If you are setting up an SRDF/Metro configuration, it is recommended that you configure a Witness or vWitness for bias management. Please see the SRDF Metro Overview & Best Practices guide for more information.

   Note

   The PowerMax Cinder drivers do not support Cascaded SRDF.

Note

The transmit idle functionality must be disabled on the R2 array for Asynchronous rdf groups. If this is not disabled it will prevent failover promotion in the event of access to the R1 array being lost.

```
# symrdf -sid <sid> -rdfg <rdfg> set rdfa -transmit_idle off
```

Note

When creating RDF enabled volumes, if there are existing volumes in the target storage group, all rdf pairs related to that storage group must have the same rdf state i.e. rdf pair states must be consistent across all volumes in a storage group when attempting to create a new replication enabled volume. If mixed rdf pair states are found during a volume creation attempt, an error will be raised by the rdf state validation checks. In this event, please wait until all volumes in the storage group have reached a consistent state.

3. Enable replication in `/etc/cinder/cinder.conf`. To enable the replication functionality in PowerMax Cinder driver, it is necessary to create a replication volume-type. The corresponding back-end stanza in `cinder.conf` for this volume-type must then include a `replication_device` parameter. This parameter defines a single replication target array and takes the form of a list of key value pairs.

```
enabled_backends = POWERMAX_FC_REPLICATION
[POWERMAX_FC_REPLICATION]
volume_driver = cinder.volume.drivers.dell_emc.powermax.fc.
↪PowerMaxFCDriver
san_ip = 10.10.10.10
san_login = my_u4v_username
san_password = my_u4v_password
powermax_srp = SRP_1
powermax_array = 000123456789
powermax_port_groups = [OS-FC-PG]
volume_backend_name = POWERMAX_FC_REPLICATION
replication_device = target_device_id:000197811111,
                     remote_port_group:os-failover-pg,
                     remote_pool:SRP_1,
                     rdf_group_label: 28_11_07,
                     mode:Metro,
                     metro_use_bias:False,
                     sync_interval:3,
                     sync_retries:200
```

Note

`replication_device` key value pairs will need to be on the same line (separated by commas) in `cinder.conf`. They are displayed here on separate lines above for improved readability.

- `target_device_id` The unique PowerMax array serial number of the target array. For full failover functionality, the source and target PowerMax arrays must be discovered and managed by the same U4V server.

- `remote_port_group` The name of a PowerMax port group that has been pre-configured to expose volumes managed by this backend in the event of a failover. Make sure that this port group contains either all FC or all iSCSI port groups (for a given back end), as appropriate for the configured driver (iSCSI or FC).
- `remote_pool` The unique pool name for the given target array.
- `rdf_group_label` The name of a PowerMax SRDF group that has been pre-configured between the source and target arrays.
- `mode` The SRDF replication mode. Options are `Synchronous`, `Asynchronous`, and `Metro`. This defaults to `Synchronous` if not set.
- `metro_use_bias` Flag to indicate if bias protection should be used instead of Witness. This defaults to False.
- `sync_interval` How long in seconds to wait between intervals for SRDF sync checks during Cinder PowerMax SRDF operations. Default is 3 seconds.
- `sync_retries` How many times to retry RDF sync checks during Cinder PowerMax SRDF operations. Default is 200 retries.
- `allow_extend` Only applicable to *VMAX-Hybrid* arrays or All Flash arrays running HyperMax OS (5977). It is a flag for allowing the extension of replicated volumes. To extend a volume in an SRDF relationship, this relationship must first be broken, the R1 device extended, and a new device pair established. If not explicitly set, this flag defaults to `False`.

Note

As the SRDF link must be severed, due caution should be exercised when performing this operation. If absolutely necessary, only one source and target pair should be extended at a time (only only applicable to *VMAX-Hybrid* arrays or All Flash arrays with HyperMax OS).

4. Create a `replication-enabled` volume type. Once the `replication_device` parameter has been entered in the PowerMax backend entry in the `cinder.conf`, a corresponding volume type needs to be created `replication_enabled` property set. See above *Create volume types* for details.

```
# openstack volume type set --property replication_enabled="<is> True" \
                        POWERMAX_FC_REPLICATION
```

Note

Service Level and Workload: An attempt will be made to create a storage group on the target array with the same service level and workload combination as the primary. However, if this combination is unavailable on the target (for example, in a situation where the source array is a *VMAX-Hybrid*, the target array is an All Flash, and an All Flash incompatible service level like Bronze is configured), no service level will be applied.

Configure multiple replication targets

Setting multiple replication devices in cinder.conf allows the use of all the supported replication modes simultaneously. Up to three replication devices can be set, one for each of the replication modes available. An additional volume type `extra spec` (`storagetype:replication_device_backend_id`) is then used to determine which replication device should be utilized when attempting to perform an operation on a volume which is replication enabled. All details, guidelines and recommendations set out in the *Configure a single replication target* section also apply in a multiple replication device scenario.

Multiple replication targets limitations and restrictions:

1. There can only be one of each replication mode present across all of the replication devices set in `cinder.conf`.
2. Details for `target_device_id`, `remote_port_group` and `remote_pool` should be identical across replication devices.
3. The `backend_id` and `rdf_group_label` values must be unique across all replication devices.

Adding additional `replication_device` to cinder.conf:

1. Open `cinder.conf` for editing
2. If a replication device is already present, add the `backend_id` key with a value of `backend_id_legacy_rep`. If this key is already defined, its value must be updated to `backend_id_legacy_rep`.
3. Add the additional replication devices to the backend stanza. Any additional replication devices must have a `backend_id` key set. The value of these must **not** be `backend_id_legacy_rep`.

Example existing backend stanza pre-multiple replication:

```
enabled_backends = POWERMAX_FC_REPLICATION

[POWERMAX_FC_REPLICATION]
volume_driver = cinder.volume.drivers.dell_emc.powermax.fc.PowerMaxFCDriver
san_ip = 10.10.10.10
san_login = my_u4v_username
san_password = my_u4v_password
powermax_srp = SRP_1
powermax_array = 000123456789
powermax_port_groups = [OS-FC-PG]
volume_backend_name = POWERMAX_FC_REPLICATION
replication_device = backend_id:id,
                     target_device_id:000197811111,
                     remote_port_group:os-failover-pg,
                     remote_pool:SRP_1,
                     rdf_group_label: 28_11_07,
                     mode:Metro,
                     metro_use_bias:False,
                     sync_interval:3,
                     sync_retries:200
```

Example updated backend stanza:

```
enabled_backends = POWERMAX_FC_REPLICATION

[POWERMAX_FC_REPLICATION]
volume_driver = cinder.volume.drivers.dell_emc.powermax.fc.PowerMaxFCDriver
san_ip = 10.10.10.10
san_login = my_u4v_username
san_password = my_u4v_password
powermax_srp = SRP_1
powermax_array = 000123456789
powermax_port_groups = [OS-FC-PG]
volume_backend_name = POWERMAX_FC_REPLICATION
replication_device = backend_id:backend_id_legacy_rep
                     target_device_id:000197811111,
                     remote_port_group:os-failover-pg,
                     remote_pool:SRP_1,
                     rdf_group_label: 28_11_07,
                     mode:Metro,
                     metro_use_bias:False,
                     sync_interval:3,
                     sync_retries:200
replication_device = backend_id:sync-rep-id
                     target_device_id:000197811111,
                     remote_port_group:os-failover-pg,
                     remote_pool:SRP_1,
                     rdf_group_label: 29_12_08,
                     mode:Synchronous,
                     sync_interval:3,
                     sync_retries:200
replication_device = backend_id:async-rep-id
                     target_device_id:000197811111,
                     remote_port_group:os-failover-pg,
                     remote_pool:SRP_1,
                     rdf_group_label: 30_13_09,
                     mode:Asynchronous,
                     sync_interval:3,
                     sync_retries:200
```

Note

For environments without existing replication devices. The `backend_id` values can be set to any value for all replication devices. The `backend_id_legacy_rep` value is only needed when updating a legacy system with an existing replication device to use multiple replication devices.

The additional replication devices defined in `cinder.conf` will be detected after restarting the cinder volume service.

To specify which `replication_device` a volume type should use an additional property named `storagetype:replication_device_backend_id` must be added to the extra specs of the volume type. The id value assigned to the `storagetype:replication_device_backend_id` key in the volume type must match the `backend_id` assigned to the `replication_device` in `cinder.conf`.

```
# openstack volume type set \
--property storagetype:replication_device_backend_id="<id>" \
<VOLUME_TYPE>
```

Note

Specifying which replication device to use is done in addition to the basic replication setup for a volume type seen in *Configure a single replication target*

Note

In a legacy system where volume types are present that were replication enabled before adding multiple replication devices, the `storagetype:replication_device_backend_id` should be omitted from any volume type that does/will use the legacy `replication_device` i.e. when `storagetype:replication_device_backend_id` is omitted the replication_device with a `backend_id` of `backend_id_legacy_rep` will be used.

Volume replication interoperability with other features

Most features are supported, except for the following:

- Replication Group operations are available for volumes in Synchronous mode only.
- The Ussuri release of OpenStack supports retyping in-use volumes to and from replication enabled volume types with limited exception of volumes with Metro replication enabled. To retype to a volume-type that is Metro enabled the volume **must** first be detached then retyped. The reason for this is so the paths from the Nova instance to the Metro R1 & R2 volumes must be initialised, this is not possible on the R2 device whilst a volume is attached.
- The image volume cache functionality is supported (enabled by setting `image_volume_cache_enabled = True`), but one of two actions must be taken when creating the cached volume:
  - The first boot volume created on a backend (which will trigger the cached volume to be created) should be the smallest necessary size. For example, if the minimum size disk to hold an image is 5GB, create the first boot volume as 5GB. All subsequent boot volumes are extended to the user specific size.
  - Alternatively, ensure that the `allow_extend` option in the `replication_device parameter` is set to `True`. This is only applicable to *VMAX-Hybrid* arrays or All Flash array with HyperMax OS.

Failover host

Note

Failover and failback operations are not applicable in Metro configurations.

In the event of a disaster, or where there is required downtime, upgrade of the primary array for example, the administrator can issue the failover host command to failover to the configured target:

```
# cinder failover-host cinder_host@POWERMAX_FC_REPLICATION
```

After issuing `cinder failover-host` Cinder will set the R2 array as the target array for Cinder, however, to get existing instances to use this new array and paths to volumes it is necessary to first shelve Nova instances and then unshelve them, this will effectively restart the Nova instance and re-establish data paths between Nova instances and the volumes on the R2 array.

```
# nova shelve <server>
# nova unshelve [--availability-zone <availability_zone>] <server>
```

When a host is in failover mode performing normal volume or snapshot provisioning will not be possible, failover host mode simply provides access to replicated volumes to minimise environment down-time. The primary objective whilst in failover mode should be to get the R1 array back online. When the primary array becomes available again, you can initiate a fail-back using the same failover command and specifying `--backend_id default`:

```
# cinder failover-host cinder_host@POWERMAX_FC_REPLICATION --backend_id␣
↪default
```

After issuing the failover command to revert to the default backend host it is necessary to re-issue the Nova shelve and unshelve commands to restore the data paths between Nova instances and their corresponding back end volumes. Once reverted to the default backend volume and snapshot provisioning operations can continue as normal.

Failover promotion

Failover promotion can be used to transfer all existing RDF enabled volumes to the R2 array and overwrite any references to the original R1 array. This can be used in the event of total R1 array failure or in other cases where an array transfer is warranted. If the R1 array is online and working and the RDF links are still enabled the failover promotion will automatically delete rdf pairs as necessary. If the R1 array or the link to the R1 array is down, a half deletepair must be issued manually for those volumes during the failover promotion.

1. Issue failover command:

```
# cinder failover-host <host>
```

2. Enable array promotion:

```
# cinder failover-host --backend_id=pmax_failover_start_array_promotion <host>
```

3. View and re-enable the cinder service

```
# cinder service-list
# cinder service-enable <host> <binary>
```

Note

With Cinder volume active/active deployment, use the following commands to view and enable the cluster as well.

```
# cinder --os-volume-api-version 3.17 cluster-list
# cinder --os-volume-api-version 3.17 cluster-enable [<binary>] <cluster-
↪name>
```

4. Remove all volumes from volume groups

```
# cinder --os-volume-api-version 3.13 group-update --remove-volumes <Vol1ID,␣
↪etc..> <volume_group_name>
```

5. Detach all volumes that are attached to instances

```
# openstack server remove volume <instance_id> <volume_id>
```

Note

Deleting the instance will call a detach volume for each attached volume. A terminate connection can be issued manually using the following command for volumes that are stuck in the attached state without an instance.

```
# cinder --os-volume-api-version 3.50 attachment-delete <attachment_id>
```

6. Delete all remaining instances

```
# nova delete <instance_id>
```

7. Create new volume types

New volume types must be created with references to the remote array. All new volume types must adhere to the following guidelines:

```
1. Uses the same workload, SLO & compression setting as the previous R1␣
↪volume type.
2. Uses the remote array instead of the primary for its pool name.
3. Uses the same volume_backend_name as the previous volume type.
4. Must not have replication enabled.
```

Example existing volume type extra specs.

```
pool_name='Gold+None+SRP_1+000297900330', replication_enabled='<is> True',
storagetype:replication_device_backend_id='async-rep-1', volume_backend_name=
↪'POWERMAX_ISCSI_NONE'
```

Example new volume type extra specs.

```
pool_name='Gold+None+SRP_1+000197900049', volume_backend_name='POWERMAX_ISCSI_
↪NONE'
```

8. Retype volumes to new volume types

Additional checks will be performed during failover promotion retype to ensure workload, compression and slo settings meet the criteria specified above when creating the new volume types.

```
# cinder retype --migration-policy on-demand <volume> <volume_type>
```

Note

If the volumes RDF links are offline during this retype then a half deletepair must be performed manually after retype. Please reference section 8.a. below for guidance on this process.

8.a. Retype and RDF half deletepair

In instances where the rdf links are offline and rdf pairs have been set to partitioned state there are additional requirements. In that scenario the following order should be adhered to:

```
1. Retype all Synchronous volumes.
2. Half_deletepair all Synchronous volumes using the default storage group.
3. Retype all Asynchronous volumes.
4. Half_deletepair all Asynchronous volumes using their management storage␣
↪group.
5. Retype all Metro volumes.
6. Half_deletepair all Metro volumes using their management storage group.
7. Delete the Asynchronous and Metro management storage groups.
```

Note

A half deletepair cannot be performed on Metro enabled volumes unless the symforce option has been enabled in the symapi options. In symapi/config/options uncomment and set SYMAPI_ALLOW_RDF_SYMFORCE = True.

```
# symrdf -sid <sid> -sg <sg> -rdfg <rdfg> -force -symforce half_deletepair
```

9. Issue failback

Issuing the failback command will disable both the failover and promotion flags. Please ensure all volumes have been retyped and all replication pairs have been deleted before issuing this command.

```
# cinder failover-host --backend_id default <host>
```

10. Update cinder.conf

Update the cinder.conf file to include details for the new primary array. For more information please see the Configure block storage in cinder.conf section of this documentation.

11. Restart the cinder services

Restart the cinder volume service to allow it to detect the changes made to the cinder.conf file.

12. Set Metro volumes to ready state

Metro volumes will be set to a Not Ready state after performing rdf pair cleanup. Set these volumes back to Ready state to allow them to be attached to instances. The U4P instance must be restarted for this change to be detected.

```
# symdev -sid <sid> ready -devs <dev_id1, dev_id2>
```

Asynchronous and metro replication management groups

Asynchronous and metro volumes in an RDF session, i.e. belonging to an SRDF group, must be managed together for RDF operations (although there is a `consistency exempt` option for creating and deleting pairs in an Async group). To facilitate this management, we create an internal RDF management storage group on the backend. This RDF management storage group will use the following naming convention:

```
OS-[rdf_group_label]-[replication_mode]-rdf-sg
```

It is crucial for correct management that the volumes in this storage group directly correspond to the volumes in the RDF group. For this reason, it is imperative that the RDF group specified in the `cinder.conf` is for the exclusive use by this Cinder backend. If there are any issues with the state of your RDF enabled volumes prior to performing additional operations in Cinder you will be notified in the Cinder volume logs.

Metro support

SRDF/Metro is a high availability solution. It works by masking both sides of the RDF relationship to the host, and presenting all paths to the host, appearing that they all point to the one device. In order to do this, there needs to be multi-path software running to manage writing to the multiple paths.

Note

The metro issue around formatting volumes when they are added to existing metro RDF groups has been fixed in Unisphere for PowerMax 9.1, however, it has only been addressed on arrays with PowerMax OS and will not be available on arrays running a HyperMax OS.

Volume retype - storage assisted volume migration

Volume retype with storage assisted migration is supported now for PowerMax arrays. Cinder requires that for storage assisted migration, a volume cannot be retyped across backends. For using storage assisted volume retype, follow these steps:

Note

From the Ussuri release of OpenStack the PowerMax driver supports retyping in-use volumes to and from replication enabled volume types with limited exception of volumes with Metro replication enabled. To retype to a volume-type that is Metro enabled the volume **must** first be detached then retyped. The reason for this is so the paths from the instance to the Metro R1 & R2 volumes must be initialised, this is not possible on the R2 device whilst a volume is attached.

Note

When multiple replication devices are configured. If retyping from one replication mode to another the R1 device ID is preserved and a new R2 side device is created. As a result, the device ID on the R2 array may be different after the retype operation has completed.

Note

Retyping an in-use volume to a metro enabled volume type is not currently supported via storage-assisted migration. This retype can still be performed using host-assisted migration by setting the migration-policy to `on-demand`.

```
cinder retype --migration-policy on-demand <volume> <volume-type>
```

1. For migrating a volume from one Service Level or Workload combination to another, use volume retype with the migration-policy to on-demand. The target volume type should have the same volume_backend_name configured and should have the desired pool_name to which you are trying to retype to (please refer to *Create volume types* for details).

   ```
   $ cinder retype --migration-policy on-demand <volume> <volume-type>
   ```

Generic volume group support

Generic volume group operations are performed through the CLI using API version 3.1x of the Cinder API. Generic volume groups are multi-purpose groups which can be used for various features. The PowerMax driver supports consistent group snapshots and replication groups. Consistent group snapshots allows the user to take group snapshots which are consistent based on the group specs. Replication groups allow for tenant facing APIs to enable and disable replication, and to failover and failback, a group of volumes. Generic volume groups have replaced the deprecated consistency groups.

Consistent group snapshot

To create a consistent group snapshot, set a group-spec, having the key `consistent_group_snapshot_enabled` set to `<is> True` on the group.

```
# cinder --os-volume-api-version 3.11 group-type-key GROUP_TYPE set␣
↪consistent_group_snapshot_enabled="<is> True"
```

Similarly the same key should be set on any volume type which is specified while creating the group.

```
# openstack volume type set --property consistent_group_snapshot_enabled="<is>
↪ True" POWERMAX_GROUP
```

If this key is not set on the group-spec or volume type, then the generic volume group will be created/managed by Cinder (not the PowerMax driver).

Note

The consistent group snapshot should not be confused with the PowerMax consistency group which is an SRDF construct.

Replication groups

As with Consistent group snapshot `consistent_group_snapshot_enabled` should be set to true on the group and the volume type for replication groups. Only Synchronous replication is supported for use with Replication Groups. When a volume is created into a replication group, replication is on by default. The `disable_replication` api suspends I/O traffic on the devices, but does NOT remove replication for the group. The `enable_replication` api resumes I/O traffic on the RDF links. The `failover_group` api allows a group to be failed over and back without failing over the entire host. See below for usage.

Note

A generic volume group can be both consistent group snapshot enabled and consistent group replication enabled.

Storage group names

Storage groups are created on the PowerMax as a result of creation of generic volume groups. These storage groups follow a different naming convention and are of the following format depending upon whether the groups have a name.

```
TruncatedGroupName_GroupUUID or GroupUUID
```

Group type, group, and group snapshot operations

Please refer to the official OpenStack block-storage groups documentation for the most up to date group operations

Group replication operations

Generic volume group operations no longer require the user to specify the Cinder CLI version, however, performing generic volume group replication operations still require this setting. When running generic volume group commands set the value `--os-volume-api-version` to `3.38`. These commands are not listed in the latest Cinder CLI documentation so will remain here until added to the latest Cinder CLI version or deprecated from Cinder.

This is how to create a replication group. Please refer to the official OpenStack block-storage groups documentation for the most up to date group operations.

- Make sure there is a *replication_device* for Synchronous in *cinder.conf*

```
replication_device = backend_id:backend_id_legacy_rep,target_device_
↪id:0001234567890,remote_port_group:PG1,remote_pool:SRP_1,rdf_group_label:os-
↪sync,mode:Synchronous
```

- Create a volume type with property *replication_enabled=<is> True*.

```
$ openstack volume type create --property replication_enabled='<is> True'␣
↪SYNC_REP_VT
```

- Create a Generic group type with extra specs *consistent_group_snapshot_enabled=<is> True* and *consistent_group_replication_enabled=<is> True*.

```
$ cinder --os-volume-api-version 3.38 group-type-create GROUP_REP_VT
$ cinder --os-volume-api-version 3.38 group-type-key GROUP_REP_VT set \
  consistent_group_snapshot_enabled='<is> True' \
  consistent_group_replication_enabled='<is> True'
```

- Create a Generic group with synchronous volume type SYNC_REP_VT

```
$ cinder --os-volume-api-version 3.13 group-create --name GROUP_REP GROUP_REP_
↪VT GROUP_REP_VT
```

- Create a volume in the Generic group

```
$ cinder --os-volume-api-version 3.38 create --volume-type SYNC_REP_VT --
↪group-id GROUP_REP \
  --name VOL_REP_GROUP 1
```

- Enable group replication

```
$ cinder --os-volume-api-version 3.38 group-enable-replication GROUP_REP
```

- Disable group replication

```
$ cinder --os-volume-api-version 3.38 group-disable-replication GROUP_REP
```

- Failover group

```
$ cinder --os-volume-api-version 3.38 group-failover-replication GROUP_REP
```

- Failback group

```
$ cinder --os-volume-api-version 3.38 group-failover-replication GROUP_REP \
    --secondary-backend-id default
```

Manage and unmanage Volumes

Managing volumes in OpenStack is the process whereby a volume which exists on the storage device is imported into OpenStack to be made available for use in the OpenStack environment. For a volume to be valid for managing into OpenStack, the following prerequisites must be met:

- The volume exists in a Cinder managed pool
- The volume is not part of a Masking View
- The volume is not part of an SRDF relationship
- The volume is configured as a TDEV (thin device)
- The volume is set to FBA emulation
- The volume must a whole GB e.g. 5.5GB is not a valid size
- The volume cannot be a SnapVX target

For a volume to exist in a Cinder managed pool, it must reside in the same Storage Resource Pool (SRP) as the backend which is configured for use in OpenStack. Specifying the pool correctly can be entered manually as it follows the same format:

```
Pool format: <service_level>+<srp>+<array_id>
Pool example: Diamond+SRP_1+111111111111
```

Table 15: Pool values

Key	Value
service_level	The service level of the volume to be managed
srp	The Storage Resource Pool configured for use by the backend
array_id	The PowerMax serial number (12 digit numerical)

Manage volumes

With your pool name defined you can now manage the volume into OpenStack, this is possible with the CLI command `cinder manage`. The `bootable` parameter is optional in the command, if the volume to be managed into OpenStack is not bootable leave this parameter out. OpenStack will also determine the size of the value when it is managed so there is no need to specify the volume size.

Command format:

```
$ cinder manage --name <new_volume_name> --volume-type <powermax_vol_type> \
  --availability-zone <av_zone> <--bootable> <host> <identifier>
```

Command Example:

```
$ cinder manage --name powermax_managed_volume --volume-type POWERMAX_ISCSI_
↪DIAMOND \
  --availability-zone nova demo@POWERMAX_ISCSI_DIAMOND#Diamond+SRP_
↪1+111111111111 031D8
```

After the above command has been run, the volume will be available for use in the same way as any other OpenStack PowerMax volume.

Note

An unmanaged volume with a prefix of `OS-` in its identifier name cannot be managed into OpenStack, as this is a reserved keyword for managed volumes. If the identifier name has this prefix, an exception will be thrown by the PowerMax driver on a manage operation.

Managing volumes with replication enabled

Whilst it is not possible to manage volumes into OpenStack that are part of a SRDF relationship, it is possible to manage a volume into OpenStack and enable replication at the same time. This is done by having a replication enabled PowerMax volume type (for more information see section Volume Replication) during the manage volume process you specify the replication volume type as the chosen volume type. Once managed, replication will be enabled for that volume.

Note

It is not possible to manage into OpenStack SnapVX linked target volumes, only volumes which are a SnapVX source are permitted. We do not want a scenario where a snapshot source can exist outside of OpenStack management.

Unmanage volume

Unmanaging a volume is not the same as deleting a volume. When a volume is deleted from OpenStack, it is also deleted from the PowerMax at the same time. Unmanaging a volume is the process whereby a volume is removed from OpenStack but it remains for further use on the PowerMax. The volume can also be managed back into OpenStack at a later date using the process discussed in the previous section. Unmanaging volume is carried out using the Cinder unmanage CLI command:

Command format:

```
$ cinder unmanage <volume_name/volume_id>
```

Command example:

```
$ cinder unmanage powermax_test_vol
```

Once unmanaged from OpenStack, the volume can still be retrieved using its device ID or OpenStack volume ID. Within Unisphere you will also notice that the `OS-` prefix has been removed, this is another visual indication that the volume is no longer managed by OpenStack.

Manage/unmanage snapshots

Users can manage PowerMax SnapVX snapshots into OpenStack if the source volume already exists in Cinder. Similarly, users will be able to unmanage OpenStack snapshots to remove them from Cinder but keep them on the storage backend.

Set-up, restrictions and requirements:

1. No additional settings or configuration is required to support this functionality.
2. Manage/Unmanage snapshots requires SnapVX functionality support on PowerMax.
3. Manage/Unmanage Snapshots in OpenStack Cinder is only supported at present through Cinder CLI commands.
4. It is only possible to manage or unmanage one snapshot at a time in Cinder.

Manage SnapVX snapshot

It is possible to manage PowerMax SnapVX snapshots into OpenStack, where the source volume from which the snapshot is taken already exists in, and is managed by OpenStack Cinder. The source volume may have been created in OpenStack Cinder, or it may have been managed in to OpenStack Cinder also. With the support of managing SnapVX snapshots included in OpenStack Queens, the restriction around managing SnapVX source volumes has been removed.

Note

It is not possible to manage into OpenStack SnapVX linked target volumes, only volumes which are a SnapVX source are permitted. We do not want a scenario where a snapshot source can exist outside of OpenStack management.

Requirements/restrictions:

1. The SnapVX source volume must be present in and managed by Cinder.

2. The SnapVX snapshot name must not begin with `OS-`.
3. The SnapVX snapshot source volume must not be in a failed-over state.
4. Managing a SnapVX snapshot will only be allowed if the snapshot has no linked target volumes.

Command structure:

1. Identify your SnapVX snapshot for management on the PowerMax, note the name.
2. Ensure the source volume is already managed into OpenStack Cinder, note the device ID.
3. Using the Cinder CLI, use the following command structure to manage a Snapshot into OpenStack Cinder:

```
$ cinder snapshot-manage --id-type source-name
                         [--name <name>]
                         [--description <description>]
                         [--metadata [<key=value> [<key=value> ...]]]
                         <volume name/id> <identifier>
```

Positional arguments:

- `<volume name/id>` Source OpenStack volume name
- `<identifier>` Name of existing snapshot on PowerMax backend

Optional arguments:

- `--name <name>` Snapshot name (Default="None")
- `--description <description>` Snapshot description (Default="None")
- `--metadata [<key=value> [<key=value> ...]]` Metadata `key=value` pairs (Default="None")

Example:

```
$ cinder snapshot-manage --name SnapshotManaged \
                         --description "Managed Queens Feb18" \
                         powermax-vol-1 PowerMaxSnapshot
```

Where:

- The name in OpenStack after managing the SnapVX snapshot will be `SnapshotManaged`.
- The snapshot will have the description `Managed Queens Feb18`.
- The Cinder volume name is `powermax-vol-1`.
- The name of the SnapVX snapshot on the PowerMax backend is `PowerMaxSnapshot`.

Outcome:

After the process of managing the Snapshot has completed, the SnapVX snapshot on the PowerMax backend will be prefixed by the letters `OS-`, leaving the snapshot in this example named `OS-PowerMaxSnapshot`. The associated snapshot managed by Cinder will be present for use under the name `SnapshotManaged`.

Unmanage cinder snapshot

Unmanaging a snapshot in Cinder is the process whereby the snapshot is removed from and no longer managed by Cinder, but it still exists on the storage backend. Unmanaging a SnapVX snapshot in OpenStack Cinder follows this behaviour, whereby after unmanaging a PowerMax SnapVX snapshot from Cinder, the snapshot is removed from OpenStack but is still present for use on the PowerMax backend.

Requirements/Restrictions:

- The SnapVX source volume must not be in a failed over state.

Command Structure:

Identify the SnapVX snapshot you want to unmanage from OpenStack Cinder, note the snapshot name or ID as specified by Cinder. Using the Cinder CLI use the following command structure to unmanage the SnapVX snapshot from Cinder:

```
$ cinder snapshot-unmanage <snapshot>
```

Positional arguments:

- `<snapshot>` Cinder snapshot name or ID.

Example:

```
$ cinder snapshot-unmanage SnapshotManaged
```

Where:

- The SnapVX snapshot name in OpenStack Cinder is SnapshotManaged.

After the process of unmanaging the SnapVX snapshot in Cinder, the snapshot on the PowerMax backend will have the `OS-` prefix removed to indicate it is no longer OpenStack managed. In the example above, the snapshot after unmanaging from OpenStack will be named `PowerMaxSnapshot` on the storage backend.

List manageable volumes and snapshots

Manageable volumes

Volumes that can be managed by and imported into Openstack.

List manageable volume is filtered by:

- Volume size should be 1026MB or greater (1GB PowerMax Cinder Vol = 1026 MB)
- Volume size should be a whole integer GB capacity
- Volume should not be a part of masking view.
- Volume status should be `Ready`
- Volume service state should be `Normal`
- Volume emulation type should be `FBA`
- Volume configuration should be `TDEV`
- Volume should not be a system resource.
- Volume should not be `private`
- Volume should not be `encapsulated`

- Volume should not be `reserved`
- Volume should not be a part of an RDF session
- Volume should not be a SnapVX Target
- Volume identifier should not begin with `OS-`.
- Volume should not be in more than one storage group.

Manageable snaphots

Snapshots that can be managed by and imported into Openstack

List manageable snapshots is filtered by:

- The source volume should be marked as SnapVX source.
- The source volume should be 1026MB or greater
- The source volume should be a whole integer GB capacity.
- The source volume emulation type should be `FBA`.
- The source volume configuration should be `TDEV`.
- The source volume should not be `private`.
- The source volume should be not be a system resource.
- The snapshot identifier should not start with `OS-` or `temp-`.
- The snapshot should not be expired.
- The snapshot generation number should npt be greater than 0.

Note

There is some delay in the syncing of the Unisphere for PowerMax database when the state/properties of a volume is modified using `symcli`. To prevent this it is preferable to modify state/properties of volumes within Unisphere.

Cinder backup support

PowerMax Cinder driver support Cinder backup functionality. For further information on setup, configuration and usage please see the official OpenStack volume backup documentation and related volume backup CLI guide.

Note

`rpc_response_timeout` may need to be increased significantly in volume backup operations especially in replication scenarios where the creation operation will be longer. For more information on `rpc_response_timeout` please refer to *Live migration configuration*

Port group & port load balancing

By default port groups are selected at random from `cinder.conf` when connections are initialised between volumes on the backend array and compute instances in Nova. If a port group is set in the volume type extra specifications this will take precedence over any port groups configured in `cinder.conf`. Port selection within the chosen port group is also selected at random by default.

With port group and port load balancing in the PowerMax for Cinder driver users can now select the port group and port load by determining which has the lowest load. The load metric is defined by the user in both instances so the selection process can better match the needs of the user and their environment. Available metrics are detailed in the `performance metrics` section.

Port Groups are reported on at five minute time deltas (diagnostic), and FE Ports are reported on at one minute time deltas (real-time) if real-time metrics are enabled, else default five minute time delta (diagnostic). The window at which performance metrics are analysed is a user-configured option in `cinder.conf`, this is detailed in the `configuration` section.

Calculating load

The process by which Port Group or Port load is calculated is the same for both. The user specifies the look back window which determines how many performance intervals to measure, 60 minutes will give 12 intervals of 5 minutes each for example. If no lookback window is specified or is set to 0 only the most recent performance metric will be analysed. This will give a slight performance improvement but with the improvements made to the performance REST endpoints for load this improvement is negligible. For real-time stats a minimum of 1 minute is required.

Once a call is made to the performance REST endpoints, the performance data for that PG or port is extracted. Then the metric values are summed and divided by the count of intervals to get the average for the look back window.

The performance metric average value for each asset is added to a Python heap. Once all assets have been measured the lowest value will always be at position 0 in the heap so there is no extra time penalty requirement for search.

Pre-requisites

Before load balancing can be enabled in the PowerMax for Cinder driver performance metrics collection must be enabled in Unisphere. Real-time performance metrics collection is enabled separately from diagnostic metrics collection. Performance metric collection is only available for local arrays in Unisphere.

After performance metrics registration there is a time delay before Unisphere records performance metrics, adequate time must be given before enabling load balancing in Cinder else default random selection method will be used. It is recommended to wait 4 hours after performance registration before enabling load balancing in Cinder.

Configuration

A number of configuration options are available for users so load balancing can be set to better suit the needs of the environment. These configuration options are detailed in the table below.

Table 16: Load balance cinder.conf configuration options

cinder.conf parameter	options	Default	Description
load_balance	True/False	False	Enable/disable load balancing for a PowerMax backend.
load_balance_real_t	True/False	False	Enable/disable real-time performance metrics for Port level metrics (not available for Port Group).
load_data_format	Avg/Max	Avg	Performance data format, not applicable for real-time.
load_lookback	int	60	How far in minutes to look back for diagnostic performance metrics in load calculation, minimum of 0 maximum of 1440 (24 hours).
load_real_time_look	int	1	How far in minutes to look back for real-time performance metrics in load calculation, minimum of 1 maximum of 60 (24 hours).
snapvx_unlink_symfc	True/False	False	Enable/disable symforce for SnapVx unlink.
port_group_load_met	See below	PercentBusy	Metric used for port group load

Port-Group Metrics

Table 17: Port-group performance metrics

Metric	cinder.conf option	Description
% Busy	`PercentBusy`	The percent of time the port group is busy.
Avg IO Size (KB)	`AvgIOSize`	Calculated value: (HA Kbytes transferred per sec / total IOs per sec)
Host IOs/sec	`IOs`	The number of host IO operations performed each second, including writes and random and sequential reads.
Host MBs/sec	`MBs`	The number of host MBs read each second.
MBs Read/sec	`MBRead`	The number of reads per second in MBs.
MBs Written/sec	`MBWritten`	The number of writes per second in MBs.
Reads/sec	`Reads`	The average number of host reads performed per second.
Writes/sec	`Writes`	The average number of host writes performed per second.

Port Metrics

Table 18: Port performance metrics

Metric	cinder.conf option	Real-Time Supported	Description
% Busy	`PercentBusy`	Yes	The percent of time the port is busy.
Avg IO Size (KB)	`AvgIOSize`	Yes	Calculated value: (HA Kbytes transferred per sec / total IOs per sec)
Host IOs/sec	`IOs`	Yes	The number of host IO operations performed each second, including writes and random and sequential reads.
Host MBs/sec	`MBs`	Yes	The number of host MBs read each second.
MBs Read/sec	`MBRead`	Yes	The number of reads per second in MBs.
MBs Written/sec	`MBWritten`	Yes	The number of writes per second in MBs.
Reads/sec	`Reads`	Yes	The number of read operations performed by the port per second.
Writes/sec	`Writes`	Yes	The number of write operations performed each second by the port.
Speed Gb/sec	`SpeedGBs`	No	Speed.
Response Time (ms)	`ResponseTime`	No	The average response time for the reads and writes.
Read RT (ms)	`ReadResponseTime`	No	The average time it takes to serve one read IO.
Write RT (ms)	`WriteResponseTime`	No	The average time it takes to serve one write IO.

Upgrading from SMI-S based driver to REST API based driver

Seamless upgrades from an SMI-S based driver to REST API based driver, following the setup instructions above, are supported with a few exceptions:

1. Seamless upgrade from SMI-S(Ocata and earlier) to REST(Pike and later) is now available on all functionality including Live Migration.
2. Consistency groups are deprecated in Pike. Generic Volume Groups are supported from Pike onwards.

Known issues

These known issues exist in the current release of OpenStack:

- Launchpad #1951977 Cannot create backups for metro volumes with multipath enabled.

Dell PowerStore driver

This section explains how to configure and connect the block storage nodes to an PowerStore storage cluster.

Supported operations

- Create, delete, attach and detach volumes.
- Create, delete volume snapshots.
- Create a volume from a snapshot.
- Copy an image to a volume.
- Copy a volume to an image.
- Clone a volume.
- Extend a volume.
- Get volume statistics.
- Attach a volume to multiple servers simultaneously (multiattach).
- Revert a volume to a snapshot.
- OpenStack replication v2.1 support.
- Create, delete, update Consistency Groups.
- Create, delete Consistency Groups snapshots.
- Clone a Consistency Group.
- Create a Consistency Group from a Consistency Group snapshot.
- Quality of Service (QoS)
- Cinder volume active/active support.

Driver configuration

Add the following content into `/etc/cinder/cinder.conf`:

```
[DEFAULT]
enabled_backends = powerstore

[powerstore]
# PowerStore REST IP
san_ip = <San IP>
# PowerStore REST username and password
san_login = <San username>
san_password = <San Password>
# Storage protocol
storage_protocol = <Storage protocol> # FC or iSCSI
# Volume driver name
volume_driver = cinder.volume.drivers.dell_emc.powerstore.driver.
↪PowerStoreDriver
# Backend name
volume_backend_name = <Backend name>
# PowerStore allowed ports
powerstore_ports = <Allowed ports> # Ex. 58:cc:f0:98:49:22:07:02,
↪58:cc:f0:98:49:23:07:02
```

Driver configuration to use NVMe-OF

NVMe-OF support was added in PowerStore starting from version 2.1.

Note

Currently the driver supports only NVMe over TCP.

To configure NVMe-OF driver add the following content into `/etc/cinder/cinder.conf`:

```
[DEFAULT]
enabled_backends = powerstore

[powerstore]
# PowerStore REST IP
san_ip = <San IP>
# PowerStore REST username and password
san_login = <San username>
san_password = <San Password>
# Volume driver name
volume_driver = cinder.volume.drivers.dell_emc.powerstore.driver.
↪PowerStoreDriver
# Backend name
volume_backend_name = <Backend name>
powerstore_nvme = True
```

Driver options

The driver supports the following configuration options:

Table 19: Description of configuration options

Configuration option = Default value	Description
powerstore_nvme = False	(Boolean) Connect PowerStore volumes using NVMe-OF.
powerstore_ports = []	(List of String) Allowed ports. Comma separated list of PowerStore iSCSI IPs or FC WWNs (ex. 58:cc:f0:98:49:22:07:02) to be used. If option is not set all ports are allowed.
rest_api_call_co = 30	(Integer(min=1)) Use this value to specify the connect timeout value (in seconds) for REST API calls to the PowerStore backend.
rest_api_call_re = 30	(Integer(min=1)) Use this value to specify the read timeout value (in seconds) for REST API calls to the PowerStore backend.
powerstore_appli = []	(List of String) Appliances names. Comma separated list of PowerStore appliances names used to provision volumes. **DEPRECATED**

SSL support

To enable the SSL certificate verification, modify the following options in the `cinder.conf` file:

```
driver_ssl_cert_verify = True
driver_ssl_cert_path = <path to the CA>
```

By default, the SSL certificate validation is disabled.

If the `driver_ssl_cert_path` option is omitted, the system default CA will be used.

Image Volume Caching support

The image volume cache functionality is supported. To enable it, modify the following options in the `cinder.conf` file:

```
image_volume_cache_enabled = True
```

By default, Image Volume Caching is disabled.

Thin provisioning and compression

The driver creates thin provisioned compressed volumes by default. Thick provisioning is not supported.

CHAP authentication support

The driver supports one-way (Single mode) CHAP authentication. To use CHAP authentication CHAP Single mode has to be enabled on the storage side.

Note

When enabling CHAP, any previously added hosts will need to be updated with CHAP configuration since there will be I/O disruption for those hosts. It is recommended that before adding hosts to the cluster, decide what type of CHAP configuration is required, if any.

CHAP configuration is retrieved from the storage during driver initialization, no additional configuration is needed. Secrets are generated automatically.

Replication support

Configure replication

1. Pair source and destination PowerStore systems.
2. Create Protection policy and Replication rule with desired RPO.
3. Enable replication in `cinder.conf` file.

   To enable replication feature for storage backend set `replication_device` as below:

   ```
   ...
   replication_device = backend_id:powerstore_repl_1,
                        san_ip: <Replication system San ip>,
                        san_login: <Replication system San username>,
                        san_password: <Replication system San password>
   ```

   - Only one replication device is supported for storage backend.
   - Replication device supports the same options as the main storage backend.

4. Create volume type for volumes with replication enabled.

   ```
   $ openstack volume type create powerstore_replicated
   $ openstack volume type set --property replication_enabled='<is> True' ↪powerstore_replicated
   ```

5. Set Protection policy name for volume type.

   ```
   $ openstack volume type set --property powerstore:protection_policy=
   ↪<protection policy name> \
       powerstore_replicated
   ```

Failover host

In the event of a disaster, or where there is a required downtime the administrator can issue the failover host command:

```
$ cinder failover-host cinder_host@powerstore --backend_id powerstore_repl_1
```

After issuing Cinder failover-host command Cinder will switch to configured replication device, however to get existing instances to use this target and new paths to volumes it is necessary to first shelve Nova instances and then unshelve them, this will effectively restart the Nova instance and re-establish data paths between Nova instances and the volumes.

```
$ nova shelve <server>
$ nova unshelve [--availability-zone <availability_zone>] <server>
```

If the primary system becomes available, the administrator can initiate failback operation using `--backend_id default`:

```
$ cinder failover-host cinder_host@powerstore --backend_id default
```

Consistency Groups support

To use PowerStore Volume Groups create Group Type with consistent group snapshot enabled.

```
$ cinder --os-volume-api-version 3.11 group-type-create powerstore_vg
$ cinder --os-volume-api-version 3.11 group-type-key powerstore_vg set␣
↪consistent_group_snapshot_enabled="<is> True"
```

Note

Currently driver does not support Consistency Groups replication. Adding volume to Consistency Group and creating volume in Consistency Group will fail if volume is replicated.

QoS (Quality of Service) support

Note

QoS is supported in PowerStore version 4.0 or later.

The PowerStore driver supports Quality of Service (QoS) by enabling the following capabilities:

bandwidth_limit_type

The QoS bandwidth limit type. This type setting determines how the max_iops and max_bw attributes are used. This has the following two values:

1. `Absolute` - Limits are absolute values specified, either I/O operations per second or bandwidth.
2. `Density` - Limits specified are per GB, e.g. I/O operations per second per GB.

Note

This (bandwidth_limit_type) property is mandatory when creating QoS.

max_iops

Maximum I/O operations in either I/O operations per second (IOPS) or I/O operations per second per GB. The specification of the type attribute determines which metric is used. If type is set to absolute, max_iops is specified in IOPS. If type is set to density, max_iops is specified in IOPS per GB. If both max_iops and max_bw are specified, the system will limit I/O if either value is exceeded. The value must be within the range of 1 to 2147483646.

max_bw

Maximum I/O bandwidth measured in either Kilobytes per second or Kilobytes per second / per

GB. The specification of the type attribute determines which measurement is used. If type is set to absolute, max_bw is specified in Kilobytes per second. If type is set to density max_bw is specified in Kilobytes per second / per GB. If both max_iops and max_bw are specified, the system will limit I/O if either value is exceeded. The value must be within the range of 2000 to 2147483646.

burst_percentage
Percentage indicating by how much the limit may be exceeded. If I/O normally runs below the specified limit, then the volume or volume_group will accumulate burst credits that can be used to exceed the limit for a short period (a few seconds, but will not exceed the burst limit). This burst percentage applies to both max_iops and max_bw and is independent of the type setting. The value must be within the range of 0 to 100. If this property is not specified during QoS creation, a default value of 0 will be used.

Note

When creating QoS, you must define either `max_iops` or `max_bw`, or you can define both.

```
$ openstack volume qos create --consumer back-end --property max_iops=100 --property max_bw=50000 --property bandwidth_limit_type=Absolute --property burst_percentage=80 powerstore_qos
$ openstack volume type create --property volume_backend_name=powerstore powerstore
$ openstack volume qos associate powerstore_qos powerstore
```

Note

There are two approaches for updating QoS properties in PowerStore:

1. **Retype the Volume:**
   This involves retyping the volume with the different QoS settings and migrating the volume to the new type.
2. **Modify Existing QoS Properties (Recommended):**
   This method entails changing the existing QoS properties and creating a new instance or image volume to update the QoS policy in PowerStore. This will also update the QoS properties of existing attached volumes, created with the same volume type.

Dell PowerStore NFS Driver

PowerStore NFS driver enables storing Block Storage service volumes on a PowerStore storage back end.

Supported operations

- Create, delete, attach and detach volumes.
- Create, delete volume snapshots.
- Create a volume from a snapshot.
- Copy an image to a volume.
- Copy a volume to an image.

- Clone a volume.
- Extend a volume.
- Get volume statistics.
- Attach a volume to multiple servers simultaneously (multiattach).
- Revert a volume to a snapshot.

Driver configuration

Add the following content into `/etc/cinder/cinder.conf`:

```
[DEFAULT]
enabled_backends = powerstore-nfs

[powerstore-nfs]
volume_driver = cinder.volume.drivers.dell_emc.powerstore.nfs.
↪PowerStoreNFSDriver
nfs_qcow2_volumes = True
nfs_snapshot_support = True
nfs_sparsed_volumes = False
nas_host = <Ip>
nas_share_path = /nfs-export
nas_secure_file_operations = False
nas_secure_file_permissions = False
volume_backend_name = powerstore-nfs
```

Dell PowerStore NFS Copy Offload API

A feature for effective creation of a volume from snapshot/volume was added in PowerStore NFS Driver. The dellfcopy utility provides the ability to copy a file very quickly on a Dell SDNAS filesystem mounted by a client. To download it, contact your local Dell representative.

The dellfcopy tool is used in the following operations:

- Create a volume from a snapshot.
- Clone a volume.

To use PowerStore NFS driver with this feature, you must install the tool with the following command:

```
# sudo dpkg -i ./dellfcopy_1.3-1_amd64.deb
```

Dell PowerVault ME Series Fibre Channel and iSCSI drivers

The `PVMEFCDriver` and `PVMEISCSIDriver` Cinder drivers allow the Dell PowerVault ME Series storage arrays to be used for Block Storage in OpenStack deployments.

System requirements

To use the PowerVault ME Series drivers, the following are required:

- PowerVault ME5 Series storage array with I200 or later firmware, or
- PowerVault ME4 Series storage array with G280 or later firmware
- iSCSI or Fibre Channel host interfaces
- Network connectivity between the OpenStack hosts and the arrays embedded management interface
- The HTTPS protocol must be enabled on the array

Supported operations

- Create, delete, attach, and detach volumes.
- Create, list, and delete volume snapshots.
- Create a volume from a snapshot.
- Copy an image to a volume.
- Copy a volume to an image.
- Clone a volume.
- Extend a volume.
- Migrate a volume with back-end assistance.
- Retype a volume.
- Manage and unmanage a volume.

Configuring the array

1. Verify that the array can be managed via an HTTPS connection. HTTP can also be used if `driver_use_ssl` is set to False in the `cinder.conf` file.

   Confirm that virtual pools A and B are already present on the array. If they are missing, create them.

2. Edit the `cinder.conf` file to define a storage back-end entry for each storage pool on the array that will be managed by OpenStack. Each entry consists of a unique section name, surrounded by square brackets, followed by options specified in a `key=value` format.

   - The `pvme_pool_name` value specifies the name of the storage pool or vdisk on the array.
   - The `volume_backend_name` option value can be a unique value, if you wish to be able to assign volumes to a specific storage pool on the array, or a name that is shared among multiple storage pools to let the volume scheduler choose where new volumes are allocated.

3. The following `cinder.conf` options generally have identical values for each backend section on the array:

   - `volume_driver` specifies the Cinder driver name.
   - `san_ip` specifies the IP addresses or host names of the arrays management controllers.

- `san_login` and `san_password` specify the username and password of an array user account with `manage` privileges
- `driver_use_ssl` must be set to True to enable use of the HTTPS protocol.
- `pvme_iscsi_ips` specifies the iSCSI IP addresses for the array if using the iSCSI transport protocol

In the examples below, two back ends are defined, one for pool A and one for pool B, and a common `volume_backend_name` is used so that a single volume type definition can be used to allocate volumes from both pools.

iSCSI example back-end entries

```
[pool-a]
pvme_pool_name = A
volume_backend_name = pvme-array
volume_driver = cinder.volume.drivers.dell_emc.powervault.iscsi.
↪PVMEISCSIDriver
san_ip = 10.1.2.3,10.1.2.4
san_login = manage
san_password = !manage
pvme_iscsi_ips = 10.2.3.4,10.2.3.5
driver_use_ssl = true

[pool-b]
pvme_pool_name = B
volume_backend_name = pvme-array
volume_driver = cinder.volume.drivers.dell_emc.powervault.iscsi.
↪PVMEISCSIDriver
san_ip = 10.1.2.3,10.1.2.4
san_login = manage
san_password = !manage
pvme_iscsi_ips = 10.2.3.4,10.2.3.5
driver_use_ssl = true
```

Fibre Channel example back-end entries

```
[pool-a]
pvme_pool_name = A
volume_backend_name = pvme-array
volume_driver = cinder.volume.drivers.dell_emc.powervault.fc.PVMEFCDriver
san_ip = 10.1.2.3,10.1.2.4
san_login = manage
san_password = !manage
driver_use_ssl = true

[pool-b]
pvme_pool_name = B
volume_backend_name = pvme-array
volume_driver = cinder.volume.drivers.dell_emc.powervault.fc.PVMEFCDriver
san_ip = 10.1.2.3,10.1.2.4
san_login = manage
```

(continues on next page)

(continued from previous page)

```
san_password = !manage
driver_use_ssl = true
```

4. If HTTPS is enabled, you can enable certificate verification with the option `driver_ssl_cert_verify = True`. You may also use the `driver_ssl_cert_path` parameter to specify the path to a CA_BUNDLE file containing CAs other than those in the default list.

5. Modify the `[DEFAULT]` section of the `cinder.conf` file to add an `enabled_backends` parameter specifying the backend entries you added, and a `default_volume_type` parameter specifying the name of a volume type that you will create in the next step.

   Example of [DEFAULT] section changes

```
[DEFAULT]
enabled_backends = pool-a,pool-b
default_volume_type = pvme
```

6. Create a new volume type for each distinct `volume_backend_name` value that you added in the `cinder.conf` file. The example below assumes that the same `volume_backend_name=pvme-array` option was specified in all of the entries, and specifies that the volume type `pvme` can be used to allocate volumes from any of them.

   Example of creating a volume type

```
$ openstack volume type create pvme
$ openstack volume type set --property volume_backend_name=pvme-array pvme
```

7. After modifying the `cinder.conf` file, restart the `cinder-volume` service.

Driver-specific options

The following table contains the configuration options that are specific to the PowerVault ME Series drivers.

Table 20: Description of PowerVault ME Series configuration options

Configuration option = Default value	Description
`pvme_iscsi_ips` = `[]`	(List of String) List of comma-separated target iSCSI IP addresses.
`pvme_pool_name` = `A`	(String) Pool or Vdisk name to use for volume creation.

Dell Unity driver

Unity driver has been integrated in the OpenStack Block Storage project since the Ocata release. The driver is built on the top of Block Storage framework and a Dell distributed Python package storops.

Prerequisites

Software	Version
Unity OE	4.1.X or newer
storops	1.2.3 or newer

Supported operations

- Create, delete, attach, and detach volumes.
- Create, delete, attach, and detach compressed volumes.
- Create, list, and delete volume snapshots.
- Create a volume from a snapshot.
- Copy an image to a volume.
- Create an image from a volume.
- Clone a volume.
- Extend a volume.
- Migrate a volume.
- Get volume statistics.
- Efficient non-disruptive volume backup.
- Revert a volume to a snapshot.
- Create thick volumes.
- Create volume with tiering policy.
- Create and delete consistent groups.
- Add/remove volumes to/from a consistent group.
- Create and delete consistent group snapshots.
- Clone a consistent group.
- Create a consistent group from a snapshot.
- Attach a volume to multiple servers simultaneously (multiattach).
- Volume replications.
- Consistency group replications.

Driver configuration

Note

The following instructions should all be performed on cinder-volume container.

1. Install *storops* from pypi:

```
# pip install storops
```

2. Add the following content into `/etc/cinder/cinder.conf`:

```
[DEFAULT]
enabled_backends = unity

[unity]
# Storage protocol
storage_protocol = iSCSI
# Unisphere IP
san_ip = <SAN IP>
# Unisphere username and password
san_login = <SAN LOGIN>
san_password = <SAN PASSWORD>
# Volume driver name
volume_driver = cinder.volume.drivers.dell_emc.unity.Driver
# backend's name
volume_backend_name = Storage_ISCSI_01
```

Note

These are minimal options for Unity driver, for more options, see *Driver options*.

Note

(**Optional**) If you require multipath based data access, perform below steps on both Block Storage and Compute nodes.

1. Install `sysfsutils`, `sg3-utils` and `multipath-tools`:

```
# apt-get install multipath-tools sg3-utils sysfsutils
```

2. (Required for FC driver in case *Auto-zoning Support* is disabled) Zone the FC ports of Compute nodes with Unity FC target ports.

3. Enable Unity storage optimized multipath configuration:

Add the following content into `/etc/multipath.conf`

```
blacklist {
    # Skip the files uner /dev that are definitely not FC/iSCSI devices
    # Different system may need different customization
    devnode "^(ram|raw|loop|fd|md|dm-|sr|scd|st)[0-9]*"
    devnode "^hd[a-z][0-9]*"
    devnode "^cciss!c[0-9]d[0-9]*[p[0-9]*]"

    # Skip LUNZ device from VNX/Unity
    device {
        vendor "DGC"
```

(continues on next page)

(continued from previous page)

```
        product "LUNZ"
    }
}

defaults {
    user_friendly_names no
    flush_on_last_del yes
}

devices {
    # Device attributed for EMC CLARiiON and VNX/Unity series ALUA
    device {
        vendor "DGC"
        product ".*"
        product_blacklist "LUNZ"
        path_grouping_policy group_by_prio
        path_selector "round-robin 0"
        path_checker emc_clariion
        features "0"
        no_path_retry 12
        hardware_handler "1 alua"
        prio alua
        failback immediate
    }
}
```

4. Restart the multipath service:

```
# service multipath-tools restart
```

5. Enable multipath for image transfer in `/etc/cinder/cinder.conf` for each backend or in `[backend_defaults]` section as a common configuration for all backends.

```
use_multipath_for_image_xfer = True
```

   Restart the `cinder-volume` service to load the change.

6. Enable multipath for volume attache/detach in `/etc/nova/nova.conf`.

```
[libvirt]
...
volume_use_multipath = True
...
```

7. Restart the `nova-compute` service.

Driver options

Table 21: Description of Unity configuration options

Configuration option = Default value	Description
remove_empty_host = False	(Boolean) To remove the host from Unity when the last LUN is detached from it. By default, it is False.
san_api_port = None	(Port(min=0, max=65535)) Port to use to access the SAN API
san_clustername = <>	(String) Cluster name to use for creating volumes
san_ip = <>	(String) IP address of SAN controller
san_is_local = False	(Boolean) Execute commands locally instead of over SSH; use if the volume service is running on the SAN device
san_login = admin	(String) Username for SAN controller
san_password = <>	(String) Password for SAN controller
san_private_key = <>	(String) Filename of private key to use for SSH authentication
san_ssh_port = 22	(Port(min=0, max=65535)) SSH port to use with SAN
san_thin_provision = True	(Boolean) Use thin provisioning for SAN volumes?
ssh_conn_timeout = 30	(Integer) SSH connection timeout in seconds
ssh_max_pool_conn = 5	(Integer) Maximum ssh connections in the pool
ssh_min_pool_conn = 1	(Integer) Minimum ssh connections in the pool
unity_io_ports = []	(List of String) A comma-separated list of iSCSI or FC ports to be used. Each port can be Unix-style glob expressions.
unity_storage_pool_n = []	(List of String) A comma-separated list of storage pool names to be used.

FC or iSCSI ports option

Specify the list of FC or iSCSI ports to be used to perform the IO. Wild card character is supported. For iSCSI ports, use the following format:

```
unity_io_ports = spa_eth2, spb_eth2, *_eth3
```

For FC ports, use the following format:

```
unity_io_ports = spa_iom_0_fc0, spb_iom_0_fc0, *_iom_0_fc1
```

List the port ID with the **uemcli** command:

```
$ uemcli /net/port/eth show -output csv
...
"spa_eth2","SP A Ethernet Port 2","spa","file, net, iscsi", ...
"spb_eth2","SP B Ethernet Port 2","spb","file, net, iscsi", ...
...

$ uemcli /net/port/fc show -output csv
```

(continues on next page)

(continued from previous page)

```
...
"spa_iom_0_fc0","SP A I/O Module 0 FC Port 0","spa", ...
"spb_iom_0_fc0","SP B I/O Module 0 FC Port 0","spb", ...
...
```

Live migration integration

It is suggested to have multipath configured on Compute nodes for robust data access in VM instances live migration scenario. Once `user_friendly_names no` is set in defaults section of `/etc/multipath.conf`, Compute nodes will use the WWID as the alias for the multipath devices.

To enable multipath in live migration:

Note

Make sure *Driver configuration* steps are performed before following steps.

1. Set multipath in `/etc/nova/nova.conf`:

   ```
   [libvirt]
   ...
   volume_use_multipath = True
   ...
   ```

   Restart *nova-compute* service.

2. Set `user_friendly_names no` in `/etc/multipath.conf`

   ```
   ...
   defaults {
       user_friendly_names no
   }
   ...
   ```

3. Restart the `multipath-tools` service.

Thin and thick provisioning

By default, the volume created by Unity driver is thin provisioned. Run the following commands to create a thick volume.

```
# openstack volume type create --property provisioning:type=thick \
  --property thick_provisioning_support='<is> True' thick_volume_type
# openstack volume create --type thick_volume_type thick_volume
```

Compressed volume support

Unity driver supports `compressed volume` creation, modification and deletion. In order to create a compressed volume, a volume type which enables compression support needs to be created first:

```
$ openstack volume type create CompressedVolumeType
$ openstack volume type set --property provisioning:type=compressed --
↪property compression_support='<is> True' CompressedVolumeType
```

Then create volume and specify the new created volume type.

Note

In Unity, only All-Flash pools support compressed volume, for the other type of pools, compression_support: False will be returned when getting pool stats.

Storage-assisted volume migration support

Unity driver supports storage-assisted volume migration, when the user starts migrating with `cinder migrate --force-host-copy False <volume_id> <host>` or `cinder migrate <volume_id> <host>`, cinder will try to leverage the Unitys native volume migration functionality. If Unity fails to migrate the volume, host-assisted migration will be triggered.

In the following scenarios, Unity storage-assisted volume migration will not be triggered. Instead, host-assisted volume migration will be triggered:

- Volume is to be migrated across backends.
- Migration of cloned volume. For example, if vol_2 was cloned from vol_1, the storage-assisted volume migration of vol_2 will not be triggered.

Retype volume support

Unity driver supports to change a volumes type after its creation.

```
$ cinder retype [--migration-policy <never|on-demand>] <volume> <volume-type>
```

The migration-policy is not enabled by default. Some retype operations will require migration based on back-end support. In these cases, the storage-assisted migration will be triggered regardless the migration-policy. For examples: retype between thin and thick, retype between thick and compressed, retype to type(s) current host doesnt support.

QoS support

Unity driver supports `maxBWS` and `maxIOPS` specs for the back-end consumer type. `maxBWS` represents the `Maximum Bandwidth (KBPS)` absolute limit, `maxIOPS` represents the `Maximum IO/S` absolute limit on the Unity respectively.

Storage tiering support

Unity supports fully automated storage tiering which requires the FAST VP license activated on the Unity. The OpenStack administrator can use the extra spec key `storagetype:tiering` to set the tiering policy of a volume and use the key `fast_support='<is> True'` to let Block Storage scheduler find a volume back end which manages a Unity with FAST VP license activated. There are four supported values for the extra spec key `storagetype:tiering` when creating volume.

- Key: `storagetype:tiering`

- Possible values:
  - `StartHighThenAuto`
  - `Auto`
  - `HighestAvailable`
  - `LowestAvailable`
- Default: `StartHighThenAuto`

Run the following commands to create a volume type with tiering policy:

```
$ openstack volume type create VolumeOnAutoTier
$ openstack volume type set --property storagetype:tiering=Auto --property␣
↪fast_support='<is> True' VolumeOnAutoTier
```

Auto-zoning support

Unity volume driver supports auto-zoning, and share the same configuration guide for other vendors. Refer to *Fibre Channel Zone Manager* for detailed configuration steps.

Solution for LUNZ device

The EMC host team also found LUNZ on all of the hosts, EMC best practice is to present a LUN with HLU 0 to clear any LUNZ devices as they can cause issues on the host. See KB LUNZ Device.

To workaround this issue, Unity driver creates a *Dummy LUN* (if not present), and adds it to each host to occupy the *HLU 0* during volume attachment.

Note

This *Dummy LUN* is shared among all hosts connected to the Unity.

Efficient non-disruptive volume backup

The default implementation in Block Storage for non-disruptive volume backup is not efficient since a cloned volume will be created during backup.

An effective approach to backups is to create a snapshot for the volume and connect this snapshot to the Block Storage host for volume backup.

SSL support

Admin is able to enable the SSL verification for any communication against Unity REST API.

By default, the SSL verification is disabled, user can enable it by following steps:

1. Setup the Unity array certificate and import it to the Unity, see section *Storage system certificate* of Security Configuration Guide.
2. Import the CA certificate to the Cinder nodes on which the driver is running.
3. Enable the changes on cinder nodes and restart the cinder services.

```
[unity]
...
driver_ssl_cert_verify = True
driver_ssl_cert_path = <path to the CA>
...
```

If *driver_ssl_cert_path* is omitted, the system default CA will be used for CA verification.

IPv6 support

This driver can support IPv6-based control path and data path.

For control path, please follow below steps:

- Enable Unitys Unipshere IPv6 address.
- Configure the IPv6 network to make sure that cinder node can access Unishpere via IPv6 address.
- Change Cinder config file `/etc/cinder/cinder.conf`. Make the `san_ip` as Unisphere IPv6 address. For example, `san_ip = [fd99:f17b:37d0::100]`.
- Restart the Cinder service to make new configuration take effect.

Note: The IPv6 support on control path depends on the fix of cpython bug 32185. Please make sure your Pythons version includes this bugs fix.

For data path, please follow below steps:

- On Unity, Create iSCSI interface with IPv6 address.
- Configure the IPv6 network to make sure that you can `ping` the Unitys iSCSI IPv6 address from the Cinder node.
- If you create a volume using Cinder and attach it to a VM, the connection between this VM and volume will be IPv6-based iSCSI.

Force detach volume from all hosts

The user could use *os-force_detach* action to detach a volume from all its attached hosts. For more detail, please refer to https://docs.openstack.org/api-ref/block-storage/v3/?expanded=force-detach-a-volume-detail#force-detach-a-volume

Consistent group support

For a group to support consistent group snapshot, the group specs in the corresponding group type should have the following entry:

```
{'consistent_group_snapshot_enabled': <is> True}
```

Similarly, for a volume to be in a group that supports consistent group snapshots, the volume type extra specs would also have the following entry:

```
{'consistent_group_snapshot_enabled': <is> True}
```

Refer to *Generic volume groups* for command lines detail.

Volume replications

To enable volume replications, follow below steps:

1. On Unisphere, configure remote system and interfaces for replications.

The way could be different depending on the type of replications - sync or async. Refer to Unity Replication White Paper for more detail.

2. Add *replication_device* to storage backend settings in *cinder.conf*, then restart Cinder Volume service.

   Example of *cinder.conf* for volume replications:

```
[unity-primary]
san_ip = xxx.xxx.xxx.xxx
...
replication_device = backend_id:unity-secondary,san_ip:yyy.yyy.yyy.yyy,
↪san_login:username,san_password:****,max_time_out_of_sync:60
```

   - Only one *replication_device* can be configured for each primary backend.
   - Keys *backend_id*, *san_ip*, *san_password*, and *max_time_out_of_sync* are supported in *replication_device*, while *backend_id* and *san_ip* are required.
   - *san_password* uses the same one as primary backends if it is omitted.
   - *max_time_out_of_sync* is the max time in minutes replications are out of sync. It must be equal or greater than *0*. *0* means sync replications of volumes will be created. Note that remote systems for sync replications need to be created on Unity first. *60* will be used if it is omitted.

3. Create a volume type with property *replication_enabled=<is> True*.

```
$ openstack volume type create --property replication_enabled='<is> True'␣
↪type-replication
```

4. Any volumes with volume type of step #3 will failover to secondary backend after *failover_host* is executed.

```
$ cinder failover-host --backend_id unity-secondary stein@unity-primary
```

5. Later, they could be failed back.

```
$ cinder failover-host --backend_id default stein@unity-primary
```

Note

The volume can be deleted even when it is participating in a replication. The replication session will be deleted from Unity before the LUN is deleted.

Consistency group replications

To enable consistency group replications, follow below steps:

1. On Unisphere, configure remote system and interfaces for replications.

The way could be different depending on the type of replications - sync or async. Refer to Unity Replication White Paper for more detail.

2. Add *replication_device* to storage backend settings in *cinder.conf*, then restart Cinder Volume service.

   Example of *cinder.conf* for volume replications:

```
[unity-primary]
san_ip = xxx.xxx.xxx.xxx
...
replication_device = backend_id:unity-secondary,san_ip:yyy.yyy.yyy.yyy,
↪san_login:username,san_password:****,max_time_out_of_sync:60
```

   - Only one *replication_device* can be configured for each primary backend.
   - Keys *backend_id*, *san_ip*, *san_password*, and *max_time_out_of_sync* are supported in *replication_device*, while *backend_id* and *san_ip* are required.
   - *san_password* uses the same one as primary backends if it is omitted.
   - *max_time_out_of_sync* is the max time in minutes replications are out of sync. It must be equal or greater than *0*. *0* means sync replications of volumes will be created. Note that remote systems for sync replications need to be created on Unity first. *60* will be used if it is omitted.

3. Create a volume type with property *replication_enabled=<is> True*.

```
$ openstack volume type create --property replication_enabled='<is> True'␣
↪type-replication
```

4. Create a consistency group type with properties *consistent_group_snapshot_enabled=<is> True* and *consistent_group_replication_enabled=<is> True*.

```
$ cinder --os-volume-api-version 3.38 group-type-create type-cg-
↪replication
$ cinder --os-volume-api-version 3.38 group-type-key type-cg-replication␣
↪set
consistent_group_snapshot_enabled='<is> True' consistent_group_
↪replication_enabled='<is> True'
```

5. Create a group type with volume types support replication.

```
$ cinder --os-volume-api-version 3.38 group-create --name test-cg {type-
↪cg-replication-id} type-replication
```

6. Create volume in the consistency group.

```
$ cinder --os-volume-api-version 3.38 create --volume-type type-
↪replication --group-id {test-cg-id}
--name {volume-name} {size}
```

7. Enable consistency group replication.

```
$ cinder --os-volume-api-version 3.38 group-enable-replication test-cg
```

8. Disable consistency group replication.

```
$ cinder --os-volume-api-version 3.38 group-disable-replication test-cg
```

9. Failover consistency group replication.

```
$ cinder --os-volume-api-version 3.38 group-failover-replication test-cg
```

10. Failback consistency group replication.

```
$ cinder --os-volume-api-version 3.38 group-failover-replication test-cg -
↪-secondary-backend-id default
```

Note

Only support group replication of consistency group, see step 4 and 5 to create consistency group support replication.

Troubleshooting

To troubleshoot a failure in OpenStack deployment, the best way is to enable verbose and debug log, at the same time, leverage the build-in Return request ID to caller to track specific Block Storage command logs.

1. Enable verbose log, set following in `/etc/cinder/cinder.conf` and restart all Block Storage services:

```
[DEFAULT]

...

debug = True
verbose = True

...
```

If other projects (usually Compute) are also involved, set *debug* and `verbose` to `True`.

2. use `--debug` to trigger any problematic Block Storage operation:

```
# cinder --debug create --name unity_vol1 100
```

You will see the request ID from the console, for example:

```
DEBUG:keystoneauth:REQ: curl -g -i -X POST
http://192.168.1.9:8776/v2/e50d22bdb5a34078a8bfe7be89324078/volumes -H
"User-Agent: python-cinderclient" -H "Content-Type: application/json" -H
"Accept: application/json" -H "X-Auth-Token:
{SHA1}bf4a85ad64302b67a39ad7c6f695a9630f39ab0e" -d '{"volume": {"status":
"creating", "user_id": null, "name": "unity_vol1", "imageRef": null,
"availability_zone": null, "description": null, "multiattach": false,
"attach_status": "detached", "volume_type": null, "metadata": {},
"consistencygroup_id": null, "source_volid": null, "snapshot_id": null,
"project_id": null, "source_replica": null, "size": 10}}'
DEBUG:keystoneauth:RESP: [202] X-Compute-Request-Id:
req-3a459e0e-871a-49f9-9796-b63cc48b5015 Content-Type: application/json
Content-Length: 804 X-Openstack-Request-Id:
req-3a459e0e-871a-49f9-9796-b63cc48b5015 Date: Mon, 12 Dec 2016 09:31:44␣
↪GMT
Connection: keep-alive
```

3. Use commands like `grep`, `awk` to find the error related to the Block Storage operations.

```
# grep "req-3a459e0e-871a-49f9-9796-b63cc48b5015" cinder-volume.log
```

Dell VNX driver

EMC VNX driver interacts with configured VNX array. It supports both iSCSI and FC protocol.

The VNX cinder driver performs the volume operations by executing Navisphere CLI (NaviSecCLI) which is a command-line interface used for management, diagnostics, and reporting functions for VNX. It also supports both iSCSI and FC protocol.

System requirements

- VNX Operational Environment for Block version 5.32 or higher.
- VNX Snapshot and Thin Provisioning license should be activated for VNX.
- Python library `storops` version 0.5.7 or higher to interact with VNX.
- Navisphere CLI v7.32 or higher is installed along with the driver.

Supported operations

- Create, delete, attach, and detach volumes.
- Create, list, and delete volume snapshots.
- Create a volume from a snapshot.
- Copy an image to a volume.
- Clone a volume.
- Extend a volume.
- Migrate a volume.
- Retype a volume.

- Get volume statistics.
- Create and delete consistency groups.
- Create, list, and delete consistency group snapshots.
- Modify consistency groups.
- Efficient non-disruptive volume backup.
- Create a cloned consistency group.
- Create a consistency group from consistency group snapshots.
- Replication v2.1 support.
- Generic Group support.
- Revert a volume to a snapshot.

Preparation

This section contains instructions to prepare the Block Storage nodes to use the EMC VNX driver. You should install the Navisphere CLI and ensure you have correct zoning configurations.

Install Navisphere CLI

Navisphere CLI needs to be installed on all Block Storage nodes within an OpenStack deployment. You need to download different versions for different platforms:

- For Ubuntu x64, DEB is available at EMC OpenStack Github.
- For all other variants of Linux, Navisphere CLI is available at Downloads for VNX2 Series or Downloads for VNX1 Series.

Install Python library storops

`storops` is a Python library that interacts with VNX array through Navisphere CLI. Use the following command to install the `storops` library:

```
$ pip install storops
```

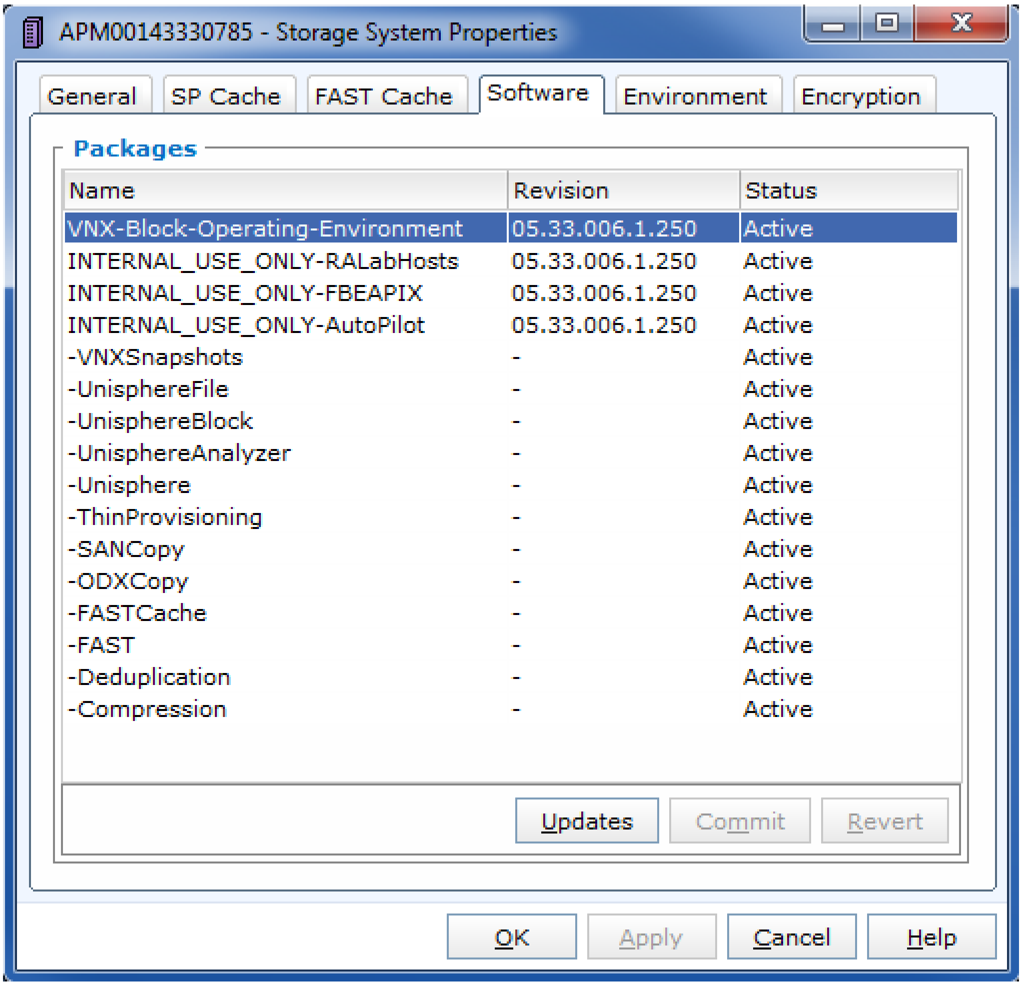
Check array software

Make sure your have the following software installed for certain features:

Feature	Software Required
All	ThinProvisioning
All	VNXSnapshots
FAST cache support	FASTCache
Create volume with type `compressed`	Compression
Create volume with type `deduplicated`	Deduplication

Required software

You can check the status of your array software in the *Software* page of *Storage System Properties*. Here is how it looks like:



Network configuration

For the FC Driver, FC zoning is properly configured between the hosts and the VNX. Check *Register FC port with VNX* for reference.

For the iSCSI Driver, make sure your VNX iSCSI port is accessible by your hosts. Check *Register iSCSI port with VNX* for reference.

You can use `initiator_auto_registration = True` configuration to avoid registering the ports manually. Check the detail of the configuration in *Back-end configuration* for reference.

If you are trying to setup multipath, refer to *Multipath setup*.

Back-end configuration

Make the following changes in the `/etc/cinder/cinder.conf` file.

Minimum configuration

Here is a sample of minimum back-end configuration. See the following sections for the detail of each option. Set `storage_protocol = iscsi` if iSCSI protocol is used.

```
[DEFAULT]
enabled_backends = vnx_array1

[vnx_array1]
san_ip = 10.10.72.41
san_login = sysadmin
san_password = sysadmin
naviseccli_path = /opt/Navisphere/bin/naviseccli
volume_driver = cinder.volume.drivers.dell_emc.vnx.driver.VNXDriver
initiator_auto_registration = True
storage_protocol = fc
```

Multiple back-end configuration

Here is a sample of a minimum back-end configuration. See following sections for the detail of each option. Set `storage_protocol = iscsi` if iSCSI protocol is used.

```
[DEFAULT]
enabled_backends = backendA, backendB

[backendA]
storage_vnx_pool_names = Pool_01_SAS, Pool_02_FLASH
san_ip = 10.10.72.41
storage_vnx_security_file_dir = /etc/secfile/array1
naviseccli_path = /opt/Navisphere/bin/naviseccli
volume_driver = cinder.volume.drivers.dell_emc.vnx.driver.VNXDriver
initiator_auto_registration = True
storage_protocol = fc

[backendB]
storage_vnx_pool_names = Pool_02_SAS
san_ip = 10.10.26.101
san_login = username
san_password = password
naviseccli_path = /opt/Navisphere/bin/naviseccli
volume_driver = cinder.volume.drivers.dell_emc.vnx.driver.VNXDriver
initiator_auto_registration = True
storage_protocol = fc
```

The value of the option `storage_protocol` can be either `fc` or `iscsi`, which is case insensitive.

For more details on multiple back ends, see *Configure multiple-storage back ends*.

Required configurations

IP of the VNX Storage Processors

Specify SP A or SP B IP to connect:

```
san_ip = <IP of VNX Storage Processor>
```

VNX login credentials

There are two ways to specify the credentials.

- Use plain text username and password.

  Supply for plain username and password:

  ```
  san_login = <VNX account with administrator role>
  san_password = <password for VNX account>
  storage_vnx_authentication_type = global
  ```

  Valid values for `storage_vnx_authentication_type` are: `global` (default), `local`, and `ldap`.

- Use Security file.

  This approach avoids the plain text password in your cinder configuration file. Supply a security file as below:

  ```
  storage_vnx_security_file_dir = <path to security file>
  ```

Check Unisphere CLI user guide or *Authenticate by security file* for how to create a security file.

Path to your Unisphere CLI

Specify the absolute path to your naviseccli:

```
naviseccli_path = /opt/Navisphere/bin/naviseccli
```

Drivers storage protocol

- For the FC Driver, add the following option:

  ```
  volume_driver = cinder.volume.drivers.dell_emc.vnx.driver.VNXDriver
  storage_protocol = fc
  ```

- For iSCSI Driver, add the following option:

  ```
  volume_driver = cinder.volume.drivers.dell_emc.vnx.driver.VNXDriver
  storage_protocol = iscsi
  ```

Optional configurations

VNX pool names

Specify the list of pools to be managed, separated by commas. They should already exist in VNX.

```
storage_vnx_pool_names = pool 1, pool 2
```

If this value is not specified, all pools of the array will be used.

Initiator auto registration

When `initiator_auto_registration` is set to `True`, the driver will automatically register initiators to all working target ports of the VNX array during volume attaching (The driver will skip those initiators that have already been registered) if the option `io_port_list` is not specified in the `cinder.conf` file.

If the user wants to register the initiators with some specific ports but not register with the other ports, this functionality should be disabled.

When a comma-separated list is given to `io_port_list`, the driver will only register the initiator to the ports specified in the list and only return target port(s) which belong to the target ports in the `io_port_list` instead of all target ports.

- Example for FC ports:

  ```
  io_port_list = a-1,B-3
  ```

  `a` or `B` is *Storage Processor*, number `1` and `3` are *Port ID*.

- Example for iSCSI ports:

  ```
  io_port_list = a-1-0,B-3-0
  ```

  `a` or `B` is *Storage Processor*, the first numbers `1` and `3` are *Port ID* and the second number `0` is *Virtual Port ID*

Note

- Rather than de-registered, the registered ports will be simply bypassed whatever they are in `io_port_list` or not.
- The driver will raise an exception if ports in `io_port_list` do not exist in VNX during startup.

Force delete volumes in storage group

Some `available` volumes may remain in storage group on the VNX array due to some OpenStack timeout issue. But the VNX array do not allow the user to delete the volumes which are in storage group. Option `force_delete_lun_in_storagegroup` is introduced to allow the user to delete the `available` volumes in this tricky situation.

When `force_delete_lun_in_storagegroup` is set to `True` in the back-end section, the driver will move the volumes out of the storage groups and then delete them if the user tries to delete the volumes that remain in the storage group on the VNX array.

The default value of `force_delete_lun_in_storagegroup` is `True`.

Over subscription in thin provisioning

Over subscription allows that the sum of all volumes capacity (provisioned capacity) to be larger than the pools total capacity.

`max_over_subscription_ratio` in the back-end section is the ratio of provisioned capacity over total capacity.

The default value of `max_over_subscription_ratio` is 20.0, which means the provisioned capacity can be 20 times of the total capacity. If the value of this ratio is set larger than 1.0, the provisioned capacity can exceed the total capacity.

Storage group automatic deletion

For volume attaching, the driver has a storage group on VNX for each compute node hosting the vm instances which are going to consume VNX Block Storage (using compute nodes host name as storage groups name). All the volumes attached to the VM instances in a compute node will be put into the storage group. If `destroy_empty_storage_group` is set to `True`, the driver will remove the empty storage group after its last volume is detached. For data safety, it does not suggest to set `destroy_empty_storage_group=True` unless the VNX is exclusively managed by one Block Storage node because consistent `lock_path` is required for operation synchronization for this behavior.

Initiator auto deregistration

Enabling storage group automatic deletion is the precondition of this function. If `initiator_auto_deregistration` is set to `True` is set, the driver will deregister all FC and iSCSI initiators of the host after its storage group is deleted.

FC SAN auto zoning

The EMC VNX driver supports FC SAN auto zoning when `ZoneManager` is configured and `zoning_mode` is set to `fabric` in `cinder.conf`. For ZoneManager configuration, refer to *Fibre Channel Zone Manager*.

Volume number threshold

In VNX, there is a limitation on the number of pool volumes that can be created in the system. When the limitation is reached, no more pool volumes can be created even if there is remaining capacity in the storage pool. In other words, if the scheduler dispatches a volume creation request to a back end that has free capacity but reaches the volume limitation, the creation fails.

The default value of `check_max_pool_luns_threshold` is `False`. When `check_max_pool_luns_threshold=True`, the pool-based back end will check the limit and will report 0 free capacity to the scheduler if the limit is reached. So the scheduler will be able to skip this kind of pool-based back end that runs out of the pool volume number.

Note

From Queens, `check_max_pool_luns_threshold` is obsolete. And the behavior is like where `check_max_pool_luns_threshold` is set to `True`.

iSCSI initiators

`iscsi_initiators` is a dictionary of IP addresses of the iSCSI initiator ports on OpenStack compute and block storage nodes which want to connect to VNX via iSCSI. If this option is configured, the driver will leverage this information to find an accessible iSCSI target portal for the initiator when attaching volume. Otherwise, the iSCSI target portal will be chosen in a relative random way.

Note

This option is only valid for iSCSI driver.

Here is an example. VNX will connect `host1` with `10.0.0.1` and `10.0.0.2`. And it will connect `host2` with `10.0.0.3`.

The key name (`host1` in the example) should be the output of **hostname** command.

```
iscsi_initiators = {"host1":["10.0.0.1", "10.0.0.2"],"host2":["10.0.0.3"]}
```

Default timeout

Specify the timeout in minutes for operations like LUN migration, LUN creation, etc. For example, LUN migration is a typical long running operation, which depends on the LUN size and the load of the array. An upper bound in the specific deployment can be set to avoid unnecessary long wait.

The default value for this option is `infinite`.

```
default_timeout = 60
```

Max LUNs per storage group

The `max_luns_per_storage_group` specify the maximum number of LUNs in a storage group. Default value is 255. It is also the maximum value supported by VNX.

Ignore pool full threshold

If `ignore_pool_full_threshold` is set to `True`, driver will force LUN creation even if the full threshold of pool is reached. Default to `False`.

Default value for async migration

Option `vnx_async_migrate` is used to set the default value of async migration for the backend. The default value of this option is *True* if it isnt set in `cinder.conf` to preserve compatibility. If `async_migrate` is not set in metadata of volume, the value of this option will be used. Otherwise, `async_migrate` value in metadata will override the value of this option. For more detail, refer to *asynchronous migration support*.

Extra spec options

Extra specs are used in volume types created in Block Storage as the preferred property of the volume.

The Block Storage scheduler will use extra specs to find the suitable back end for the volume and the Block Storage driver will create the volume based on the properties specified by the extra spec.

Use the following command to create a volume type:

```
$ openstack volume type create demoVolumeType
```

Use the following command to update the extra spec of a volume type:

```
$ openstack volume type set --property provisioning:type=thin --property␣
↪thick_provisioning_support='<is> True' demoVolumeType
```

The following sections describe the VNX extra keys.

Provisioning type

- Key: `provisioning:type`
- Possible Values:
  - `thick`

    Volume is fully provisioned.

    Run the following commands to create a `thick` volume type:

    ```
    $ openstack volume type create ThickVolumeType
    $ openstack volume type set --property provisioning:type=thick --
    ↪property thick_provisioning_support='<is> True' ThickVolumeType
    ```

  - `thin`

    Volume is virtually provisioned.

    Run the following commands to create a `thin` volume type:

    ```
    $ openstack volume type create ThinVolumeType
    $ openstack volume type set --property provisioning:type=thin --
    ↪property thin_provisioning_support='<is> True' ThinVolumeType
    ```

  - `deduplicated`

    Volume is `thin` and deduplication is enabled. The administrator shall go to VNX to configure the system level deduplication settings. To create a deduplicated volume, the VNX Deduplication license must be activated on VNX, and specify `deduplication_support=True` to let Block Storage scheduler find the proper volume back end.

    Run the following commands to create a `deduplicated` volume type:

    ```
    $ openstack volume type create DeduplicatedVolumeType
    $ openstack volume type set --property␣
    ↪provisioning:type=deduplicated --property deduplicated_support='
    ↪<is> True' DeduplicatedVolumeType
    ```

  - `compressed`

    Volume is `thin` and compression is enabled. The administrator shall go to the VNX to configure the system level compression settings. To create a compressed volume, the VNX Compression license must be activated on VNX, and use `compression_support=True` to let Block Storage scheduler find a volume back end. VNX does not support creating snapshots on a compressed volume.

    Run the following commands to create a `compressed` volume type:

```
$ openstack volume type create CompressedVolumeType
$ openstack volume type set --property provisioning:type=compressed -
↪-property compression_support='<is> True' CompressedVolumeType
```

- Default: `thick`

> **Note**
>
> `provisioning:type` replaces the old spec key `storagetype:provisioning`. The latter one is obsolete since the *Mitaka* release.

Storage tiering support

- Key: `storagetype:tiering`
- Possible values:
  - `StartHighThenAuto`
  - `Auto`
  - `HighestAvailable`
  - `LowestAvailable`
  - `NoMovement`
- Default: `StartHighThenAuto`

VNX supports fully automated storage tiering which requires the FAST license activated on the VNX. The OpenStack administrator can use the extra spec key `storagetype:tiering` to set the tiering policy of a volume and use the key `fast_support='<is> True'` to let Block Storage scheduler find a volume back end which manages a VNX with FAST license activated. Here are the five supported values for the extra spec key `storagetype:tiering`:

Run the following commands to create a volume type with tiering policy:

```
$ openstack volume type create ThinVolumeOnAutoTier
$ openstack volume type set --property provisioning:type=thin --property␣
↪storagetype:tiering=Auto --property fast_support='<is> True'␣
↪ThinVolumeOnAutoTier
```

> **Note**
>
> The tiering policy cannot be applied to a deduplicated volume. Tiering policy of the deduplicated LUN align with the settings of the pool.

FAST cache support

- Key: `fast_cache_enabled`
- Possible values:
  - `True`

  - False

- Default: False

VNX has FAST Cache feature which requires the FAST Cache license activated on the VNX. Volume will be created on the backend with FAST cache enabled when <is> True is specified.

Pool name

- Key: pool_name
- Possible values: name of the storage pool managed by cinder
- Default: None

If the user wants to create a volume on a certain storage pool in a back end that manages multiple pools, a volume type with a extra spec specified storage pool should be created first, then the user can use this volume type to create the volume.

Run the following commands to create the volume type:

```
$ openstack volume type create HighPerf
$ openstack volume type set --property pool_name=Pool_02_SASFLASH --property␣
↪volume_backend_name=vnx_41 HighPerf
```

Obsolete extra specs

Note

DO NOT use the following obsolete extra spec keys:

- storagetype:provisioning
- storagetype:pool

Force detach

The user could use *os-force_detach* action to detach a volume from all its attached hosts. For more detail, please refer to https://docs.openstack.org/api-ref/block-storage/v3/?expanded=force-detach-a-volume-detail#force-detach-a-volume

Advanced features

Snap copy

- Metadata Key: snapcopy
- Possible Values:
  - True or true
  - False or false
- Default: *False*

VNX driver supports snap copy which accelerates the process for creating a copied volume.

By default, the driver will use *asynchronous migration support*, which will start a VNX migration session. When snap copy is used, driver creates a snapshot and mounts it as a volume for the 2 kinds of operations which will be instant even for large volumes.

To enable this functionality, append `--metadata snapcopy=True` when creating cloned volume or creating volume from snapshot.

```
$ cinder create --source-volid <source-void> --name "cloned_volume" --
↪metadata snapcopy=True
```

Or

```
$ cinder create --snapshot-id <snapshot-id> --name "vol_from_snapshot" --
↪metadata snapcopy=True
```

The newly created volume is a snap copy instead of a full copy. If a full copy is needed, retype or migrate can be used to convert the snap-copy volume to a full-copy volume which may be time-consuming.

You can determine whether the volume is a snap-copy volume or not by showing its metadata. If the `snapcopy` in metadata is `True` or `true`, the volume is a snap-copy volume. Otherwise, it is a full-copy volume.

```
$ cinder metadata-show <volume>
```

Constraints

- The number of snap-copy volumes created from a single source volume is limited to 255 at one point in time.
- The source volume which has snap-copy volume can not be deleted or migrated.
- snapcopy volume will be change to full-copy volume after host-assisted or storage-assisted migration.
- snapcopy volume can not be added to consisgroup because of VNX limitation.

Efficient non-disruptive volume backup

The default implementation in Block Storage for non-disruptive volume backup is not efficient since a cloned volume will be created during backup.

The approach of efficient backup is to create a snapshot for the volume and connect this snapshot (a mount point in VNX) to the Block Storage host for volume backup. This eliminates migration time involved in volume clone.

Constraints

- Backup creation for a snap-copy volume is not allowed if the volume status is `in-use` since snapshot cannot be taken from this volume.

Configurable migration rate

VNX cinder driver is leveraging the LUN migration from the VNX. LUN migration is involved in cloning, migrating, retyping, and creating volume from snapshot. When admin set `migrate_rate` in volumes `metadata`, VNX driver can start migration with specified rate. The available values for the `migrate_rate` are `high`, `asap`, `low` and `medium`.

The following is an example to set `migrate_rate` to `asap`:

```
$ cinder metadata <volume-id> set migrate_rate=asap
```

After set, any cinder volume operations involving VNX LUN migration will take the value as the migration rate. To restore the migration rate to default, unset the metadata as following:

```
$ cinder metadata <volume-id> unset migrate_rate
```

Note

Do not use the `asap` migration rate when the system is in production, as the normal host I/O may be interrupted. Use asap only when the system is offline (free of any host-level I/O).

Replication v2.1 support

Cinder introduces Replication v2.1 support in Mitaka, it supports fail-over and fail-back replication for specific back end. In VNX cinder driver, **MirrorView** is used to set up replication for the volume.

To enable this feature, you need to set configuration in `cinder.conf` as below:

```
replication_device = backend_id:<secondary VNX serial number>,
                     san_ip:192.168.1.2,
                     san_login:admin,
                     san_password:admin,
                     naviseccli_path:/opt/Navisphere/bin/naviseccli,
                     storage_vnx_authentication_type:global,
                     storage_vnx_security_file_dir:
```

Currently, only synchronized mode **MirrorView** is supported, and one volume can only have 1 secondary storage system. Therefore, you can have only one `replication_device` presented in driver configuration section.

To create a replication enabled volume, you need to create a volume type:

```
$ openstack volume type create replication-type
$ openstack volume type set --property replication_enabled="<is> True" ␣
↪replication-type
```

And then create volume with above volume type:

```
$ openstack volume create replication-volume --type replication-type --size 1
```

Supported operations

- Create volume
- Create cloned volume
- Create volume from snapshot
- Fail-over volume:

```
$ cinder failover-host --backend_id <secondary VNX serial number>
↪<hostname>
```

- Fail-back volume:

```
$ cinder failover-host --backend_id default <hostname>
```

Requirements

- 2 VNX systems must be in same domain.
- For iSCSI MirrorView, user needs to setup iSCSI connection before enable replication in Cinder.
- For FC MirrorView, user needs to zone specific FC ports from 2 VNX system together.
- MirrorView Sync enabler(**MirrorView/S**) installed on both systems.
- Write intent log enabled on both VNX systems.

For more information on how to configure, please refer to: MirrorView-Knowledgebook:-Releases-30--33

Asynchronous migration support

VNX Cinder driver now supports asynchronous migration during volume cloning.

The driver now using asynchronous migration when creating a volume from source as the default cloning method. The driver will return immediately after the migration session starts on the VNX, which dramatically reduces the time before a volume is available for use.

To disable this feature, user needs to do any one of below actions:

- Configure `vnx_async_migrate = False` for the backend in `cinder.conf`, then restart Cinder services.
- Add `--metadata async_migrate=False` when creating new volume from source.

Be aware, `async_migrate` in metadata overrides the option `vnx_async_migrate` when both are set.

Constraints

- Before the migration finishes, snapshots cannot be created from the source volume, which could affect subsequent clones from the same source volume. The typical affected use case is that creating volume-2 via cloning volume-1 immediately after creating volume-1 via cloning volume-0. To achieve so, users are advised to take any one of below actions:

  1) wait for the first clone finishing, or
  2) create volume-2 via cloning volume-0 instead of volume-1, or
  3) create volume-1 with `--metadata async_migrate=False`.

Best practice

Multipath setup

Enabling multipath volume access is recommended for robust data access. The major configuration includes:

1. Install `multipath-tools`, `sysfsutils` and `sg3-utils` on the nodes hosting compute and `cinder-volume` services. Check the operating system manual for the system distribution for specific installation steps. For Red Hat based distributions, they should be `device-mapper-multipath`, `sysfsutils` and `sg3_utils`.
2. Specify `use_multipath_for_image_xfer=true` in the `cinder.conf` file for each FC/iSCSI back end.
3. Specify `volume_use_multipath=True` in `libvirt` section of the `nova.conf` file. This option is valid for both iSCSI and FC driver. In versions prior to Newton, the option was called `iscsi_use_multipath`.

For multipath-tools, here is an EMC recommended sample of `/etc/multipath.conf` file.

`user_friendly_names` is not specified in the configuration and thus it will take the default value `no`. It is not recommended to set it to `yes` because it may fail operations such as VM live migration.

```
blacklist {
    # Skip the files under /dev that are definitely not FC/iSCSI devices
    # Different system may need different customization
    devnode "^(ram|raw|loop|fd|md|dm-|sr|scd|st)[0-9]*"
    devnode "^hd[a-z][0-9]*"
    devnode "^cciss!c[0-9]d[0-9]*[p[0-9]*]"

    # Skip LUNZ device from VNX
    device {
        vendor "DGC"
        product "LUNZ"
        }
}

defaults {
    user_friendly_names no
    flush_on_last_del yes
}

devices {
    # Device attributed for EMC CLARiiON and VNX series ALUA
    device {
        vendor "DGC"
        product ".*"
        product_blacklist "LUNZ"
        path_grouping_policy group_by_prio
        path_selector "round-robin 0"
        path_checker emc_clariion
        features "1 queue_if_no_path"
        hardware_handler "1 alua"
        prio alua
        failback immediate
    }
}
```

Note

When multipath is used in OpenStack, multipath faulty devices may come out in Nova-Compute nodes due to different issues (Bug 1336683 is a typical example).

A solution to completely avoid faulty devices has not been found yet. `faulty_device_cleanup.py` mitigates this issue when VNX iSCSI storage is used. Cloud administrators can deploy the script in all Nova-Compute nodes and use a CRON job to run the script on each Nova-Compute node periodically so that faulty devices will not stay too long. Refer to: VNX faulty device cleanup for detailed usage and the script.

Restrictions and limitations

iSCSI port cache

EMC VNX iSCSI driver caches the iSCSI ports information, so that the user should restart the `cinder-volume` service or wait for seconds (which is configured by `periodic_interval` in the `cinder.conf` file) before any volume attachment operation after changing the iSCSI port configurations. Otherwise the attachment may fail because the old iSCSI port configurations were used.

No extending for volume with snapshots

VNX does not support extending the thick volume which has a snapshot. If the user tries to extend a volume which has a snapshot, the status of the volume would change to `error_extending`.

Limitations for deploying cinder on computer node

It is not recommended to deploy the driver on a compute node if `cinder upload-to-image --force True` is used against an in-use volume. Otherwise, `cinder upload-to-image --force True` will terminate the data access of the vm instance to the volume.

Storage group with host names in VNX

When the driver notices that there is no existing storage group that has the host name as the storage group name, it will create the storage group and also add the compute nodes or Block Storage nodes registered initiators into the storage group.

If the driver notices that the storage group already exists, it will assume that the registered initiators have also been put into it and skip the operations above for better performance.

It is recommended that the storage administrator does not create the storage group manually and instead relies on the driver for the preparation. If the storage administrator needs to create the storage group manually for some special requirements, the correct registered initiators should be put into the storage group as well (otherwise the following volume attaching operations will fail).

EMC storage-assisted volume migration

EMC VNX driver supports storage-assisted volume migration, when the user starts migrating with `cinder migrate --force-host-copy False <volume_id> <host>` or `cinder migrate <volume_id> <host>`, cinder will try to leverage the VNXs native volume migration functionality.

In following scenarios, VNX storage-assisted volume migration will not be triggered:

- `in-use` volume migration between back ends with different storage protocol, for example, FC and iSCSI.
- Volume is to be migrated across arrays.

Appendix

Authenticate by security file

VNX credentials are necessary when the driver connects to the VNX system. Credentials in `global`, `local` and `ldap` scopes are supported. There are two approaches to provide the credentials.

The recommended one is using the Navisphere CLI security file to provide the credentials which can get rid of providing the plain text credentials in the configuration file. Following is the instruction on how to do this.

1. Find out the Linux user id of the `cinder-volume` processes. Assuming the `cinder-volume` service is running by the account `cinder`.
2. Run `su` as root user.
3. In `/etc/passwd` file, change `cinder:x:113:120::/var/lib/cinder:/bin/false` to `cinder:x:113:120::/var/lib/cinder:/bin/bash` (This temporary change is to make step 4 work.)
4. Save the credentials on behalf of `cinder` user to a security file (assuming the array credentials are `admin/admin` in `global` scope). In the command below, the `-secfilepath` switch is used to specify the location to save the security file.

```
# su -l cinder -c \
  '/opt/Navisphere/bin/naviseccli -AddUserSecurity -user admin -password␣
↪admin -scope 0 -secfilepath <location>'
```

5. Change `cinder:x:113:120::/var/lib/cinder:/bin/bash` back to `cinder:x:113:120::/var/lib/cinder:/bin/false` in `/etc/passwd` file.
6. Remove the credentials options `san_login`, `san_password` and `storage_vnx_authentication_type` from `cinder.conf` file. (normally it is `/etc/cinder/cinder.conf` file). Add option `storage_vnx_security_file_dir` and set its value to the directory path of your security file generated in the above step. Omit this option if `-secfilepath` is not used in the above step.
7. Restart the `cinder-volume` service to validate the change.

Register FC port with VNX

This configuration is only required when `initiator_auto_registration=False`.

To access VNX storage, the Compute nodes should be registered on VNX first if initiator auto registration is not enabled.

To perform `Copy Image to Volume` and `Copy Volume to Image` operations, the nodes running the `cinder-volume` service (Block Storage nodes) must be registered with the VNX as well.

The steps mentioned below are for the compute nodes. Follow the same steps for the Block Storage nodes also (The steps can be skipped if initiator auto registration is enabled).

1. Assume `20:00:00:24:FF:48:BA:C2:21:00:00:24:FF:48:BA:C2` is the WWN of a FC initiator port name of the compute node whose host name and IP are `myhost1` and `10.10.61.1`. Register `20:00:00:24:FF:48:BA:C2:21:00:00:24:FF:48:BA:C2` in Unisphere:
2. Log in to *Unisphere*, go to *FNM0000000000* > *Hosts* > *Initiators*.
3. Refresh and wait until the initiator `20:00:00:24:FF:48:BA:C2:21:00:00:24:FF:48:BA:C2` with SP Port `A-1` appears.
4. Click the *Register* button, select *CLARiiON/VNX* and enter the host name (which is the output of the **`hostname`** command) and IP address:
   - Hostname: `myhost1`
   - IP: `10.10.61.1`
   - Click *Register*.
5. Then host `10.10.61.1` will appear under *Hosts* > *Host List* as well.
6. Register the `wwn` with more ports if needed.

Register iSCSI port with VNX

This configuration is only required when `initiator_auto_registration=False`.

To access VNX storage, the compute nodes should be registered on VNX first if initiator auto registration is not enabled.

To perform `Copy Image to Volume` and `Copy Volume to Image` operations, the nodes running the `cinder-volume` service (Block Storage nodes) must be registered with the VNX as well.

The steps mentioned below are for the compute nodes. Follow the same steps for the Block Storage nodes also (The steps can be skipped if initiator auto registration is enabled).

1. On the compute node with IP address `10.10.61.1` and host name `myhost1`, execute the following commands (assuming `10.10.61.35` is the iSCSI target):
   1. Start the iSCSI initiator service on the node:
      ```
      # /etc/init.d/open-iscsi start
      ```
   2. Discover the iSCSI target portals on VNX:
      ```
      # iscsiadm -m discovery -t st -p 10.10.61.35
      ```
   3. Change directory to `/etc/iscsi` :
      ```
      # cd /etc/iscsi
      ```
   4. Find out the `iqn` of the node:
      ```
      # more initiatorname.iscsi
      ```
2. Log in to *VNX* from the compute node using the target corresponding to the SPA port:
   ```
   # iscsiadm -m node -T iqn.1992-04.com.emc:cx.apm01234567890.a0 -p 10.10.
   ↪61.35 -l
   ```

3. Assume `iqn.1993-08.org.debian:01:1a2b3c4d5f6g` is the initiator name of the compute node. Register `iqn.1993-08.org.debian:01:1a2b3c4d5f6g` in Unisphere:

   1. Log in to *Unisphere*, go to *FNM0000000000 > Hosts > Initiators*.

   2. Refresh and wait until the initiator `iqn.1993-08.org.debian:01:1a2b3c4d5f6g` with SP Port `A-8v0` appears.

   3. Click the *Register* button, select *CLARiiON/VNX* and enter the host name (which is the output of the **hostname** command) and IP address:

      - Hostname: `myhost1`
      - IP: `10.10.61.1`
      - Click *Register*.

   4. Then host `10.10.61.1` will appear under *Hosts > Host List* as well.

4. Log out *iSCSI* on the node:

```
# iscsiadm -m node -u
```

5. Log in to *VNX* from the compute node using the target corresponding to the SPB port:

```
# iscsiadm -m node -T iqn.1992-04.com.emc:cx.apm01234567890.b8 -p 10.10.
↪61.36 -l
```

6. In `Unisphere`, register the initiator with the SPB port.

7. Log out *iSCSI* on the node:

```
# iscsiadm -m node -u
```

8. Register the `iqn` with more ports if needed.

Dell XtremIO Block Storage driver

The high performance XtremIO All Flash Array (AFA) offers Block Storage services to OpenStack. Using the driver, OpenStack Block Storage hosts can connect to an XtremIO Storage cluster.

This section explains how to configure and connect the block storage nodes to an XtremIO storage cluster.

Support matrix

XtremIO version 4.x is supported.

Supported operations

- Create, delete, clone, attach, and detach volumes.
- Create and delete volume snapshots.
- Create a volume from a snapshot.
- Copy an image to a volume.
- Copy a volume to an image.
- Extend a volume.

- Manage and unmanage a volume.
- Manage and unmanage a snapshot.
- Get volume statistics.
- Create, modify, delete, and list consistency groups.
- Create, modify, delete, and list snapshots of consistency groups.
- Create consistency group from consistency group or consistency group snapshot.
- Volume Migration (host assisted)

XtremIO Block Storage driver configuration

Edit the `cinder.conf` file by adding the configuration below under the [DEFAULT] section of the file in case of a single back end or under a separate section in case of multiple back ends (for example [XTREMIO]). The configuration file is usually located under the following path `/etc/cinder/cinder.conf`.

Table 22: Description of XtremIO configuration options

Configuration option = Default value	Description
`xtremio_arr` = `5`	(Integer) Number of retries in case array is busy
`xtremio_arr` = `5`	(Integer) Interval between retries in case array is busy
`xtremio_cle` = `False`	(Boolean) Should the driver remove initiator groups with no volumes after the last connection was terminated. Since the behavior till now was to leave the IG be, we default to False (not deleting IGs without connected volumes); setting this parameter to True will remove any IG after terminating its connection to the last volume.
`xtremio_clu` = <>	(String) XMS cluster id in multi-cluster environment
`xtremio_por` = `[]`	(List of String) Allowed ports. Comma separated list of XtremIO iSCSI IPs or FC WWNs (ex. 58:cc:f0:98:49:22:07:02) to be used. If option is not set all ports are allowed.
`xtremio_vol` = `100`	(Integer) Number of volumes created from each cached glance image

For a configuration example, refer to the configuration *Configuration example*.

XtremIO driver name

Configure the driver name by setting the following parameter in the `cinder.conf` file:

- For iSCSI:

```
volume_driver = cinder.volume.drivers.dell_emc.xtremio.XtremIOISCSIDriver
```

- For Fibre Channel:

```
volume_driver = cinder.volume.drivers.dell_emc.xtremio.
↪XtremIOFibreChannelDriver
```

XtremIO management server (XMS) IP

To retrieve the management IP, use the **show-xms** CLI command.

Configure the management IP by adding the following parameter:

```
san_ip = XMS Management IP
```

XtremIO cluster name

In XtremIO version 4.0, a single XMS can manage multiple cluster back ends. In such setups, the administrator is required to specify the cluster name (in addition to the XMS IP). Each cluster must be defined as a separate back end.

To retrieve the cluster name, run the **show-clusters** CLI command.

Configure the cluster name by adding the following parameter:

```
xtremio_cluster_name = Cluster-Name
```

Note

When a single cluster is managed in XtremIO version 4.0, the cluster name is not required.

XtremIO user credentials

OpenStack Block Storage requires an XtremIO XMS user with administrative privileges. XtremIO recommends creating a dedicated OpenStack user account that holds an administrative user role.

Refer to the XtremIO User Guide for details on user account management.

Create an XMS account using either the XMS GUI or the **add-user-account** CLI command.

Configure the user credentials by adding the following parameters:

```
san_login = XMS username
san_password = XMS username password
```

Multiple back ends

Configuring multiple storage back ends enables you to create several back-end storage solutions that serve the same OpenStack Compute resources.

When a volume is created, the scheduler selects the appropriate back end to handle the request, according to the specified volume type.

Setting thin provisioning and multipathing parameters

To support thin provisioning and multipathing in the XtremIO Array, the following parameters from the Nova and Cinder configuration files should be modified as follows:

- Thin Provisioning

  All XtremIO volumes are thin provisioned. The default value of 20 should be maintained for the `max_over_subscription_ratio` parameter.

  The `use_cow_images` parameter in the `nova.conf` file should be set to `False` as follows:

  ```
  use_cow_images = False
  ```

- Multipathing

  The `use_multipath_for_image_xfer` parameter in the `cinder.conf` file should be set to `True` for each backend or in `[backend_defaults]` section as a common configuration for all backends.

  ```
  use_multipath_for_image_xfer = True
  ```

Image service optimization

Limit the number of copies (XtremIO snapshots) taken from each image cache.

```
xtremio_volumes_per_glance_cache = 100
```

The default value is `100`. A value of `0` ignores the limit and defers to the array maximum as the effective limit.

SSL certification

To enable SSL certificate validation, modify the following option in the `cinder.conf` file:

```
driver_ssl_cert_verify = true
```

By default, SSL certificate validation is disabled.

To specify a non-default path to `CA_Bundle` file or directory with certificates of trusted CAs:

```
driver_ssl_cert_path = Certificate path
```

Configuring CHAP

The XtremIO Block Storage driver supports CHAP initiator authentication and discovery.

If CHAP initiator authentication is required, set the CHAP Authentication mode to initiator.

To set the CHAP initiator mode using CLI, run the following XMCLI command:

```
$ modify-chap chap-authentication-mode=initiator
```

If CHAP initiator discovery is required, set the CHAP discovery mode to initiator.

To set the CHAP initiator discovery mode using CLI, run the following XMCLI command:

```
$ modify-chap chap-discovery-mode=initiator
```

The CHAP initiator modes can also be set via the XMS GUI.

Refer to XtremIO User Guide for details on CHAP configuration via GUI and CLI.

The CHAP initiator authentication and discovery credentials (username and password) are generated automatically by the Block Storage driver. Therefore, there is no need to configure the initial CHAP credentials manually in XMS.

Configuring ports filtering

The XtremIO Block Storage driver supports ports filtering to define a list of iSCSI IP-addresses or FC WWNs which will be used to attach volumes. If option is not set all ports are allowed.

```
xtremio_ports = iSCSI IPs or FC WWNs
```

Configuration example

You can update the `cinder.conf` file by editing the necessary parameters as follows:

```
[Default]
enabled_backends = XtremIO

[XtremIO]
volume_driver = cinder.volume.drivers.dell_emc.xtremio.
↪XtremIOFibreChannelDriver
san_ip = XMS_IP
xtremio_cluster_name = Cluster01
xtremio_ports = 21:00:00:24:ff:57:b2:36,21:00:00:24:ff:57:b2:55
san_login = XMS_USER
san_password = XMS_PASSWD
volume_backend_name = XtremIOAFA
```

Dell SC Series Fibre Channel and iSCSI drivers

The Dell Storage Center volume driver interacts with configured Storage Center arrays.

The Dell Storage Center driver manages a Storage Center array via the Dell Storage Manager (DSM) Data Collector or by directly connecting to the Storage Center at the cost of replication and Live Volume functionality. Also note that the directly connecting to the Storage Center is only supported with Storage Center OS 7.1.1 or later. Any version of Storage Center OS supported by DSM is supported if connecting via the Data Collector.

Driver configuration settings and Storage Center options are defined in the `cinder.conf` file.

Prerequisites:

- Storage Center OS version 7.1.1 or later and OpenStack Ocata or later must be used if connecting directly to the Storage Center.
- Dell Storage Manager 2015 R1 or later if connecting through DSM.

Supported operations

The Dell Storage Center volume driver provides the following Cinder volume operations:

- Create, delete, attach (map), and detach (unmap) volumes.
- Create, list, and delete volume snapshots.
- Create a volume from a snapshot.
- Copy an image to a volume.
- Copy a volume to an image.
- Clone a volume.
- Extend a volume.
- Create, delete, list and update a consistency group.
- Create, delete, and list consistency group snapshots.
- Manage an existing volume.
- Replication (Requires DSM.)
- Failover-host for replicated back ends. (Requires DSM.)
- Create a replication using Live Volume. (Requires DSM.)

Extra spec options

Volume type extra specs can be used to enable a variety of Dell Storage Center options. Selecting Storage Profiles, Replay Profiles, enabling replication, replication options including Live Volume and Active Replay replication. (Replication options are available when connected via DSM.)

Storage Profiles control how Storage Center manages volume data. For a given volume, the selected Storage Profile dictates which disk tier accepts initial writes, as well as how data progression moves data between tiers to balance performance and cost. Predefined Storage Profiles are the most effective way to manage data in Storage Center.

By default, if no Storage Profile is specified in the volume extra specs, the default Storage Profile for the user account configured for the Block Storage driver is used. The extra spec key `storagetype:storageprofile` with the value of the name of the Storage Profile on the Storage Center can be set to allow to use Storage Profiles other than the default.

For ease of use from the command line, spaces in Storage Profile names are ignored. As an example, here is how to define two volume types using the `High Priority` and `Low Priority` Storage Profiles:

```
$ openstack volume type create "GoldVolumeType"
$ openstack volume type set --property␣
↪storagetype:storageprofile=highpriority "GoldVolumeType"
$ openstack volume type create "BronzeVolumeType"
$ openstack volume type set --property storagetype:storageprofile=lowpriority
↪"BronzeVolumeType"
```

Replay Profiles control how often the Storage Center takes a replay of a given volume and how long those replays are kept. The default profile is the `daily` profile that sets the replay to occur once a day and to persist for one week.

The extra spec key `storagetype:replayprofiles` with the value of the name of the Replay Profile or profiles on the Storage Center can be set to allow to use Replay Profiles other than the default `daily` profile.

As an example, here is how to define a volume type using the `hourly` Replay Profile and another specifying both `hourly` and the default `daily` profile:

```
$ openstack volume type create "HourlyType"
$ openstack volume type set --property storagetype:replayprofile=hourly
↪"HourlyType"
$ openstack volume type create "HourlyAndDailyType"
$ openstack volume type set --property storagetype:replayprofiles=hourly,
↪daily "HourlyAndDailyType"
```

Note the comma separated string for the `HourlyAndDailyType`.

Replication for a given volume type is enabled via the extra spec `replication_enabled`.

To create a volume type that specifies only replication enabled back ends:

```
$ openstack volume type create "ReplicationType"
$ openstack volume type set --property replication_enabled='<is> True'
↪"ReplicationType"
```

Extra specs can be used to configure replication. In addition to the Replay Profiles above, `replication:activereplay` can be set to enable replication of the volumes active replay. And the replication type can be changed to synchronous via the `replication_type` extra spec can be set.

To create a volume type that enables replication of the active replay:

```
$ openstack volume type create "ReplicationType"
$ openstack volume type key --property replication_enabled='<is> True'
↪"ReplicationType"
$ openstack volume type key --property replication:activereplay='<is> True'
↪"ReplicationType"
```

To create a volume type that enables synchronous replication :

```
$ openstack volume type create "ReplicationType"
$ openstack volume type key --property replication_enabled='<is> True'
↪"ReplicationType"
$ openstack volume type key --property replication_type='<is> sync'
↪"ReplicationType"
```

To create a volume type that enables replication using Live Volume:

```
$ openstack volume type create "ReplicationType"
$ openstack volume type key --property replication_enabled='<is> True'
↪"ReplicationType"
$ openstack volume type key --property replication:livevolume='<is> True'
↪"ReplicationType"
```

If QOS options are enabled on the Storage Center they can be enabled via extra specs. The name of the Volume QOS can be specified via the `storagetype:volumeqos` extra spec. Likewise the name of the

Group QOS to use can be specified via the `storagetype:groupqos` extra spec. Volumes created with these extra specs set will be added to the specified QOS groups.

To create a volume type that sets both Volume and Group QOS:

```
$ openstack volume type create "StorageCenterQOS"
$ openstack volume type key --property 'storagetype:volumeqos'='unlimited'
↪"StorageCenterQOS"
$ openstack volume type key --property 'storagetype:groupqos'='limited'
↪"StorageCenterQOS"
```

Data reduction profiles can be specified in the `storagetype:datareductionprofile` extra spec. Available options are None, Compression, and Deduplication. Note that not all options are available on every Storage Center.

To create volume types that support no compression, compression, and deduplication and compression respectively:

```
$ openstack volume type create "NoCompressionType"
$ openstack volume type key --property 'storagetype:datareductionprofile'=
↪'None' "NoCompressionType"
$ openstack volume type create "CompressedType"
$ openstack volume type key --property 'storagetype:datareductionprofile'=
↪'Compression' "CompressedType"
$ openstack volume type create "DedupType"
$ openstack volume type key --property 'storagetype:datareductionprofile'=
↪'Deduplication' "DedupType"
```

Note: The default is no compression.

iSCSI configuration

Use the following instructions to update the configuration file for iSCSI:

```
default_volume_type = delliscsi
enabled_backends = delliscsi

[delliscsi]
# Name to give this storage back-end
volume_backend_name = delliscsi
# The iSCSI driver to load
volume_driver = cinder.volume.drivers.dell_emc.sc.storagecenter_iscsi.
↪SCISCSIDriver
# IP address of the DSM or the Storage Center if attaching directly.
san_ip = 172.23.8.101
# DSM user name
san_login = Admin
# DSM password
san_password = secret
# The Storage Center serial number to use
dell_sc_ssn = 64702

```

(continues on next page)

(continued from previous page)

```
# ==Optional settings==

# The DSM API port
dell_sc_api_port = 3033
# Server folder to place new server definitions
dell_sc_server_folder = devstacksrv
# Volume folder to place created volumes
dell_sc_volume_folder = devstackvol/Cinder
```

Fibre Channel configuration

Use the following instructions to update the configuration file for fibre channel:

```
default_volume_type = dellfc
enabled_backends = dellfc

[dellfc]
# Name to give this storage back-end
volume_backend_name = dellfc
# The FC driver to load
volume_driver = cinder.volume.drivers.dell_emc.sc.storagecenter_fc.SCFCDriver

# IP address of the DSM or the Storage Center if attaching directly.
san_ip = 172.23.8.101
# DSM user name
san_login = Admin
# DSM password
san_password = secret
# The Storage Center serial number to use
dell_sc_ssn = 64702

# ==Optional settings==

# The DSM API port
dell_sc_api_port = 3033
# Server folder to place new server definitions
dell_sc_server_folder = devstacksrv
# Volume folder to place created volumes
dell_sc_volume_folder = devstackvol/Cinder
```

Dual DSM

It is possible to specify a secondary DSM to use in case the primary DSM fails.

Configuration is done through the cinder.conf. Both DSMs have to be configured to manage the same set of Storage Centers for this backend. That means the dell_sc_ssn and any Storage Centers used for replication or Live Volume.

Add network and credential information to the backend to enable Dual DSM.

```
[dell]
# The IP address and port of the secondary DSM.
secondary_san_ip = 192.168.0.102
secondary_sc_api_port = 3033
# Specify credentials for the secondary DSM.
secondary_san_login = Admin
secondary_san_password = secret
```

The driver will use the primary until a failure. At that point it will attempt to use the secondary. It will continue to use the secondary until the volume service is restarted or the secondary fails at which point it will attempt to use the primary.

Note: Requires two DSM Data Collectors.

Replication configuration

Add the following to the back-end specification to specify another Storage Center to replicate to.

```
[dell]
replication_device = target_device_id: 65495, qosnode: cinderqos
```

The `target_device_id` is the SSN of the remote Storage Center and the `qosnode` is the QoS Node setup between the two Storage Centers.

Note that more than one `replication_device` line can be added. This will slow things down, however.

A volume is only replicated if the volume is of a volume-type that has the extra spec `replication_enabled` set to <is> True.

Warning: replication_device requires DSM. If this is on a backend that is directly connected to the Storage Center the driver will not load as it is unable to meet the replication requirement.

Replication notes

This driver supports both standard replication and Live Volume (if supported and licensed). The main difference is that a VM attached to a Live Volume is mapped to both Storage Centers. In the case of a failure of the primary Live Volume still requires a failover-host to move control of the volume to the second controller.

Existing mappings should work and not require the instance to be remapped but it might need to be rebooted.

Live Volume is more resource intensive than replication. One should be sure to plan accordingly.

Failback

The failover-host command is designed for the case where the primary system is not coming back. If it has been executed and the primary has been restored it is possible to attempt a failback.

Simply specify default as the backend_id.

```
$ cinder failover-host cinder@delliscsi --backend_id default
```

Non trivial heavy lifting is done by this command. It attempts to recover as best it can but if things have diverged too far it can only do so much. It is also a one time only command so do not reboot or restart the service in the middle of it.

Failover and failback are significant operations under OpenStack Cinder. Be sure to consult with support before attempting.

Server type configuration

This option allows one to set a default Server OS type to use when creating a server definition on the Dell Storage Center.

When attaching a volume to a node the Dell Storage Center driver creates a server definition on the storage array. This definition includes a Server OS type. The type used by the Dell Storage Center cinder driver is Red Hat Linux 6.x. This is a modern operating system definition that supports all the features of an OpenStack node.

Add the following to the back-end specification to specify the Server OS to use when creating a server definition. The server type used must come from the drop down list in the DSM.

```
[dell]
dell_server_os = 'Red Hat Linux 7.x'
```

Note that this server definition is created once. Changing this setting after the fact will not change an existing definition. The selected Server OS does not have to match the actual OS used on the node.

Excluding a domain

This option excludes a list of Storage Center ISCSI fault domains from the ISCSI properties returned by the initialize_connection call. This only applies to the ISCSI driver.

Add the excluded_domain_ips option into the backend config for several fault domains to be excluded. This option takes a comma separated list of Target IP addresses listed under the fault domain. Older versions of DSM (EM) may list this as the Well Known IP Address.

Note that the `included_domain_ips` takes precedance over `excluded_domain_ips`. When `included_domain_ips` is not an empty list, the option `excluded_domain_ips` is ignored.

Add the following to the back-end specification to exclude the domains at 172.20.25.15 and 172.20.26.15.

```
[dell]
excluded_domain_ips=172.20.25.15, 172.20.26.15, 0:0:0:0:0:ffff:c0a8:15
```

Including domains

This option includes or will whitelist a list of Storage Center ISCSI fault domains from the ISCSI properties returned by the initialize_connection call. This only applies to the ISCSI driver.

Add the `included_domain_ips` option into the backend config for several default domains to be included or whitelisted. This option takes a comma separated list of Target IP addresses listed under the fault domain. Older versions of DSM (EM) may list this as the Well Known IP Address.

Note that the `included_domain_ips` takes precedance over `excluded_domain_ips`. When `included_domain_ips` is not an empty list, the option `excluded_domain_ips` is ignored.

Add the following to the back-end specification to include or whitelist the domains at 172.20.25.15 and 172.20.26.15.

```
[dell]
included_domain_ips=172.20.25.15, 172.20.26.15, 0:0:0:0:0:ffff:c0a8:15
```

Setting Dell SC REST API timeouts

The user can specify timeouts for Dell SC REST API calls.

To set the timeout for ASYNC REST API calls in seconds.

```
[dell]
dell_api_async_rest_timeout=15
```

To set the timeout for SYNC REST API calls in seconds.

```
[dell]
dell_api_sync_rest_timeout=30
```

Generally these should not be set without guidance from Dell support.

Driver options

The following table contains the configuration options specific to the Dell Storage Center volume driver.

Table 23: Description of SC Series configuration options

Configuration option = Default value	Description
dell_api_async_rest_ti = 15	(Integer) Dell SC API async call default timeout in seconds.
dell_api_sync_rest_tim = 30	(Integer) Dell SC API sync call default timeout in seconds.
dell_sc_api_port = 3033	(Port(min=0, max=65535)) Dell API port
dell_sc_server_folder = openstack	(String) Name of the server folder to use on the Storage Center
dell_sc_ssn = 64702	(Integer) Storage Center System Serial Number
dell_sc_verify_cert = False	(Boolean) Enable HTTPS SC certificate verification
dell_sc_volume_folder = openstack	(String) Name of the volume folder to use on the Storage Center
dell_server_os = Red Hat Linux 6.x	(String) Server OS type to use when creating a new server on the Storage Center.
excluded_domain_ips = []	(List of IPAddress) Comma separated Fault Domain IPs to be excluded from iSCSI returns.
included_domain_ips = []	(List of IPAddress) Comma separated Fault Domain IPs to be included from iSCSI returns.
san_api_port = None	(Port(min=0, max=65535)) Port to use to access the SAN API
san_clustername = <>	(String) Cluster name to use for creating volumes
san_ip = <>	(String) IP address of SAN controller
san_is_local = False	(Boolean) Execute commands locally instead of over SSH; use if the volume service is running on the SAN device
san_login = admin	(String) Username for SAN controller
san_password = <>	(String) Password for SAN controller
san_private_key = <>	(String) Filename of private key to use for SSH authentication
san_ssh_port = 22	(Port(min=0, max=65535)) SSH port to use with SAN
san_thin_provision = True	(Boolean) Use thin provisioning for SAN volumes?
secondary_san_ip = <>	(String) IP address of secondary DSM controller
secondary_san_login = Admin	(String) Secondary DSM user name
secondary_san_passworc = <>	(String) Secondary DSM user password name
secondary_sc_api_port = 3033	(Port(min=0, max=65535)) Secondary Dell API port
ssh_conn_timeout = 30	(Integer) SSH connection timeout in seconds
ssh_max_pool_conn = 5	(Integer) Maximum ssh connections in the pool
ssh_min_pool_conn = 1	(Integer) Minimum ssh connections in the pool
excluded_domain_ip = None	(IPAddress) DEPRECATED: Fault Domain IP to be excluded from iSCSI returns. **DEPRECATED**

Fujitsu ETERNUS DX driver

Fujitsu ETERNUS DX driver provides FC and iSCSI support for ETERNUS DX series.

The driver performs volume operations by communicating with ETERNUS DX. It uses a CIM client in Python called PyWBEM to perform CIM operations over HTTP.

You can specify RAID Group and Thin Provisioning Pool (TPP) in ETERNUS DX as a storage pool.

System requirements

Supported storages:

- ETERNUS AF150 S3
- ETERNUS AF250 S3/AF250 S2/AF250
- ETERNUS AF650 S3/AF650 S2/AF650
- ETERNUS DX200F
- ETERNUS DX60 S5/S4/S3
- ETERNUS DX100 S5/S4/S3
- ETERNUS DX200 S5/S4/S3
- ETERNUS DX500 S5/S4/S3
- ETERNUS DX600 S5/S4/S3
- ETERNUS DX8700 S3/DX8900 S4/S3

Requirements:

- Firmware version V10L30 or later is required.
- The multipath environment with ETERNUS Multipath Driver is unsupported.
- An Advanced Copy Feature license is required to create snapshots, create volume from snapshots, or clone volumes.

Supported operations

- Create, delete, attach, and detach volumes.
- Create, list, and delete volume snapshots.
- Create a volume from a snapshot.
- Copy an image to a volume.
- Copy a volume to an image.
- Clone a volume.
- Extend a volume.
- Get volume statistics.
- Migrate Volume.
- Revert a volume to snapshot.

Preparation

Package installation

Install the `python-pywbem` package for your distribution.

ETERNUS DX setup

Perform the following steps using ETERNUS Web GUI or ETERNUS CLI.

Note

- These following operations require an account that has the `Admin` role.
- For detailed operations, refer to ETERNUS Web GUI Users Guide or ETERNUS CLI Users Guide for ETERNUS DX series.

1. Create an account with software role for communication with cinder controller.
2. Enable the SMI-S of ETERNUS DX.
3. Register an Advanced Copy Feature license and configure copy table size.
4. Create a storage pool for volumes.
5. (Optional) If you want to create snapshots on a different storage pool for volumes, create a storage pool for snapshots.
6. Create Snap Data Pool Volume (SDPV) to enable Snap Data Pool (SDP) for `create a snapshot`.
7. Configure storage ports to be used by the Block Storage service.
   - Set those storage ports to CA mode.
   - Enable the host-affinity settings of those storage ports.

     (ETERNUS CLI command for enabling host-affinity settings):

     ```
     CLI> set fc-parameters -host-affinity enable -port <CM#><CA#><Port>
     CLI> set iscsi-parameters -host-affinity enable -port <CM#><CA#>
     ↪<Port>
     ```

   Note

   - Replace <CM#> and <CA#> with the name of the controller enclosure where the port is located.
   - Replace <Port> with the port number.

8. Ensure LAN connection between cinder controller and MNT port of ETERNUS DX and SAN connection between Compute nodes and CA ports of ETERNUS DX.
9. (Optional) If you want to use a public key to SSH to the ETERNUS DX storage, generate the SSH key, and upload the `eternus.ietf` file to the ETERNUS storage.

   For information about how to set the public key, refer to the ETERNUS Web GUI manuals.

```
$ ssh-keygen -t rsa -N "" -f ./eternus -m PEM
$ ssh-keygen -e -f ./eternus.pub > ./eternus.ietf
```

If the public key(eternus.ietf) that was created is deleted by mistake, use the following command to recreate the key.

```
$ ssh-keygen -e -f /root/.ssh/eternus.pub > ./eternus.ietf
```

Configuration

1. Add the following entries to `/etc/cinder/cinder.conf`:

   FC entries:

   ```
   volume_driver = cinder.volume.drivers.fujitsu.eternus_dx.eternus_dx_fc.
   ↪FJDXFCDriver
   cinder_eternus_config_file = /etc/cinder/eternus_dx.xml
   ```

   iSCSI entries:

   ```
   volume_driver = cinder.volume.drivers.fujitsu.eternus_dx.eternus_dx_iscsi.
   ↪FJDXISCSIDriver
   cinder_eternus_config_file = /etc/cinder/eternus_dx.xml
   ```

   If there is no description about `cinder_eternus_config_file`, then the parameter is set to default value `/etc/cinder/cinder_fujitsu_eternus_dx.xml`.

2. Create a driver configuration file.

   Create a driver configuration file in the file path specified as `cinder_eternus_config_file` in `cinder.conf`, and add parameters to the file as below:

   FC configuration:

   ```
   <?xml version='1.0' encoding='UTF-8'?>
   <FUJITSU>
   <EternusIP>0.0.0.0</EternusIP>
   <EternusPort>5988</EternusPort>
   <EternusUser>smisuser</EternusUser>
   <EternusPassword>smispassword</EternusPassword>
   <EternusPool>raid5_0001</EternusPool>
   <EternusPool>tpp_0001</EternusPool>
   <EternusPool>raid_0002</EternusPool>
   <EternusSnapPool>raid5_0001</EternusSnapPool>
   </FUJITSU>
   ```

   iSCSI configuration:

   ```
   <?xml version='1.0' encoding='UTF-8'?>
   <FUJITSU>
   <EternusIP>0.0.0.0</EternusIP>
   <EternusPort>5988</EternusPort>
   <EternusUser>smisuser</EternusUser>
   ```

   (continues on next page)

(continued from previous page)

```
<EternusPassword>smispassword</EternusPassword>
<EternusPool>raid5_0001</EternusPool>
<EternusPool>tpp_0001</EternusPool>
<EternusPool>raid_0002</EternusPool>
<EternusSnapPool>raid5_0001</EternusSnapPool>
<EternusISCSIIP>1.1.1.1</EternusISCSIIP>
<EternusISCSIIP>1.1.1.2</EternusISCSIIP>
<EternusISCSIIP>1.1.1.3</EternusISCSIIP>
<EternusISCSIIP>1.1.1.4</EternusISCSIIP>
</FUJITSU>
```

Where:

EternusIP
IP address of the SMI-S connection of the ETRENUS device.

Use the IP address of the MNT port of device.

EternusPort
Port number for the SMI-S connection port of the ETERNUS device.

EternusUser
User name of `sofware` role for the connection `EternusIP`.

EternusPassword
Corresponding password of `EternusUser` on `EternusIP`.

EternusPool (Multiple setting allowed)
Name of the storage pool for the volumes from `ETERNUS DX setup`.

Use the pool RAID Group pool name or TPP pool name in the ETERNUS device.

EternusSnapPool (Multiple setting allowed)
Name of the storage pool for the snapshots from `ETERNUS DX setup`.

Use the pool RAID Group pool name or TPP pool name in the ETERNUS device.

If you did not create a different pool for snapshots, use the same value as `EternusPool`.

EternusISCSIIP (Multiple setting allowed)
iSCSI connection IP address of the ETERNUS DX.

Note

- You can specify the same RAID Group pool name or TPP pool name for `EternusPool` and `EternusSnapPool` if you create volumes and snapshots on a same storage pool.
- For `EternusPool`, when multiple pools are specified, cinder-scheduler will select one from multiple pools to create the volume.

Configuration example

1. Edit `cinder.conf`:

```
[DEFAULT]
enabled_backends = DXFC, DXISCSI

[DXFC]
volume_driver = cinder.volume.drivers.fujitsu.eternus_dx.eternus_dx_fc.
↪FJDXFCDriver
cinder_eternus_config_file = /etc/cinder/fc.xml
volume_backend_name = FC
fujitsu_passwordless = False

[DXISCSI]
volume_driver = cinder.volume.drivers.fujitsu.eternus_dx.eternus_dx_iscsi.
↪FJDXISCSIDriver
cinder_eternus_config_file = /etc/cinder/iscsi.xml
volume_backend_name = ISCSI
fujitsu_passwordless = True
fujitsu_private_key_path = /etc/cinder/eternus
```

2. Create the driver configuration files `fc.xml` and `iscsi.xml`.
3. Create a volume type and set extra specs to the type:

```
$ cinder type-create DX_FC
$ cinder type-key DX_FX set volume_backend_name=FC
$ cinder type-create DX_ISCSI
$ cinder type-key DX_ISCSI set volume_backend_name=ISCSI
```

By issuing these commands, the volume type `DX_FC` is associated with the `FC`, and the type `DX_ISCSI` is associated with the `ISCSI`.

Supported Functions of the ETERNUS OpenStack VolumeDriver

Migrate Volume

Moves volumes to a different storage pool.

1. ETERNUS AF/DX functions
   - Creates migration destination volumes / deletes migration source volumes.
   - Sets access paths to migration volumes / deletes migration access paths to migration source volumes.
   - Uses Create Volume, Delete Volume, Attach Volume and Detach Volume.
2. Cinder operation
   - Copies data in the migration source volume to the migration destination volume.

Note

Host information must be specified in Migrated Volume.

The input format is as follows:

```
Host-Name@Backend-Name#Pool-Name
```

For the following environment or settings, specify `test.localhost@Backend1#PoolA` for the host.

- PoolA is a pool specified in `/etc/cinder/cinder_fujitsu_eternus_dx.xml`.

```
$ hostname
  test.localhost

$ cat /etc/cinder/cinder.conf
  (snip)
  [Backend1]
  volume_driver=cinder.volume.drivers.fujitsu.eternus_dx.eternus_
↪dx_fc.FJDXFCDriver
  cinder_eternus_config_file = /etc/cinder/cinder_fujitsu_eternus_
↪dx.xml
  volume_backend_name=volume_backend_name1
```

Warning

There are some restrictions for volume migration:

1. You cannot migrate a volume that has snapshots.
2. You cannot use driver-assisted migration to move a volume to or from a backend that does not use the ETERNUS OpenStack volume driver.

Supplementary Information for the Supported Functions

QoS Settings

The QoS settings that are linked with the volume QoS function of the ETERNUS AF/DX are available.

An upper limit value of the bandwidth(BWS) can be set for each volume. A lower limit value can not be set.

The upper limit is set if the firmware version of the ETERNUS AF/DX is earlier than V11L30, and the IOPS/Throughput of Total/Read/Write for the volume is set separately for V11L30 and later.

The following procedure shows how to set the QoS.

1. Create a QoS definition.

   - The firmware version of the ETERNUS AF/DX is earlier than V11L30

   ```
   $ cinder qos-create <qos_name> maxBWS=xx
   ```

   For <qos_name>, specify the name of the definition that is to be created.

   For maxBWS, specify a value in MB.

   - The firmware version of the ETERNUS AF/DX is V11L30 or later

```
$ cinder qos-create <qos_name> read_iops_sec=15000 write_iops_sec=12600␣
↪total_iops_sec=15000 read_bytes_sec=800 write_bytes_sec=700 total_bytes_
↪sec=800
```

2. When not using the existing volume type, create a new volume type.

   ```
   $ cinder type-create <volume_type_name>
   ```

   For <volume_type_name>, specify the name of the volume type that is to be created.

3. Associate the QoS definition with the volume type.

   ```
   $ cinder qos-associate <qos_specs> <volume_type_id>
   ```

   For <qos_specs>, specify the ID of the QoS definition that was created.

   For <volume_type_id>, specify the ID of the volume type that was created.

Cautions

1. For the procedure to cancel the QoS settings, refer to OpenStack Command-Line Interface Reference.
2. The QoS mode of the ETERNUS AF/DX must be enabled in advance. For details, refer to the ETERNUS Web GUI manuals.
3. When the firmware version of the ETERNUS AF/DX is earlier than V11L30, for the volume QoS settings of the ETERNUS AF/DX, upper limits are set using the predefined options.

   Therefore, set the upper limit of the ETERNUS AF/DX side to a maximum value that does not exceed the specified maxBWS.

   The following table shows the upper limits that can be set on the ETERNUS AF/DX side and example settings. For details about the volume QoS settings of the ETERNUS AF/DX, refer to the ETERNUS Web GUI manuals.

Settings for the ETERNUS AF/DX
Unlimited
15000 IOPS (800MB/s)
12600 IOPS (700MB/s)
10020 IOPS (600MB/s)
7500 IOPS (500MB/s)
5040 IOPS (400MB/s)
3000 IOPS (300MB/s)
1020 IOPS (200MB/s)
780 IOPS (100MB/s)
600 IOPS (70MB/s)
420 IOPS (40MB/s)
300 IOPS (25MB/s)
240 IOPS (20MB/s)
180 IOPS (15MB/s)
120 IOPS (10MB/s)
60 IOPS (5MB/s)

- When specified maxBWS=750

  12600 IOPS (700MB/s) is set on the ETERNUS AF/DX side.

- When specified maxBWS=900

  15000 IOPS (800MB/s) is set on the ETERNUS AF/DX side.

4. While a QoS definition is being created, if an option other than maxBWS/read_iops_sec/write_iops_sec/total_iops_sec/read_bytes_sec /write_bytes_sec/total_bytes_sec is specified, a warning log is output and the QoS information setting is continued.

5. For an ETERNUS AF/DX wth a firmware version of before V11L30, if a QoS definition volume type that is set with read_iops_sec/ write_iops_sec/total_iops_sec/read_bytes_sec/write_bytes_sec/total_bytes_sec is specified for Create Volume, a warning log is output and the process is terminated.

6. For an ETERNUS AF/DX with a firmware version of V11L30 or later, if a QoS definition volume type that is set with maxBWS is specified for Create Volume, a warning log is output and the process is terminated.

7. After the firmware of the ETERNUS AF/DX is upgraded from V11L10/V11L2x to a newer version, the volume types related to the QoS definition created before the firmware upgrade can no longer be used. Set a QoS definition and create a new volume type.

8. When the firmware of the ETERNUS AF/DX is downgraded to V11L10/V11L2x, do not use a volume type linked to a pre-firmware downgrade QoS definition, because the QoS definition may work differently from ones post-firmware downgrade. For the volume, create and link a volume type not associated with any QoS definition and after the downgrade, create and link a volume type associated with a QoS definition.

9. If Create Volume terminates with an error, Cinder may not invoke Delete Volume.

   If volumes are created but the QoS settings fail, the ETERNUS OpenStack VolumeDriver ends the process to prevent the created volumes from being left in the ETERNUS AF/DX. If volumes fail to be created, the process terminates with an error.

Specification of the Snapshot Creation Destination Pool

A RAID Group or a Thin Provisioning Pool (TPP) can be specified as the snapshot creation destination pool. In an ETERNUS AF/DX with a firmware version earlier than or equal to V10L60, Thin Provisioning Pools(TPPs) cannot be used as the snapshot creation destination pool.

Multiple snapshot creation destination pools can be specified.

A pool where snapshots can be created is searched in the order written in the driver configuration file and if one is found, snapshots are created in that pool.

Cautions

1. If the creation destination pool is a RAID Group, more than 128 snapshots cannot be created. Therefore, to create more than 128 snapshots in a RAID Group, multiple RAID Groups must be specified as snapshot creation destination pools.

2. When creating a snapshot, Cinder Scheduler checks the capacity of the pool where the source volume is located. This may lead to the failure of snapshot creation fail to be created if this pool has insufficient capacity, even if the snapshot pool specified by `EternusSnapPool` has sufficient capacity.

3. If multiple snapshot creation destination pools are specified, a different pool must be specified for the volume creation destination pool (`EternusPool` and `EternusSnapPool` can be specified multiple times but the same pool name cannot be specified). If the same pool name is specified and instructions to create multiple volumes and multiple snapshots are issued at the same time, the number of logical volumes in a RAID Group will reach 128 and the operation may fail.

4. To address the issue that a volume with snapshot cannot be extended, a parameter `fujitsu_use_cli_copy` has been introduced.

   The default value of `fujitsu_use_cli_copy` is `False`.

   If `fujitsu_use_cli_copy` is set to `True`, create a Snapshot using the CLI method instead of SMI-S method, allowing volume extension of the source volume.

   ```
   $ cat /etc/cinder/cinder.conf
     (snip)
     [Backend1]
     volume_driver=cinder.volume.drivers.fujitsu.eternus_dx.eternus_
   ↪dx_fc.FJDXFCDriver
     cinder_eternus_config_file = /etc/cinder/cinder_fujitsu_
   ↪eternus_dx.xml
     volume_backend_name = volume_backend_name1
     fujitsu_use_cli_copy = True
   ```

   Note that `fujitsu_use_cli_copy` cannot be set to True when the type of target pool is RAID Group.

Fungible Storage Driver

Fungible Storage volume driver provides OpenStack Compute instances with access to Fungible Storage Cluster.

This documentation explains how to configure Cinder for use with the Fungible Storage Cluster.

Driver requirements

- Fungible Storage Cluster
- FSC version >= 4.0
- nvme cli version >= v1.13
- The Block Storage Node should also have a data path to the Fungible Storage Cluster for the following operations:
  - Copy volume to image
  - Copy image to volume

Driver options

The following table contains the configuration options supported by the Fungible Storage driver.

Table 24: Description of Fungible Storage Cluster configuration options

Configuration option = Default value	Description
`api_enable_ssl = True`	(Boolean) Specify whether to use SSL or not when accessing the composer APIs
`fsc_clone_volume_time = 1800`	(Integer) Create clone volume timeout in seconds
`iops_for_image_migrat = 250000`	(Integer) Maximum read IOPS that volume can get when reading data from the volume during host assisted migration
`nvme_connect_port = 4420`	(Port(min=0, max=65535)) The port number to be used when doing nvme connect from host

Supported operations

- Create, list, delete, attach and detach volumes
- Create, list and delete volume snapshots
- Copy image to volume
- Copy volume to image
- Create volume from snapshot
- Clone volume
- Extend volume

Configure Fungible Storage Cluster backend

This section details the steps required to configure the Fungible Storage cinder driver.

1. In the `cinder.conf` configuration file under the `[DEFAULT]` section, set the enabled_backends parameter.

   ```
   [DEFAULT]
   enabled_backends = fungible
   ```

2. Add a backend group section for the backend group specified in the enabled_backends parameter.

3. In the newly created backend group section, set the following configuration options:

   ```
   [fungible]
   # Backend name
   volume_backend_name=fungible
   # The driver path
   volume_driver=cinder.volume.drivers.fungible.driver.FungibleDriver
   # Fungible composer details
   san_ip = <composer node VIP>
   san_login = <composer username>
   san_password = <composer password>
   # List below are optional
   ```

(continues on next page)

(continued from previous page)

```
nvme_connect_port = <nvme target endpoint port>
api_enable_ssl = True/False
iops_for_image_migration = <IOPS value>
```

Hedvig Volume Driver

Hedvig provides software-defined storage for enterprises building private, hybrid, or multi-cloud environments. Hedvigs patented Universal Data Plane technology forms a distributed, scale-out cluster that transforms commodity servers or cloud computing into a unified data fabric.

The Hedvig Cinder Driver interacts with a configured backend Hedvig Cluster using REST APIs.

Using the Hedvig Volume Driver

With the Hedvig Volume Driver for OpenStack, you can :

- **Integrate public and private clouds:**
  Build a unified hybrid environment to easily migrate to or from your data center and public clouds.
- **Set granular virtual disk policies:**
  Assign enterprise-class features on a per volume basis to best fit your application requirements.
- **Connect to any compute environment:**
  Use with any hypervisor, application, or bare-metal system.
- **Grow seamlessly with an elastic cluster:**
  Scale storage performance and capacity on-the-fly with off-the-shelf x86 servers.
- **Deliver predictable performance:**
  Receive consistent high-IOPS performance for demanding applications through massive parallelism, dedicated flash, and edge cache configurations.

Requirement

Hedvig Volume Driver, version 1.0.0 and later, supports Hedvig release 3.0 and later.

Supported operations

Hedvig supports the core features of OpenStack Cinder:

- Create and delete volumes
- Attach and detach volumes
- Create and delete snapshots
- Create volume from snapshot
- Get volume stats
- Copy image to volume
- Copy volume to image
- Clone volume

- Extend volume
- Enable deduplication, encryption, cache, compression, custom replication policy on a volume level using volume-type extra-specs

Hedvig Volume Driver configuration

The Hedvig Volume Driver can be configured by editing the cinder.conf file located in the /etc/cinder/ directory.

```
[DEFAULT]
enabled_backends=hedvig

[HEDVIG_BACKEND_NAME]
volume_driver=cinder.volume.drivers.hedvig.hedvig_cinder.HedvigISCSIDriver
san_ip=<Comma-separated list of HEDVIG_IP/HOSTNAME of the cluster nodes>
san_login=HEDVIG_USER
san_password=HEDVIG_PASSWORD
san_clustername=HEDVIG_CLUSTER
```

Run the following commands on the OpenStack Cinder Node to create a Volume Type for Hedvig:

```
cinder type-create HEDVIG_VOLUME_TYPE
cinder type-key HEDVIG_VOLUME_TYPE  set volume_backend_name=HEDVIG_BACKEND_
↪NAME
```

This section contains definitions of the terms used above.

HEDVIG_IP/HOSTNAME
The IP address or hostnames of the Hedvig Storage Cluster Nodes

HEDVIG_USER
Username to login to the Hedvig Cluster with minimum `super user` (admin) privilege

HEDVIG_PASSWORD
Password to login to the Hedvig Cluster

HEDVIG_CLUSTER
Name of the Hedvig Cluster

Note

Restart the `cinder-volume` service after updating the `cinder.conf` file to apply the changes and to initialize the Hedvig Volume Driver.

Hedvig QoS Spec parameters and values

- dedup_enable true/false
- compressed_enable true/false
- cache_enable true/false
- replication_factor 1-6
- replication_policy Agnostic/RackAware/DataCenterAware

- replication_policy_info comma-separated list of data center names (applies only to a replication_policy of DataCenterAware)
- disk_residence Flash/HDD
- encryption true/false

Creating a Hedvig Cinder Volume with custom attributes (QoS Specs)

1. Create a QoS Spec with the list of attributes that you want to associate with a volume. For example, to create a Cinder Volume with deduplication enabled, create a QoS Spec called dedup_enable with dedup_enable=true
2. Create a new volume type and associate this QoS Spec with it, OR associate the QoS Spec with an existing volume type.
3. Every Cinder Volume that you create of the above volume type will have deduplication enabled.
4. If you do create a new volume type, make sure to add the key volume_backend_name so OpenStack knows that the Hedvig Volume Driver handles all requests for this volume.

Hitachi block storage driver

Hitachi block storage driver provides Fibre Channel and iSCSI support for Hitachi VSP storages.

System requirements

Supported storages:

Storage model	Firmware version
VSP E590, E790	93-03-22 or later
VSP E990	93-01-01 or later
VSP E1090, E1090H	93-06-2x or later
VSP F350, F370, F700, F900 VSP G350, G370, G700, G900	88-01-04 or later
VSP F400, F600, F800 VSP G200, G400, G600, G800	83-04-43 or later
VSP N400, N600, N800	83-06-01 or later
VSP 5100, 5500, 5100H, 5500H	90-01-41 or later
VSP 5200, 5600, 5200H, 5600H	90-08-0x or later
VSP F1500 VSP G1000, VSP G1500	80-05-43 or later

Required storage licenses:

- Hitachi Storage Virtualization Operating System (SVOS)
  - Hitachi LUN Manager
  - Hitachi Dynamic Provisioning
- Hitachi Local Replication (Hitachi Thin Image)

Optional storage licenses:

- Deduplication and compression

- Global-Active Device

Supported operations

- Create, delete, attach, and detach volumes.
- Create, list, and delete volume snapshots.
- Create a volume from a snapshot.
- Create, list, update, and delete consistency groups.
- Create, list, and delete consistency group snapshots.
- Copy a volume to an image.
- Copy an image to a volume.
- Clone a volume.
- Extend a volume.
- Migrate a volume (host assisted).
- Migrate a volume (storage assisted).
- Get volume statistics.
- Efficient non-disruptive volume backup.
- Manage and unmanage a volume.
- Attach a volume to multiple instances at once (multi-attach).
- Revert a volume to a snapshot.

Hitachi block storage driver also supports the following additional features:

- Global-Active Device
- Maximum number of copy pairs and consistency groups
- Data deduplication and compression
- Port scheduler
- Port assignment using extra spec
- Configuring Quality of Service (QoS) settings

Note

- A volume having snapshots cannot be extended with this driver.
- Storage assisted volume migration is only supported between same storage.

Configuration

Set up Hitachi storage

You need to specify settings as described below for storage systems. For details about each setting, see the users guide of the storage systems.

Common resources:

1. **All resources**
   The name of any storage resource, such as a DP pool or a host group, cannot contain any whitespace characters or else it will be unusable by the driver.
2. **User accounts**
   Create a storage device account belonging to the Administrator User Group.
3. **DP Pool**
   Create a DP pool that is used by the driver.
4. **Resource group**
   If using a new resource group for exclusive use by an OpenStack system, create a new resource group, and assign the necessary resources, such as LDEVs, port, and host group (iSCSI target) to the created resource.
5. **Ports**
   Enable Port Security for the ports used by the driver.

If you use iSCSI:

1. **Ports**
   Assign an IP address and a TCP port number to the port.

Note

- Do not change LDEV nickname for the LDEVs created by Hitachi block storage driver. The nickname is referred when deleting a volume or a snapshot, to avoid data-loss risk. See details in bug #2072317.

Set up Hitachi storage volume driver and volume operations

Set the volume driver to Hitachi block storage driver by setting the volume_driver option in the cinder.conf file as follows:

If you use Fibre Channel:

```
[hitachi_vsp]
volume_driver = cinder.volume.drivers.hitachi.hbsd_fc.HBSDFCDriver
volume_backend_name = hitachi_vsp
san_ip = 1.2.3.4
san_login = hitachiuser
san_password = password
hitachi_storage_id = 123456789012
hitachi_pools = pool0
```

If you use iSCSI:

```
[hitachi_vsp]
volume_driver = cinder.volume.drivers.hitachi.hbsd_iscsi.HBSDISCSIDriver
volume_backend_name = hitachi_vsp
san_ip = 1.2.3.4
san_login = hitachiuser
```

(continues on next page)

(continued from previous page)

```
san_password = password
hitachi_storage_id = 123456789012
hitachi_pools = pool0, pool1
```

Configuration options

This table shows configuration options for Hitachi block storage driver.

Table 25: Descrip
options

Configuration option = Default value	Description
hitachi_async_copy_check_interval = 10	(Integer(min=1, max=600)) Interval in seconds t
hitachi_compute_target_ports = []	(List of String) IDs of the storage ports used to a
hitachi_copy_check_interval = 3	(Integer(min=1, max=600)) Interval in seconds t
hitachi_copy_speed = 3	(Integer(min=1, max=15)) Copy speed of storage
hitachi_discard_zero_page = True	(Boolean) Enable or disable zero page reclamatio
hitachi_exec_retry_interval = 5	(Integer) Retry interval in seconds for REST API
hitachi_extend_timeout = 600	(Integer) Maximum wait time in seconds for a vo
hitachi_group_create = False	(Boolean) If True, the driver will create host grou
hitachi_group_delete = False	(Boolean) If True, the driver will delete host grou
hitachi_group_name_format = None	(String) Format of host groups, iSCSI targets, an
hitachi_host_mode_options = []	(List of Integer) Host mode option for host group
hitachi_ldev_range = None	(String) Range of the LDEV numbers in the form
hitachi_lock_timeout = 7200	(Integer) Maximum wait time in seconds for stora
hitachi_lun_retry_interval = 1	(Integer) Retry interval in seconds for REST API
hitachi_lun_timeout = 50	(Integer) Maximum wait time in seconds for addi
hitachi_mirror_auth_password = None	(String) iSCSI authentication password
hitachi_mirror_auth_user = None	(String) iSCSI authentication username
hitachi_mirror_compute_target_ports = []	(List of String) Target port names of compute no
hitachi_mirror_ldev_range = None	(String) Logical device range of secondary storag
hitachi_mirror_pair_target_number = 0	(Integer(min=0, max=99)) Pair target name of th
hitachi_mirror_pool = None	(String) Pool of secondary storage system
hitachi_mirror_rest_api_ip = None	(String) IP address of REST API server
hitachi_mirror_rest_api_port = 443	(Port(min=0, max=65535)) Port number of REST
hitachi_mirror_rest_pair_target_ports = []	(List of String) Target port names for pair of the
hitachi_mirror_rest_password = None	(String) Password of secondary storage system fo
hitachi_mirror_rest_user = None	(String) Username of secondary storage system f
hitachi_mirror_snap_pool = None	(String) Thin pool of secondary storage system
hitachi_mirror_ssl_cert_path = None	(String) Can be used to specify a non default path
hitachi_mirror_ssl_cert_verify = False	(Boolean) If set to True the http client will valida
hitachi_mirror_storage_id = None	(String) ID of secondary storage system
hitachi_mirror_target_ports = []	(List of String) Target port names for host group
hitachi_mirror_use_chap_auth = False	(Boolean) Whether or not to use iSCSI authentica
hitachi_pair_target_number = 0	(Integer(min=0, max=99)) Pair target name of the
hitachi_path_group_id = 0	(Integer(min=0, max=255)) Path group ID assign
hitachi_pools = []	(List of String) Pool number[s] or pool name[s] o
hitachi_port_scheduler = False	(Boolean) Enable port scheduling of WWNs to th
hitachi_quorum_disk_id = None	(Integer(min=0, max=31)) ID of the Quorum disk

Table

Configuration option = Default value	Description
hitachi_replication_copy_speed = 3	(Integer(min=1, max=15)) Remote copy speed of
hitachi_replication_number = 0	(Integer(min=0, max=255)) Instance number for
hitachi_replication_status_check_long_interval = 600	(Integer) Interval at which remote replication pai
hitachi_replication_status_check_short_interval = 5	(Integer) Initial interval at which remote replicati
hitachi_replication_status_check_timeout = 86400	(Integer) Maximum wait time before the remote r
hitachi_rest_another_ldev_mapped_retry_timeout = 600	(Integer) Retry time in seconds when new LUN a
hitachi_rest_connect_timeout = 30	(Integer) Maximum wait time in seconds for conn
hitachi_rest_disable_io_wait = True	(Boolean) This option will allow detaching volun
hitachi_rest_get_api_response_timeout = 1800	(Integer) Maximum wait time in seconds for a res
hitachi_rest_job_api_response_timeout = 1800	(Integer) Maximum wait time in seconds for a res
hitachi_rest_keep_session_loop_interval = 180	(Integer) Loop interval in seconds for keeping RE
hitachi_rest_pair_target_ports = []	(List of String) Target port names for pair of the l
hitachi_rest_server_busy_timeout = 7200	(Integer) Maximum wait time in seconds when R
hitachi_rest_tcp_keepalive = True	(Boolean) Enables or disables use of REST API t
hitachi_rest_tcp_keepcnt = 4	(Integer) Maximum number of transmissions for
hitachi_rest_tcp_keepidle = 60	(Integer) Wait time in seconds for sending a first
hitachi_rest_tcp_keepintvl = 15	(Integer) Interval of transmissions in seconds for
hitachi_rest_timeout = 30	(Integer) Maximum wait time in seconds for each
hitachi_restore_timeout = 86400	(Integer) Maximum wait time in seconds for the r
hitachi_set_mirror_reserve_attribute = True	(Boolean) Whether or not to set the mirror reserv
hitachi_snap_pool = None	(String) Pool number or pool name of the snapsh
hitachi_state_transition_timeout = 900	(Integer) Maximum wait time in seconds for a vo
hitachi_storage_id = None	(String) Product number of the storage system.
hitachi_target_ports = []	(List of String) IDs of the storage ports used to at
hitachi_zoning_request = False	(Boolean) If True, the driver will configure FC zo

Required options

- **san_ip**
  IP address of SAN controller
- **san_login**
  Username for SAN controller
- **san_password**
  Password for SAN controller
- **hitachi_storage_id**
  Product number of the storage system.
- **hitachi_pools**
  Pool number(s) or pool name(s) of the DP pool.

Set up and operation for additional features

Set up Global-Active Device and volume operation

Beginning with the 2023.1, If you use Global-Active Device (GAD), you can make the data of individual volumes redundant between two storage systems, thereby improving the availability of the storage systems. For details, see the Global-Active Device User Guide.

Note

- You cannot apply Global-Active Device configuration and remote replication configuration to the same backend.
- You cannot use Asymmetric Logical Unit Access (ALUA).

Storage firmware versions for GAD

If you are using a VSP F350, F370, F700, F900 storage system or a VSP G350, G370, G700,G900 storage system in a Global-Active Device configuration, make sure the firmware version is 88-03-21 or later.

Creating a Global-Active Device environment

Before using Global-Active Device, create the prerequisite environment, such as connecting remote paths, configuring a quorum disk, and creating a virtual storage machine (VSM), by other storage system management tools. Hitachi block storage driver supports the following configurations.

- Configuration where the P-VOL is not registered to a VSM
- Configuration where the P-VOL is registered to a VSM

For details, see the Workflow for creating a GAD environment in the Global-Active Device User Guide

Hitachi block storage driver automatically setups following procedures that are described in the section Workflow for creating a GAD environment :

- The following steps of Setting up the secondary storage system:
  - Setting the GAD reserve attribute on the S-VOL
  - Creating a host group (Only if the configuration option `hitachi_group_create` is True)
  - Creating the S-VOL
  - Adding an LU path to the S-VOL
- Updating the CCI configuration definition files
- Creating the GAD pair
- Adding an alternate path to the S-VOL

You must register the information about the secondary storage system to the REST API server in the primary site and register the information about the primary storage system to the REST API server in the secondary site. For details about how to register the information, see the Hitachi Command Suite Configuration Manager REST API Reference Guide or the Hitachi Ops Center API Configuration Manager REST API Reference Guide.

Note

- The users specified for both configuration options `san_login` and `hitachi_mirror_rest_user` must have following roles:
  - Storage Administrator (View & Modify)
  - Storage Administrator (Remote Copy)

- Reserve unused host group IDs (iSCSI target IDs) for the resource groups related on the VSM. Reserve the IDs in ascending order. The number of IDs you need to reserve is 1 plus the sum of the number of controller nodes and the number of compute nodes. For details on how to reserve a host group ID (iSCSI target ID), see Global-Active Device User Guide.
- The LUNs of the host groups (iSCSI targets) of the specified ports on the primary storage system must match the LUNs of the host groups (iSCSI targets) of the specified ports on the secondary storage system. If they do not match, match the LUNs for the primary storage system with those for the secondary storage system.
- When you use a same storage system as secondary storage system for Global-Active Device configuration and backend storage system for general use at the same time, you cannot use the same ports between different backend storage systems. Please specify different ports in the configuration options `hitachi_target_ports`, `hitachi_compute_target_ports`, or `hitachi_rest_pair_target_ports` between different backend storage systems.

Create volume in a Global-Active Device configuration

If you create a Cinder volume in a Global-Active Device configuration, each Global-Active Device pair is mapped to a Cinder volume.

In order for you to create volumes with the Global-Active Device attribute specified, you must first create a volume type that contains the `hbsd:topology=active_active_mirror_volume` extra-spec. You can do this as follows:

```
$ openstack volume type create <volume type name>
$ openstack volume type set --property \
hbsd:topology=active_active_mirror_volume <volume type name>
```

You can then create GAD volumes as follows:

```
$ openstack volume create --type <volume type name> --size <size>
```

Note

- In this case, the following restrictions apply:
  - You cannot create a volume for which the deduplication and compression function is enabled, or creating a volume will be failed with the error `MSGID0753-E: Failed to create a volume in a GAD environment because deduplication is enabled for the volume type.`.
- Note the following if the configuration is P-VOL registered to a VSM:
  - Do not create volumes whose volume types do not have `hbsd:topology=active_active_mirror_volume` extra-spec.
  - While setting up the environment, set a virtual LDEV ID for every LDEV specified by the configuration option `hitachi_ldev_range parameter` on the primary storage system using storage management software because virtual LDEV IDs are necessary for GAD pair creation.

Unavailable Cinder functions

Following cinder functions are unavailable in a Global-Active Device configuration:

- Migrate a volume (storage assisted)
- Manage Volume
- Unmanage Volume

Note

In addition, if the configuration is P-VOL registered to a VSM, the backup creation command of the Backup Volume functions cannot be run with the `--snapshot option` or the `--force` option specified.

Maximum number of copy pairs and consistency groups

The maximum number of Thin Image pairs that can be created for each LDEV assigned to a volume (or snapshot) is restricted on a per-storage-system basis. If the number of pairs exceeds the maximum, copying cannot proceed normally.

For information about the maximum number of copy pairs and consistency groups that can be created, see the Hitachi Thin Image User Guide.

Configuring Quality of Service (QoS) settings

By configuring Quality of Service (QoS) settings, you can restrict the I/O processing of each volume, thereby maintaining the required performance and quality levels.

In Hitachi block storage driver, you can configure the following settings for each volume. However, you cannot configure these settings for journal volumes.

- Throughput (IOPS, amount of data transferred in MB/s)

  You can set the upper and lower limits on throughput. If an upper limit is exceeded, I/O is suppressed. If a lower limit is not met, I/O is adjusted so that the lower limit is met.

- Priority level of the I/O processing

  You can set priority levels for the I/O processing of multiple volumes. I/O is adjusted for faster I/O response, starting with high-priority volumes.

System requirements for a QoS

Storage firmware versions

Storage model	Firmware version
VSP F350, F370, F700, F900 VSP G350, G370, G700, G900	88-06-01 or later
VSP 5100, 5500, 5100H, 5500H	90-04-01 or later

Storage management software

Configuration Manager REST API version 10.2.0-00 or later is required.

Configuring QoS settings and creating volumes

Create QoS specs that define QoS settings, and then associate the QoS specs with a volume type. You can configure QoS settings for a volume by running the following functions with this volume type specified.

- Create Volume
- Create Snapshot
- Create Volume from Snapshot
- Create Volume from Volume (Clone)
- Consistency Group
- Generic volume group

The following example describes the procedure for configuring QoS settings when creating a new volume using the Create Volume function.

Before you begin, Check the following information.

- QoS settings
  - Upper or lower limit on throughput (IOPS, amount of data transferred in MB/s)
  - Priority level of I/O processing
- ID and name of the volume type

  A volume type is needed in order to associate it with the QoS specs. If no volume types exist, create one in advance.

Procedure

1. Create the QoS specs

   a. If you use the cinder command:

```
$ cinder qos-create <name-of-the-QoS-specs> [consumer=back-end] \
<name-of-a-QoS-specs-property>=<value-of-the-QoS-specs-property> \
[<name-of-a-QoS-specs-property>=<value-of-the-QoS-specs-property> ...]
```

   b. If you use the openstack command:

```
$ openstack volume qos create [--consumer back-end] \
--property \
<name-of-a-QoS-specs-property>=<value-of-the-QoS-specs-property> \
[--property \
<name-of-a-QoS-specs-property>=<value-of-the-QoS-specs-property> ...] \
<name-of-the-QoS-specs>
```

   Specify a name for <name-of-the-QoS-specs>.

   Specify <name-of-a-QoS-specs-property> and <value-of-the-QoS-specs-property> as follows. For details on the range of values you can specify, see the overview of QoS operations in the Performance Guide.

QoS specs property	Description
upperIops	The upper limit on IOPS.
upperTransferRate	The upper limit on the amount of data transferred in MB/s.
lowerIops	The lower limit on IOPS.
lowerTransferRate	The lower limit on the amount of data transferred in MB/s.
responsePriority	The priority level of the I/O processing.

The following is an example of running the command.

a. If you use the cinder command:

```
$ cinder qos-create test_qos consumer=back-end upperIops=2000
```

b. If you use the openstack command:

```
$ openstack volume qos create --consumer back-end \
--property upperIops=2000 test_qos
```

When you run this command, the ID of the created QoS specs is also output. Record this ID, because you will need it in a later step.

2. Associate the QoS specs with a volume type.

   a. If you use the cinder command:

```
$ cinder qos-associate <ID-of-the-QoS-specs> <ID-of-the-volume-type>
```

   b. If you use the openstack command:

```
$ openstack volume qos associate <name-of-the-QoS-specs> \
<name-of-the-volume-type>
```

3. Specify the volume type that is associated with the QoS specs, and then create a volume.

   a. If you use the cinder command:

```
$ cinder create --volume-type <name-of-the-volume-type> <size>
```

   b. If you use the openstack command:

```
$ openstack volume create --size <size> --type <name-of-the-volume-type> \
<name>
```

Changing QoS settings

To change the QoS settings, use the Retype function to change the volume type to one that has different QoS specs.

You can also change a volume type for which no QoS specs are set to a volume type for which QoS specs are set, and vice versa.

Clearing QoS settings

To clear the QoS settings, clear the association between the volume type and QoS specs, and then delete the QoS specs.

Data deduplication and compression

Use deduplication and compression to improve storage utilization using data reduction.

For details, see Capacity saving function: data deduplication and compression in the Provisioning Guide.

Enabling deduplication and compression

To use the deduplication and compression on the storage models, your storage administrator must first enable the deduplication and compression for the DP pool.

For details about how to enable this setting, see the description of pool management in the Hitachi Command Suite Configuration Manager REST API Reference Guide or the Hitachi Ops Center API Configuration Manager REST API Reference Guide.

Note

- Do not set a subscription limit (virtualVolumeCapacityRate) for the DP pool.

Creating a volume with deduplication and compression enabled

To create a volume with the deduplication and compression setting enabled, enable deduplication and compression for the relevant volume type.

Procedure

1. To enable the deduplication and compression setting, specify the value `deduplication_compression` for `hbsd:capacity_saving` in the extra specs for the volume type.

2. When creating a volume of the volume type created in the previous step, you can create a volume with the deduplication and compression function enabled.

Deleting a volume with deduplication and compression enabled

The cinder delete command finishes when the storage system starts the LDEV deletion process. The LDEV cannot be reused until the LDEV deletion process is completed on the storage system.

Port scheduler

You can use the port scheduler function to reduce the number of WWNs, which are storage system resource.

In Hitachi block storage driver, if host groups are created automatically, host groups are created for each compute node or VM (in an environment that has a WWN for each VM). If you do not use the port scheduler function, host groups are created and the same WWNs are registered in all of the ports that are specified for the configuration option `hitachi_compute_target_ports` or for the configuration option `hitachi_target_ports`. For Hitachi storage devices, a maximum of 255 host groups and 255 WWNs can be registered for one port. When volumes are attached, the upper limit on the number of WWNs that can be registered might be unexpectedly exceeded.

For the port scheduler function, when the cinder-volume service starts, the Fibre Channel Zone Manager obtains the WWNs of active compute nodes and of active VMs. When volumes are attached, the WWNs are registered in a round-robin procedure, in the same order as the order of ports specified for the configuiration option `hitachi_compute_target_ports` or for the configuiration option `hitachi_target_ports`.

If you want to use the port scheduler function, set the configuration option `hitachi_port_scheduler`.

Note

- Only Fibre Channel is supported. For details about ports, see Fibre Channel connectivity.
- If a host group already exists in any of the ports specified for the configuration option `hitachi_compute_target_ports` or for the configuration option `hitachi_target_ports`, no new host group will be created on those ports.
- Restarting the cinder-volume service re-initializes the round robin scheduling determined by the configuration option `hitachi_compute_target_ports` or the configuration option `hitachi_target_ports`.
- The port scheduler function divides up the active WWNs from each fabric controller and registers them to each port. For this reason, the number of WWNs registered may vary from port to port.

Port assignment using extra specs

Defining particular ports in the Hitachi-supported extra spec `hbsd:target_ports` determines which of the ports specified by the configuration options `hitachi_target_ports` or the configuration option `hitachi_compute_target_ports` are used to create LUN paths during volume attach operations for each volume type.

Note

- Use a comma to separate multiple ports.
- In a Global-Active Device configuration, use the extra spec `hbsd:target_ports` for the primary storage system and the extra spec `hbsd:remote_target_ports` for the secondary storage system.
- In a Global-Active Device configuration, the ports specified for the extra spec `hbsd:target_ports` must be specified for both the configuration options for the primary storage system (`hitachi_target_ports` or `hitachi_compute_target_ports`) and for the secondary storage system (`hitachi_mirror_target_ports` or `hitachi_mirror_compute_target_ports`).

HPE MSA Fibre Channel and iSCSI drivers

The `HPMSAFCDriver` and `HPMSAISCSIDriver` Cinder drivers allow the HPE MSA 2060, 1060, 2050, 1050, 2040, and 1040 arrays to be used for Block Storage in OpenStack deployments.

System requirements

To use the HPMSA drivers, the following are required:

- HPE MSA 2060, 1060, 2050, 1050, 2040 or 1040 array with:
  - iSCSI or FC host interfaces
  - G22x, V270 or I100 firmware or later
- Network connectivity between the OpenStack host and the array management interfaces
- HTTPS or HTTP must be enabled on the array

Supported operations

- Create, delete, attach, and detach volumes.
- Create, list, and delete volume snapshots.
- Create a volume from a snapshot.
- Copy an image to a volume.
- Copy a volume to an image.
- Clone a volume.
- Extend a volume.
- Migrate a volume with back-end assistance.
- Retype a volume.
- Manage and unmanage a volume.

Configuring the array

1. Verify that the array can be managed using an HTTPS connection. HTTP can also be used if `hpmsa_api_protocol=http` is placed into the appropriate sections of the `cinder.conf` file, but this option is deprecated and will be removed in a future release.

   Confirm that virtual pools A and B are present if you plan to use virtual pools for OpenStack storage.

   If you plan to use vdisks instead of virtual pools, create or identify one or more vdisks to be used for OpenStack storage; typically this will mean creating or setting aside one disk group for each of the A and B controllers.

2. Edit the `cinder.conf` file to define a storage back-end entry for each storage pool on the array that will be managed by OpenStack. Each entry consists of a unique section name, surrounded by square brackets, followed by options specified in `key=value` format.

   - The `hpmsa_pool_name` value specifies the name of the storage pool or vdisk on the array.

- The `volume_backend_name` option value can be a unique value, if you wish to be able to assign volumes to a specific storage pool on the array, or a name that is shared among multiple storage pools to let the volume scheduler choose where new volumes are allocated.
- The rest of the options will be repeated for each storage pool in a given array:
  - `volume_driver` specifies the Cinder driver name.
  - `san_ip` specifies the IP addresses or host names of the arrays management controllers.
  - `san_login` and `san_password` specify the username and password of an array user account with `manage` privileges.
  - `driver_use_ssl` should be set to `true` to enable use of the HTTPS protocol.
  - `hpmsa_iscsi_ips` specifies the iSCSI IP addresses for the array if using the iSCSI transport protocol.

In the examples below, two back ends are defined, one for pool A and one for pool B, and a common `volume_backend_name` is used so that a single volume type definition can be used to allocate volumes from both pools.

Example: iSCSI example back-end entries

```
[pool-a]
hpmsa_pool_name = A
volume_backend_name = hpmsa-array
volume_driver = cinder.volume.drivers.san.hp.hpmsa_iscsi.HPMSAISCSIDriver
san_ip = 10.1.2.3,10.1.2.4
san_login = manage
san_password = !manage
hpmsa_iscsi_ips = 10.2.3.4,10.2.3.5
driver_use_ssl = true

[pool-b]
hpmsa_pool_name = B
volume_backend_name = hpmsa-array
volume_driver = cinder.volume.drivers.san.hp.hpmsa_iscsi.HPMSAISCSIDriver
san_ip = 10.1.2.3,10.1.2.4
san_login = manage
san_password = !manage
hpmsa_iscsi_ips = 10.2.3.4,10.2.3.5
driver_use_ssl = true
```

Example: Fibre Channel example back-end entries

```
[pool-a]
hpmsa_pool_name = A
volume_backend_name = hpmsa-array
volume_driver = cinder.volume.drivers.san.hp.hpmsa_fc.HPMSAFCDriver
san_ip = 10.1.2.3,10.1.2.4
san_login = manage
san_password = !manage
driver_use_ssl = true

```

(continues on next page)

(continued from previous page)

```
[pool-b]
hpmsa_pool_name = B
volume_backend_name = hpmsa-array
volume_driver = cinder.volume.drivers.san.hp.hpmsa_fc.HPMSAFCDriver
san_ip = 10.1.2.3,10.1.2.4
san_login = manage
san_password = !manage
driver_use_ssl = true
```

3. If any `volume_backend_name` value refers to a vdisk rather than a virtual pool, add an additional statement `hpmsa_pool_type = linear` to that back end entry.

4. If HTTPS is not enabled in the array, include `hpmsa_api_protocol = http` in each of the back-end definitions.

5. If HTTPS is enabled, you can enable certificate verification with the option `driver_ssl_cert_verify = True`. You may also use the `driver_ssl_cert_path` option to specify the path to a CA_BUNDLE file containing CAs other than those in the default list.

6. Modify the `[DEFAULT]` section of the `cinder.conf` file to add an `enabled_backends` parameter specifying the back-end entries you added, and a `default_volume_type` parameter specifying the name of a volume type that you will create in the next step.

   Example: [DEFAULT] section changes

   ```
   [DEFAULT]
   # ...
   enabled_backends = pool-a,pool-b
   default_volume_type = hpmsa
   ```

7. Create a new volume type for each distinct `volume_backend_name` value that you added to the `cinder.conf` file. The example below assumes that the same `volume_backend_name=hpmsa-array` option was specified in all of the entries, and specifies that the volume type `hpmsa` can be used to allocate volumes from any of them.

   Example: Creating a volume type

   ```
   $ openstack volume type create hpmsa
   $ openstack volume type set --property volume_backend_name=hpmsa-array␣
   ↪hpmsa
   ```

8. After modifying the `cinder.conf` file, restart the `cinder-volume` service.

Driver-specific options

The following table contains the configuration options that are specific to the HPMSA drivers.

Table 26: Description of HPE MSA configuration options

Configuration option = Default value	Description
hpmsa_iscsi_ips = []	(List of String) List of comma-separated target iSCSI IP addresses.
hpmsa_pool_name = A	(String) Pool or Vdisk name to use for volume creation.
hpmsa_pool_type = virtual	(String(choices=[linear, virtual])) linear (for Vdisk) or virtual (for Pool).
hpmsa_api_protocol = https	(String(choices=[http, https])) HPMSA API interface protocol. **DEPRECATED**
hpmsa_verify_certificate = False	(Boolean) Whether to verify HPMSA array SSL certificate. **DEPRECATED**
hpmsa_verify_certificate_pat = None	(String) HPMSA array SSL certificate path. **DEPRECATED**

HPE 3PAR, HPE Primera, HPE Alletra 9k and HPE Alletra MP Driver

The `HPE3PARFCDriver` and `HPE3PARISCSIDriver` drivers, which are based on the Block Storage service (Cinder) plug-in architecture, run volume operations by communicating with the HPE 3PAR, HPE Primera and HPE Alletra 9k storage systems over HTTP, HTTPS, and SSH connections. The HTTP & HTTPS communications use `python-3parclient`, which is part of PyPi.

For information on HPE storage systems, refer to the Alletra Storage product page.

System requirements

To use the HPE 3PAR, HPE Primera, HPE Alletra 9k and HPE Alletra MP drivers, install the following software and components on the HPE 3PAR storage system:

- HPE 3PAR Operating System software version 3.1.3 MU1 or higher.
  - Deduplication provisioning requires SSD disks and HPE 3PAR Operating System software version 3.2.1 MU1 or higher.
  - Enabling Flash Cache Policy requires the following:
    - Array must contain SSD disks.
    - HPE 3PAR Operating System software version 3.2.1 MU2 or higher.
    - python-3parclient version 4.2.0 or newer.
    - Flash Cache must be enabled on the array with the CLI command **`createflashcache SIZE`**, where size must be in 16 GB increments. For example, **`createflashcache 128g`** will create 128 GB of Flash Cache for each node pair in the array.
  - The Dynamic Optimization is required to support any feature that results in a volume changing provisioning type or CPG. This may apply to the volume **`migrate`**, **`retype`** and **`manage`** commands.
  - The Virtual Copy feature supports any operation that involves volume snapshots. This applies to the volume **`snapshot-*`** commands.
  - Enabling Volume Compression requires the following:

        * Array must contain SSD disks.
        * HPE 3PAR Operating System software version 3.3.1 MU1 or higher.
        * HPE 3PAR Storage System with 8k or 20k series

- HPE 3PAR Web Services API Server must be enabled and running.
- One Common Provisioning Group (CPG).
- Additionally, you must install the `python-3parclient` version 4.2.0 or newer from PyPi on the system with the enabled Block Storage service volume drivers.

To use the HPE Primera, Alletra 9k and Alletra MP backends, install the following software and components on the HPE storage system:

- Operating System software:
  - HPE Primera: version 4.4.0 or higher.
  - HPE Alletra 9k: version 9.4.0 or higher.
  - HPE Alletra MP: version 10.4.2.23 or higher.
- On HPE Primera/Alletra 9k/Alletra MP storage system, Dedup & Compression is combined as single option deco. Due to this, only either thin volume or deco volume can be created.
- Also, port number 443 is used instead of 8080. This only affects cinder configuration.
- Additionally, you must install the `python-3parclient` version 4.2.14 or newer from PyPi on the system with the enabled Block Storage service volume drivers.

Supported operations

- Create, delete, attach, and detach volumes.
- Create, list, and delete volume snapshots.
- Create a volume from a snapshot.
- Copy an image to a volume.
- Copy a volume to an image.
- Clone a volume.
- Extend a volume.
- Migrate a volume with back-end assistance.
- Retype a volume.
- Manage and unmanage a volume.
- Manage and unmanage a snapshot.
- Replicate host volumes.
- Fail-over host volumes.
- Fail-back host volumes.
- Retype a replicated volume.
- Create, delete, update, snapshot, and clone generic volume groups.

- Create and delete generic volume group snapshots.
- Create a generic volume group from a group snapshot or another group.
- Volume Compression.
- Group Replication with More Granularity (Tiramisu).
- Volume Revert to Snapshot.
- Additional Backend Capabilities.
- Report Backend State in Service List.
- Attach a volume to multiple servers simultaneously (multiattach).
- Peer Persistence.

Volume type support for both HPE 3PAR drivers includes the ability to set the following capabilities in the OpenStack Block Storage API `cinder.api.contrib.types_extra_specs` volume type extra specs extension module:

- `hpe3par:snap_cpg`
- `hpe3par:provisioning`
- `hpe3par:persona`
- `hpe3par:vvs`
- `hpe3par:flash_cache`
- `hpe3par:compression`

To work with the default filter scheduler, the key values are case sensitive and scoped with `hpe3par:`. For information about how to set the key-value pairs and associate them with a volume type, run the following command:

```
$ openstack help volume type
```

Note

Volumes that are cloned only support the extra specs keys cpg, snap_cpg, provisioning and vvs. The others are ignored. In addition the comments section of the cloned volume in the HPE 3PAR / Primera / Alletra 9k / Alletra MP array is not populated.

If volume types are not used or a particular key is not set for a volume type, the following defaults are used:

- `hpe3par:cpg` - Defaults to the `hpe3par_cpg` setting in the `cinder.conf` file.
- `hpe3par:snap_cpg` - Defaults to the `hpe3par_snap` setting in the `cinder.conf` file. If `hpe3par_snap` is not set, it defaults to the `hpe3par_cpg` setting.
- `hpe3par:provisioning` - Defaults to `thin` provisioning, the valid values are `thin`, `full`, and `dedup`.
- `hpe3par:persona` - Defaults to the `2 - Generic-ALUA` persona. The valid values are:
  - `1 - Generic`

  - 2 - Generic-ALUA
  - 3 - Generic-legacy
  - 4 - HPUX-legacy
  - 5 - AIX-legacy
  - 6 - EGENERA
  - 7 - ONTAP-legacy
  - 8 - VMware
  - 9 - OpenVMS
  - 10 - HPUX
  - 11 - WindowsServer
- `hpe3par:flash_cache` - Defaults to `false`, the valid values are `true` and `false`.

QoS support for both HPE 3PAR drivers includes the ability to set the following capabilities in the OpenStack Block Storage API `cinder.api.contrib.qos_specs_manage` qos specs extension module:

- `minBWS`
- `maxBWS`
- `minIOPS`
- `maxIOPS`
- `latency`
- `priority`

The qos keys above no longer require to be scoped but must be created and associated to a volume type. For information about how to set the key-value pairs and associate them with a volume type, run the following commands:

```
$ openstack help volume qos
```

The following keys require that the HPE 3PAR/Primera/Alletra 9k/ Alletra MP array has a Priority Optimization enabled.

hpe3par:vvs
: The virtual volume set name that has been predefined by the Administrator with quality of service (QoS) rules associated to it. If you specify extra_specs `hpe3par:vvs`, the qos_specs `minIOPS`, `maxIOPS`, `minBWS`, and `maxBWS` settings are ignored.

minBWS
: The QoS I/O issue bandwidth minimum goal in MBs. If not set, the I/O issue bandwidth rate has no minimum goal.

maxBWS
: The QoS I/O issue bandwidth rate limit in MBs. If not set, the I/O issue bandwidth rate has no limit.

minIOPS
: The QoS I/O issue count minimum goal. If not set, the I/O issue count has no minimum goal.

maxIOPS
The QoS I/O issue count rate limit. If not set, the I/O issue count rate has no limit.

latency
The latency goal in milliseconds.

priority
The priority of the QoS rule over other rules. If not set, the priority is `normal`, valid values are `low`, `normal` and `high`.

> **Note**
>
> Since the Icehouse release, minIOPS and maxIOPS must be used together to set I/O limits. Similarly, minBWS and maxBWS must be used together. If only one is set the other will be set to the same value.

The following key requires that the HPE 3PAR/Primera/Alletra 9k/Alletra MP array has an Adaptive Flash Cache enabled.

- `hpe3par:flash_cache` - The flash-cache policy, which can be turned on and off by setting the value to `true` or `false`.
- `hpe3par:compression` - The volume compression, which can be turned on and off by setting the value to `true` or `false`.

Other restrictions and considerations for `hpe3par:compression`:

- For a compressed volume, minimum volume size needed is 16 GB; otherwise resulting volume will be created successfully but will not be a compressed volume.
- A full provisioned volume cannot be compressed, if a compression is enabled and provisioning type requested is full, the resulting volume defaults to thinly provisioned compressed volume.
- While creating volume on HPE Primera/Alletra 9k/Alletra MP storage system, only below two combinations are supported. If any other combination is used, then volume is not created.
  - thin volume: provisioning = `thin` and compression = `false`
  - deco volume: provisioning = `dedup` and compression = `true`

LDAP and AD authentication is now supported in the HPE 3PAR driver.

The 3PAR back end must be properly configured for LDAP and AD authentication prior to configuring the volume driver. For details on setting up LDAP with 3PAR, see the 3PAR user guide.

Once configured, `hpe3par_username` and `hpe3par_password` parameters in `cinder.conf` can be used with LDAP and AD credentials.

Enable the HPE 3PAR Fibre Channel and iSCSI drivers

The `HPE3PARFCDriver` and `HPE3PARISCSIDriver` are installed with the OpenStack software.

1. Install the `python-3parclient` Python package on the OpenStack Block Storage system.

   ```
   $ pip install 'python-3parclient>=4.0,<5.0'
   ```

2. Verify that the HPE 3PAR Web Services API server is enabled and running on the HPE 3PAR / Primera / Alletra 9k / Alletra MP storage system.

a. Log onto the HPE 3PAR / Primera / Alletra 9k / Alletra MP storage system with administrator access.

```
$ ssh 3paradm@<HPE storage system IP Address>
```

b. View the current state of the Web Services API Server.

```
$ showwsapi
-Service- -State- -HTTP_State- HTTP_Port -HTTPS_State- HTTPS_Port -
↪Version-
Enabled   Active Enabled       8008      Enabled       8080       ␣
↪1.1
```

c. If the Web Services API Server is disabled, start it.

```
$ startwsapi
```

3. If the HTTP or HTTPS state is disabled, enable one of them.

```
$ setwsapi -http enable
```

or

```
$ setwsapi -https enable
```

Note

To stop the Web Services API Server, use the **stopwsapi** command. For other options run the **setwsapi -h** command.

4. If you are not using an existing CPG, create a CPG on the HPE 3PAR / Primera / Alletra 9k / Alletra MP storage system to be used as the default location for creating volumes.

5. Make the following changes in the `/etc/cinder/cinder.conf` file.

```
# WSAPI Server URL.
# This setting applies to all: 3PAR, Primera, Alletra 9k and Alletra MP.
# Example 1: for 3PAR, URL is:
https://<3par ip>:8080/api/v1

# Example 2: for Primera/Alletra 9k/Alletra MP, URL is:
https://<primera/alletra_9k/alletra_mp ip>:443/api/v1

# 3PAR / Primera / Alletra 9k / Alletra MP username with the 'edit' role
hpe3par_username=edit3par

# 3PAR / Primera / Alletra 9k / Alletra MP password for the user␣
↪specified in hpe3par_username
hpe3par_password=3parpass

# 3PAR / Primera / Alletra 9k / Alletra MP CPG to use for volume creation
```

(continues on next page)

(continued from previous page)

```
hpe3par_cpg=OpenStackCPG_RAID5_NL

# IP address of SAN controller for SSH access to the array
san_ip=10.10.22.241

# Username for SAN controller for SSH access to the array
san_login=3paradm

# Password for SAN controller for SSH access to the array
san_password=3parpass

# FIBRE CHANNEL DRIVER
# (uncomment the next line to enable the FC driver)
#volume_driver=cinder.volume.drivers.hpe.hpe_3par_fc.HPE3PARFCDriver

# iSCSI DRIVER
# If you enable the iSCSI driver, you must also set values
# for hpe3par_iscsi_ips or iscsi_ip_address in this file.
# Note: The iSCSI driver is supported with 3PAR (all versions)
# and Primera (version 4.2 or higher). If you configure iSCSI
# with Primera 4.0 or 4.1, the driver will fail to start.
# (uncomment the next line to enable the iSCSI driver)
#volume_driver=cinder.volume.drivers.hpe.hpe_3par_iscsi.
↪HPE3PARISCSIDriver

# iSCSI multiple port configuration
# hpe3par_iscsi_ips=10.10.220.253:3261,10.10.222.234

# Still available for single port iSCSI configuration
#iscsi_ip_address=10.10.220.253

# Enable HTTP debugging to 3PAR / Primera / Alletra 9k / Alletra MP
hpe3par_debug=False

# Enable CHAP authentication for iSCSI connections.
hpe3par_iscsi_chap_enabled=false

# The CPG to use for Snapshots for volumes. If empty hpe3par_cpg will be
# used.
hpe3par_cpg_snap=OpenStackSNAP_CPG

# Time in hours to retain a snapshot. You can't delete it before this
# expires.
hpe3par_snapshot_retention=48

# Time in hours when a snapshot expires and is deleted. This must be
# larger than retention.
hpe3par_snapshot_expiration=72
```

(continues on next page)

(continued from previous page)

```
# The ratio of oversubscription when thin provisioned volumes are
# involved. Default ratio is 20.0, this means that a provisioned
# capacity can be 20 times of the total physical capacity.
max_over_subscription_ratio=20.0

# This flag represents the percentage of reserved back-end capacity.
reserved_percentage=15
```

Note

You can enable only one driver on each cinder instance unless you enable multiple back-end support. See the Cinder multiple back-end support instructions to enable this feature.

Note

You can configure one or more iSCSI addresses by using the `hpe3par_iscsi_ips` option. Separate multiple IP addresses with a comma (`,`). When you configure multiple addresses, the driver selects the iSCSI port with the fewest active volumes at attach time. The 3PAR array does not allow the default port 3260 to be changed, so IP ports need not be specified.

6. Save the changes to the `cinder.conf` file and restart the cinder-volume service.

The HPE 3PAR Fibre Channel and iSCSI drivers are now enabled on your OpenStack system. If you experience problems, review the Block Storage service log files for errors.

The following table contains all the configuration options supported by the HPE 3PAR Fibre Channel and iSCSI drivers.

Table 27: Description of 3PAR configuration options

Configuration option = Default value	Description
hpe3par_api = <>	(String) WSAPI Server URL. This setting applies to: 3PAR, Primera, Alletra 9k and Alletra MP Example 1: for 3PAR, URL is: https://<3par ip>:8080/api/v1 Example 2: for Primera/Alletra 9k/Alletra MP, URL is: https://<primera ip>:443/api/v1
hpe3par_cpg = [OpenStack]	(List of String) List of the 3PAR/Primera/Alletra 9k/Alletra MP CPG(s) to use for volume creation
hpe3par_cpg = <>	(String) The 3PAR/Primera/Alletra 9k/Alletra MP CPG to use for snapshots of volumes. If empty the userCPG will be used
hpe3par_deb = False	(Boolean) Enable HTTP debugging to 3PAR/Primera/Alletra 9k/Alletra MP
hpe3par_isc = False	(Boolean) Enable CHAP authentication for iSCSI connections.
hpe3par_isc = []	(List of String) List of target iSCSI addresses to use.
hpe3par_pas = <>	(String) 3PAR/Primera/Alletra 9k/Alletra MP password for the user specified in hpe3par_username
hpe3par_sna = <>	(String) The time in hours when a snapshot expires and is deleted. This must be larger than expiration
hpe3par_sna = <>	(String) The time in hours to retain a snapshot. You cant delete it before this expires.
hpe3par_tar = <>	(String) The nsp of 3PAR/Primera/Alletra 9k/Alletra MP backend to be used when: (1) multipath is not enabled in cinder.conf. (2) Fiber Channel Zone Manager is not used. (3) the backend is prezoned with this specific nsp only. For example if nsp is 2 1 2, the format of the options value is 2:1:2
hpe3par_use = <>	(String) 3PAR/Primera/Alletra 9k/Alletra MP username with the edit role

Specify NSP for FC Bootable Volume

Given a system connected to HPE 3PAR via FC and multipath setting is NOT used in cinder.conf. When the user tries to create a bootable volume, it fails intermittently with the following error: Fibre Channel volume device not found

This happens when a zone is created using second or later target from 3PAR backend. In this case, HPE 3PAR client code picks up first target to form initiator target map. This can be illustrated with below example.

Sample output of showport command:

```
$ showport -sortcol 6
```

```
N:S:P      Mode State ----Node_WWN---- -Port_WWN/HW_Addr-  Type Protocol␣
↪Partner FailoverState
0:1:1    target ready 2FF70002AC002DB6   20110002AC002DB6  host       FC     ␣
↪ -            -
```

(continues on next page)

(continued from previous page)

```
0:1:2    target ready 2FF70002AC002DB6   20120002AC002DB6  host       FC ␣
↪1:1:2          none
1:1:1 initiator ready 2FF70002AC002DB6   21110002AC002DB6  rcfc       FC    ␣
↪ -             -
1:1:2    target ready 2FF70002AC002DB6   21120002AC002DB6  host       FC ␣
↪0:1:2          none
2:1:1 initiator ready 2FF70002AC002DB6   22110002AC002DB6  rcfc       FC    ␣
↪ -             -
2:1:2    target ready 2FF70002AC002DB6   22120002AC002DB6  host       FC ␣
↪3:1:2          none
3:1:1    target ready 2FF70002AC002DB6   23110002AC002DB6  host       FC    ␣
↪ -             -
3:1:2    target ready 2FF70002AC002DB6   23120002AC002DB6  host       FC ␣
↪2:1:2          none
```

Suppose zone is created using targets 2:1:2 and 3:1:2 from above output. Then initiator target map is created using target 0:1:1 only. In such a case, the path is not found, and bootable volume creation fails.

To avoid above mentioned failure, the user can specify the target in 3PAR backend section of cinder.conf as follows:

`hpe3par_target_nsp = 3:1:2`

Using above mentioned nsp, respective wwn information is fetched. Later initiator target map is created using wwn information and bootable volume is created successfully.

Note: If above mentioned option (nsp) is not specified in cinder.conf, then the original flow is executed i.e first target is picked and bootable volume creation may fail.

Peer Persistence support

Given 3PAR/Primera backend configured with replication setup, currently only Active/Passive replication is supported by 3PAR/Primera in OpenStack. When failover happens, nova does not support volume force-detach (from dead primary backend) / re-attach to secondary backend. Storage engineers manual intervention is required.

To overcome above scenario, support for Peer Persistence is added. Given a system with Peer Persistence configured and replicated volume is created. When this volume is attached to an instance, vlun is created automatically in secondary backend, in addition to primary backend. So that when a failover happens, it is seamless.

For Peer Persistence support, perform following steps: 1] enable multipath 2] set replication mode as sync 3] configure a quorum witness server

Specify ip address of quorum witness server in `/etc/cinder/cinder.conf` [within backend section] as given below:

```
[3pariscsirep]
hpe3par_api_url = http://10.50.3.7:8008/api/v1
hpe3par_username = <user_name>
hpe3par_password = <password>
...
<other parameters>
```

(continues on next page)

(continued from previous page)

```
...
replication_device = backend_id:CSIM-EOS12_1611702,
                     replication_mode:sync,
                     quorum_witness_ip:10.50.3.192,
                     hpe3par_api_url:http://10.50.3.22:8008/api/v1,
                     ...
                     <other parameters>
                     ...
```

Support duplicated FQDN in network

The 3PAR driver uses the FQDN of the node that is doing the attach as an unique identifier to map the volume.

The problem is that the FQDN is not always unique, there are environments where the same FQDN can be found in different systems, and in those cases if both try to attach volumes the second system will fail.

One example of this happening would be on a QA environment where you are creating VMs and they all have names like controller-0.localdomain and compute-0.localdomain.

To support these kind of environments, the user can specify below flag in backend_defaults section or the specific cinder driver section of cinder.conf as follows:

`unique_fqdn_network = False`

When this flag is used, then during attach volume to instance, iscsi initiator name is used instead of FQDN.

If above mentioned flag is not specified in cinder.conf, then its value is considered as True (by default) and FQDN is used (existing behavior).

HPE XP block storage driver

HPE XP block storage driver provides Fibre Channel and iSCSI support for HPE XP storages.

System requirements

Supported storages:

Storage model	Firmware version
XP8	90-01-41 or later
XP7	80-05-43 or later

Required storage licenses:

- Thin Provisioning
- Fast Snap

Supported operations

- Create, delete, attach, and detach volumes.
- Create, list, and delete volume snapshots.
- Create a volume from a snapshot.
- Create, list, update, and delete consistency groups.
- Create, list, and delete consistency group snapshots.
- Copy a volume to an image.
- Copy an image to a volume.
- Clone a volume.
- Extend a volume.
- Migrate a volume.
- Get volume statistics.
- Efficient non-disruptive volume backup.
- Manage and unmanage a volume.
- Attach a volume to multiple instances at once (multi-attach).
- Revert a volume to a snapshot.

Note

The volume having snapshots cannot be extended in this driver.

Configuration

Set up HPE XP storage

You need to specify settings as described below for storage systems. For details about each setting, see the users guide of the storage systems.

1. User accounts

   Create a storage device account belonging to the Administrator User Group.

2. THP pool

   Create a THP pool that is used by the driver.

3. Ports

   Enable Port Security for the ports used by the driver.

Set up HPE XP storage volume driver

Set the volume driver to HPE XP block storage driver by setting the volume_driver option in the cinder.conf file as follows:

If you use Fibre Channel:

```
[hpe_xp]
volume_driver = cinder.volume.drivers.hpe.xp.hpe_xp_fc.HPEXPFCDriver
volume_backend_name = hpexp_fc
san_ip = 1.2.3.4
san_login = hpexpuser
san_password = password
hpexp_storage_id = 123456789012
hpexp_pools = pool0
```

If you use iSCSI:

```
[hpe_xp]
volume_driver = cinder.volume.drivers.hpe.xp.hpe_xp_iscsi.HPEXPISCSIDriver
volume_backend_name = hpexp_iscsi
san_ip = 1.2.3.4
san_login = hpexpuser
san_password = password
hpexp_storage_id = 123456789012
hpexp_pools = pool0
```

This table shows configuration options for HPE XP block storage driver.

Table 28: Descrip
tion options

Configuration option = Default value	Description
hpexp_async_copy_check_interval = 10	(Integer(min=1, max=600)) Interval in seconds to c
hpexp_compute_target_ports = []	(List of String) IDs of the storage ports used to attac
hpexp_copy_check_interval = 3	(Integer(min=1, max=600)) Interval in seconds to c
hpexp_copy_speed = 3	(Integer(min=1, max=15)) Copy speed of storage sy
hpexp_discard_zero_page = True	(Boolean) Enable or disable zero page reclamation i
hpexp_exec_retry_interval = 5	(Integer) Retry interval in seconds for REST API ex
hpexp_extend_timeout = 600	(Integer) Maximum wait time in seconds for a volur
hpexp_group_create = False	(Boolean) If True, the driver will create host groups
hpexp_group_delete = False	(Boolean) If True, the driver will delete host groups
hpexp_host_mode_options = []	(List of String) Host mode option for host group or
hpexp_ldev_range = None	(String) Range of the LDEV numbers in the format
hpexp_lock_timeout = 7200	(Integer) Maximum wait time in seconds for storage
hpexp_lun_retry_interval = 1	(Integer) Retry interval in seconds for REST API ad
hpexp_lun_timeout = 50	(Integer) Maximum wait time in seconds for adding
hpexp_pools = []	(List of String) Pool number[s] or pool name[s] of t
hpexp_rest_another_ldev_mapped_retry_timeout = 600	(Integer) Retry time in seconds when new LUN allo
hpexp_rest_connect_timeout = 30	(Integer) Maximum wait time in seconds for REST
hpexp_rest_disable_io_wait = True	(Boolean) It may take some time to detach volume a
hpexp_rest_get_api_response_timeout = 1800	(Integer) Maximum wait time in seconds for a respo
hpexp_rest_job_api_response_timeout = 1800	(Integer) Maximum wait time in seconds for a respo
hpexp_rest_keep_session_loop_interval = 180	(Integer) Loop interval in seconds for keeping REST
hpexp_rest_server_busy_timeout = 7200	(Integer) Maximum wait time in seconds when RES
hpexp_rest_tcp_keepalive = True	(Boolean) Enables or disables use of REST API tcp
hpexp_rest_tcp_keepcnt = 4	(Integer) Maximum number of transmissions for TC

Table 2

Configuration option = Default value	Description
hpexp_rest_tcp_keepidle = 60	(Integer) Wait time in seconds for sending a first TC
hpexp_rest_tcp_keepintvl = 15	(Integer) Interval of transmissions in seconds for TC
hpexp_rest_timeout = 30	(Integer) Maximum wait time in seconds for REST
hpexp_restore_timeout = 86400	(Integer) Maximum wait time in seconds for the res
hpexp_snap_pool = None	(String) Pool number or pool name of the snapshot
hpexp_state_transition_timeout = 900	(Integer) Maximum wait time in seconds for a volu
hpexp_storage_id = None	(String) Product number of the storage system.
hpexp_target_ports = []	(List of String) IDs of the storage ports used to atta
hpexp_zoning_request = False	(Boolean) If True, the driver will configure FC zoni

Required options

- **san_ip**
  IP address of SAN controller
- **san_login**
  Username for SAN controller
- **san_password**
  Password for SAN controller
- **hpexp_storage_id**
  Product number of the storage system.
- **hpexp_pools**
  Pool number(s) or pool name(s) of the THP pool.

Huawei volume driver

Huawei volume driver can be used to provide functions such as the logical volume and snapshot for virtual machines (VMs) in the OpenStack Block Storage driver that supports iSCSI and Fibre Channel protocols.

Version mappings

The following table describes the version mappings among the Block Storage driver, Huawei storage system and OpenStack:

Table 29: **Version mappings among the Block Storage driver and Huawei storage system**

Description	Storage System Version
Create, delete, expand, attach, detach, manage and unmanage volumes Create volumes with assigned storage pools Create volumes with assigned disk types Create, delete and update a consistency group Copy an image to a volume Copy a volume to an image Auto Zoning SmartThin Volume Migration Replication V2.1 Create, delete, manage, unmanage and backup snapshots Create and delete a cgsnapshot	OceanStor T series V2R2 C00/C20/C30 OceanStor V3 V3R1C10/C20 V3R2C10 V3R3C00/C10/C20 OceanStor 2200V3 V300R005C00 OceanStor 2600V3 V300R005C00 OceanStor 18500/18800 V1R1C00/C20/C30 V3R3C00 OceanStor Dorado V300R001C00 OceanStor V3 V300R006C00 OceanStor 2200V3 V300R006C00 OceanStor 2600V3 V300R006C00
Clone a volume Create volume from snapshot Retype SmartQoS SmartTier SmartCache Thick	OceanStor T series V2R2 C00/C20/C30 OceanStor V3 V3R1C10/C20 V3R2C10 V3R3C00/C10/C20 OceanStor 2200V3 V300R005C00 OceanStor 2600V3 V300R005C00 OceanStor 18500/18800V1R1C00/C20/C30 OceanStor V3 V300R006C00 OceanStor 2200V3 V300R006C00 OceanStor 2600V3 V300R006C00
SmartPartition	OceanStor T series V2R2 C00/C20/C30 OceanStor V3 V3R1C10/C20 V3R2C10 V3R3C00/C10/C20 OceanStor 2600V3 V300R005C00 OceanStor 18500/18800V1R1C00/C20/C30 OceanStor V3 V300R006C00 OceanStor 2600V3 V300R006C00
Hypermetro Hypermetro consistency group	OceanStor V3 V3R3C00/C10/C20 OceanStor 2600V3 V3R5C00 OceanStor 18500/18800 V3R3C00 OceanStor Dorado V300R001C00 OceanStor V3 V300R006C00 OceanStor 2600V3 V300R006C00

Volume driver configuration

This section describes how to configure the Huawei volume driver for either iSCSI storage or Fibre Channel storage.

Pre-requisites

When creating a volume from image, install the `multipath` tool and add the following configuration keys for each backend section or in `[backend_defaults]` section as a common configuration for all backends in `/etc/cinder/cinder.conf` file:

```
use_multipath_for_image_xfer = True
enforce_multipath_for_image_xfer = True
```

To configure the volume driver, follow the steps below:

1. In /etc/cinder, create a Huawei-customized driver configuration file. The file format is XML.
2. Change the name of the driver configuration file based on the site requirements, for example, cinder_huawei_conf.xml.
3. Configure parameters in the driver configuration file.

   Each product has its own value for the Product parameter under the Storage xml block. The full xml file with the appropriate Product parameter is as below:

```
<?xml version="1.0" encoding="UTF-8"?>
   <config>
      <Storage>
         <Product>PRODUCT</Product>
         <Protocol>PROTOCOL</Protocol>
         <UserName>xxxxxxxx</UserName>
         <UserPassword>xxxxxxxx</UserPassword>
         <RestURL>https://x.x.x.x:8088/deviceManager/rest/</RestURL>
      </Storage>
      <LUN>
         <LUNType>xxx</LUNType>
         <WriteType>xxx</WriteType>
         <Prefetch Type="xxx" Value="xxx" />
         <StoragePool>xxx</StoragePool>
      </LUN>
      <iSCSI>
         <DefaultTargetIP>x.x.x.x</DefaultTargetIP>
         <Initiator Name="xxxxxxxx" TargetIP="x.x.x.x"/>
      </iSCSI>
      <Host OSType="Linux" HostIP="x.x.x.x, x.x.x.x"/>
   </config>

The corresponding ``Product`` values for each product are as below:
```

   - **For T series V2**

   ```
   <Product>TV2</Product>
   ```

   - **For V3**

   ```
   <Product>V3</Product>
   ```

   - **For OceanStor 18000 series**

   ```
   <Product>18000</Product>
   ```

   - **For OceanStor Dorado series**

```
<Product>Dorado</Product>
```

The `Protocol` value to be used is `iSCSI` for iSCSI and `FC` for Fibre Channel as shown below:

```
# For iSCSI
<Protocol>iSCSI</Protocol>

# For Fibre channel
<Protocol>FC</Protocol>
```

Note

For details about the parameters in the configuration file, see the *Configuration file parameters* section.

4. Configure the `cinder.conf` file.

   In the `[default]` block of `/etc/cinder/cinder.conf`, enable the `VOLUME_BACKEND`:

```
enabled_backends = VOLUME_BACKEND
```

   Add a new block `[VOLUME_BACKEND]`, and add the following contents:

```
[VOLUME_BACKEND]
volume_driver = VOLUME_DRIVER
cinder_huawei_conf_file = /etc/cinder/cinder_huawei_conf.xml
volume_backend_name = Huawei_Storage
```

   - `volume_driver` indicates the loaded driver.
   - `cinder_huawei_conf_file` indicates the specified Huawei-customized configuration file.
   - `volume_backend_name` indicates the name of the backend.

   Add information about remote devices in `/etc/cinder/cinder.conf` in target backend block for `Hypermetro`.

```
[VOLUME_BACKEND]
volume_driver = VOLUME_DRIVER
cinder_huawei_conf_file = /etc/cinder/cinder_huawei_conf.xml
volume_backend_name = Huawei_Storage
metro_san_user = xxx
metro_san_password = xxx
metro_domain_name = xxx
metro_san_address = https://x.x.x.x:8088/deviceManager/rest/
metro_storage_pools = xxx
```

   Add information about remote devices in `/etc/cinder/cinder.conf` in target backend block for `Replication`.

```
[VOLUME_BACKEND]
volume_driver = VOLUME_DRIVER
```

(continues on next page)

(continued from previous page)

```
cinder_huawei_conf_file = /etc/cinder/cinder_huawei_conf.xml
volume_backend_name = Huawei_Storage
replication_device =
    backend_id: xxx,
    storage_pool :xxx,
    san_address: https://x.x.x.x:8088/deviceManager/rest/,
    san_user: xxx,
    san_password: xxx,
    iscsi_default_target_ip: x.x.x.x
```

Note

By default, the value for `Hypermetro` and `Replication` is `None`. For details about the parameters in the configuration file, see the *Configuration file parameters* section.

The `volume-driver` value for every product is as below:

```
# For iSCSI
volume_driver = cinder.volume.drivers.huawei.huawei_driver.
↪HuaweiISCSIDriver

# For FC
volume_driver = cinder.volume.drivers.huawei.huawei_driver.HuaweiFCDriver
```

5. Run the **service cinder-volume restart** command to restart the Block Storage service.

Configuring iSCSI Multipathing

To configure iSCSI Multipathing, follow the steps below:

1. Add the port group settings in the Huawei-customized driver configuration file and configure the port group name needed by an initiator.

```
<iSCSI>
   <DefaultTargetIP>x.x.x.x</DefaultTargetIP>
   <Initiator Name="xxxxxx" TargetPortGroup="xxxx" />
</iSCSI>
```

2. Enable the multipathing switch of the Compute service module.

   Add `volume_use_multipath = True` in `[libvirt]` of `/etc/nova/nova.conf`.

3. Run the **service nova-compute restart** command to restart the `nova-compute` service.

Configuring FC Multipathing

To configure FC Multipathing, follow the steps below:

1. Enable the multipathing switch of the Compute service module.

   Add `volume_use_multipath = True` in `[libvirt]` of `/etc/nova/nova.conf`.

2. Run the **service nova-compute restart** command to restart the `nova-compute` service.

Configuring CHAP and ALUA

On a public network, any application server whose IP address resides on the same network segment as that of the storage systems iSCSI host port can access the storage system and perform read and write operations in it. This poses risks to the data security of the storage system. To ensure the storage systems access security, you can configure CHAP authentication to control application servers access to the storage system.

Adjust the driver configuration file as follows:

```
<Initiator ALUA="xxx" CHAPinfo="xxx" Name="xxx" TargetIP="x.x.x.x"/>
```

ALUA indicates a multipathing mode. 0 indicates that ALUA is disabled. 1 indicates that ALUA is enabled. CHAPinfo indicates the user name and password authenticated by CHAP. The format is mmuser; mm-user@storage. The user name and password are separated by semicolons (;).

Configuring multiple storage

Multiple storage systems configuration example:

```
enabled_backends = v3_fc, 18000_fc
[v3_fc]
volume_driver = cinder.volume.drivers.huawei.huawei_driver.HuaweiFCDriver
cinder_huawei_conf_file = /etc/cinder/cinder_huawei_conf_v3_fc.xml
volume_backend_name = huawei_v3_fc
[18000_fc]
volume_driver = cinder.volume.drivers.huawei.huawei_driver.HuaweiFCDriver
cinder_huawei_conf_file = /etc/cinder/cinder_huawei_conf_18000_fc.xml
volume_backend_name = huawei_18000_fc
```

Configuration file parameters

This section describes mandatory and optional configuration file parameters of the Huawei volume driver.

Table 30: **Mandatory parameters**

Parameter	Default value	Description	Applicable to
Product	-	Type of a storage product. Possible values are `TV2`, `18000` and `V3`.	All
Protocol	-	Type of a connection protocol. The possible value is either `'iSCSI'` or `'FC'`.	All
RestURL	-	Access address of the REST interface, `https://x.x.x.x/devicemanager/rest/`. The value `x.x.x.x` indicates the management IP address. OceanStor 18000 uses the preceding setting, and V2 and V3 requires you to add port number `8088`, for example, `https://x.x.x.x:8088/deviceManager/rest/`. If you need to configure multiple RestURL, separate them by semicolons (;).	All
UserName	-	User name of a storage administrator.	All
UserPassword	-	Password of a storage administrator.	All
StoragePool	-	Name of a storage pool to be used. If you need to configure multiple storage pools, separate them by semicolons (`;`).	All

Note

The value of `StoragePool` cannot contain Chinese characters.

Table 31: **Optional parameters**

Parameter	Default value	Description	Applicable to
LUNType	Thick	Type of the LUNs to be created. The value can be `Thick` or `Thin`. Dorado series only support `Thin` LUNs.	All
WriteType	1	Cache write type, possible values are: `1` (write back), `2` (write through), and `3` (mandatory write back).	All
LUNcopyWaitInterval	5	After LUN copy is enabled, the plug-in frequently queries the copy progress. You can set a value to specify the query interval.	All
Timeout	432000	Timeout interval for waiting LUN copy of a storage device to complete. The unit is second.	All
Initiator Name	-	Name of a compute node initiator.	All
Initiator TargetIP	-	IP address of the iSCSI port provided for compute nodes.	All
Initiator TargetPortGroup	-	IP address of the iSCSI target port that is provided for compute nodes.	All
DefaultTargetIP	-	Default IP address of the iSCSI target port that is provided for compute nodes.	All
OSType	Linux	Operating system of the Nova compute nodes host.	All
HostIP	-	IP address of the Nova compute nodes host.	All
metro_san_user	-	User name of a storage administrator of hypermetro remote device.	V3R3/2600 V3R5/18000 V3R3
metro_san_passworc	-	Password of a storage administrator of hypermetro remote device.	V3R3/2600 V3R5/18000 V3R3
metro_domain_nam	-	Hypermetro domain name configured on ISM.	V3R3/2600 V3R5/18000 V3R3
metro_san_address	-	Access address of the REST interface, https://x.x.x.x/devicemanager/rest/. The value x.x.x.x indicates the management IP address.	V3R3/2600 V3R5/18000 V3R3
metro_storage_pools	-	Remote storage pool for hypermetro.	V3R3/2600 V3R5/18000 V3R3
backend_id	-	Target device ID.	All
storage_pool	-	Pool name of target backend when failover for replication.	All
san_address	-	Access address of the REST interface, https://x.x.x.x/devicemanager/rest/. The value x.x.x.x indicates the management IP address.	All
san_user	-	User name of a storage administrator of replication remote device.	All
san_password	-	Password of a storage administrator of replication remote device.	All
iscsi_default_target_	-	Remote transaction port IP.	All

Important

The `Initiator Name`, `Initiator TargetIP`, and `Initiator TargetPortGroup` are `ISCSI` parameters and therefore not applicable to `FC`.

The following are the Huawei driver specific options that may be set in *cinder.conf*:

Table 32: Description of Huawei configuration options

Configuration option = Default value	Description
`cinder_huawei_conf_file = /etc/cinder/cinder_huawei_conf.xml`	(String) The configuration file for the Cinder Huawei driver.
`hypermetro_devices = None`	(String) The remote device hypermetro will use.
`metro_domain_name = None`	(String) The remote metro device domain name.
`metro_san_address = None`	(String) The remote metro device request url.
`metro_san_password = None`	(String) The remote metro device san password.
`metro_san_user = None`	(String) The remote metro device san user.
`metro_storage_pools = None`	(String) The remote metro device pool names.

IBM FlashSystem 840/900 driver

The volume driver for FlashSystem provides OpenStack Block Storage hosts with access to IBM FlashSystems.

This driver is to be used with IBM FlashSystem 840/900 systems only. For any other FlashSystem storage systems (including 5xxx, 7xxx, and 9xxx platforms) see the *IBM Storage Virtualize family volume driver documentation*.

Supported operations

These operations are supported:

- Create, delete, attach, and detach volumes.
- Create, list, and delete volume snapshots.
- Create a volume from a snapshot.
- Copy an image to a volume.
- Copy a volume to an image.
- Clone a volume.
- Extend a volume.
- Get volume statistics.
- Manage and unmanage a volume.

Configure FlashSystem

Configure storage array

The volume driver requires a pre-defined array. You must create an array on the FlashSystem before using the volume driver. An existing array can also be used and existing data will not be deleted.

Note

FlashSystem can only create one array, so no configuration option is needed for the IBM FlashSystem driver to assign it.

Configure user authentication for the driver

The driver requires access to the FlashSystem management interface using SSH. It should be provided with the FlashSystem management IP using the `san_ip` flag, and the management port should be provided by the `san_ssh_port` flag. By default, the port value is configured to be port 22 (SSH).

Note

Make sure the compute node running the `cinder-volume` driver has SSH network access to the storage system.

Using password authentication, assign a password to the user on the FlashSystem. For more detail, see the driver configuration flags for the user and password here: *Enable IBM FlashSystem FC driver* or *Enable IBM FlashSystem iSCSI driver*.

There are some common configuration options for either driver:

Table 33: List of common configuration options for IBM FlashSystem drivers

Flag name	Type	Default	Description
`san_ip`	Required		Management IP or host name
`san_ssh_port`	Optional	22	Management port
`san_login`	Required		Management login user name
`san_password`	Required		Management login password

IBM FlashSystem FC driver

Data Path configuration

Using Fiber Channel (FC), each FlashSystem node should have at least one WWPN port configured. If the `flashsystem_multipath_enabled` flag is set to `True` in the Block Storage service configuration file, the driver uses all available WWPNs to attach the volume to the instance. If the flag is not set, the driver uses the WWPN associated with the volumes preferred node (if available). Otherwise, it uses the first available WWPN of the system. The driver obtains the WWPNs directly from the storage system. You do not need to provide these WWPNs to the driver.

Note

Using FC, ensure that the block storage hosts have FC connectivity to the FlashSystem.

Enable IBM FlashSystem FC driver

Set the volume driver to the FlashSystem driver by setting the `volume_driver` option in the `cinder.conf` configuration file, as follows:

```
volume_driver = cinder.volume.drivers.ibm.flashsystem_fc.FlashSystemFCDriver
```

To enable the IBM FlashSystem FC driver, configure the following options in the `cinder.conf` configuration file:

Table 34: Description of IBM FlashSystem FC configuration options

Configuration option = Default value	Description
`flashsystem_connection_protocol = FC`	(String) Connection protocol should be FC. (Default is FC.)
`flashsystem_multihostmap_enabled = True`	(Boolean) Allows vdisk to multi host mapping. (Default is True)

IBM FlashSystem iSCSI driver

Network configuration

Using iSCSI, each FlashSystem node should have at least one iSCSI port configured. iSCSI IP addresses of IBM FlashSystem can be obtained by FlashSystem GUI or CLI. For more information, see the appropriate IBM Redbook for the FlashSystem.

Note

Using iSCSI, ensure that the compute nodes have iSCSI network access to the IBM FlashSystem.

Enable IBM FlashSystem iSCSI driver

Set the volume driver to the FlashSystem driver by setting the `volume_driver` option in the `cinder.conf` configuration file, as follows:

```
volume_driver = cinder.volume.drivers.ibm.flashsystem_iscsi.
↪FlashSystemISCSIDriver
```

To enable IBM FlashSystem iSCSI driver, configure the following options in the `cinder.conf` configuration file:

Table 35: Description of IBM FlashSystem iSCSI configuration options

Configuration option = Default value	Description
`flashsystem_connection_protocol = FC`	(String) Connection protocol should be FC. (Default is FC.)
`flashsystem_iscsi_portid = 0`	(Integer) Default iSCSI Port ID of FlashSystem. (Default port is 0.)
`flashsystem_multihostmap_enabled = True`	(Boolean) Allows vdisk to multi host mapping. (Default is True)

Note

On the cluster of the FlashSystem, the `iscsi_ip_address` column is the seventh column `IP_address` of the output of `lsportip`.

Note

On the cluster of the FlashSystem, port ID column is the first column `id` of the output of `lsportip`, not the sixth column `port_id`.

Limitations and known issues

IBM FlashSystem only works when:

```
open_access_enabled=off
```

Note

The `flashsystem_multihost_enabled` setting allows the driver to map a vdisk to more than one host at a time. This scenario occurs during migration of a virtual machine with an attached volume; the volume is simultaneously mapped to both the source and destination compute hosts. If your deployment does not require attaching vdisks to multiple hosts, setting this flag to `False` will provide added safety.

IBM Spectrum Scale volume driver

IBM Spectrum Scale is a flexible software-defined storage that can be deployed as high performance file storage or a cost optimized large-scale content repository. IBM Spectrum Scale, previously known as IBM General Parallel File System (GPFS), is designed to scale performance and capacity with no bottlenecks. IBM Spectrum Scale is a cluster file system that provides concurrent access to file systems from multiple nodes. The storage provided by these nodes can be direct attached, network attached, SAN attached, or a combination of these methods. Spectrum Scale provides many features beyond common data access, including data replication, policy based storage management, and space efficient file snapshot and clone operations.

How the Spectrum Scale volume driver works

The Spectrum Scale volume driver, named `gpfs.py`, enables the use of Spectrum Scale in a fashion similar to that of the NFS driver. With the Spectrum Scale driver, instances do not actually access a storage device at the block level. Instead, volume backing files are created in a Spectrum Scale file system and mapped to instances, which emulate a block device.

Note

Spectrum Scale must be installed and cluster has to be created on the storage nodes in the OpenStack environment. A file system must also be created and mounted on these nodes before configuring the cinder service to use Spectrum Scale storage. For more details, please refer to Spectrum Scale product documentation.

Optionally, the Image service can be configured to store glance images in a Spectrum Scale file system. When a Block Storage volume is created from an image, if both image data and volume data reside in the same Spectrum Scale file system, the data from image file is moved efficiently to the volume file using copy-on-write optimization strategy.

Supported operations

- Create, delete, attach, and detach volumes.
- Create, delete volume snapshots.
- Create a volume from a snapshot.
- Create cloned volumes.
- Extend a volume.
- Migrate a volume.
- Retype a volume.
- Create, delete consistency groups.
- Create, delete consistency group snapshots.
- Copy an image to a volume.
- Copy a volume to an image.
- Backup and restore volumes.

Driver configurations

The Spectrum Scale volume driver supports three modes of deployment.

Mode 1 Pervasive Spectrum Scale Client

When Spectrum Scale is running on compute nodes as well as on the cinder node. For example, Spectrum Scale filesystem is available to both Compute and Block Storage services as a local filesystem.

To use Spectrum Scale driver in this deployment mode, set the `volume_driver` in the `cinder.conf` as:

```
volume_driver = cinder.volume.drivers.ibm.gpfs.GPFSDriver
```

The following table contains the configuration options supported by the Spectrum Scale driver in this deployment mode.

Table 36: Description of Spectrum Scale volume driver configuration options

Configuration option = Default value	Description
[DEFAULT]	
gpfs_ima = None	(String) Specifies the path of the Image service repository in GPFS. Leave undefined if not storing images in GPFS.
gpfs_ima = None	(String) Specifies the type of image copy to be used. Set this when the Image service repository also uses GPFS so that image files can be transferred efficiently from the Image service to the Block Storage service. There are two valid values: copy specifies that a full copy of the image is made; copy_on_write specifies that copy-on-write optimization strategy is used and unmodified blocks of the image file are shared efficiently.
gpfs_ma: = 0	(Integer) Specifies an upper limit on the number of indirections required to reach a specific block due to snapshots or clones. A lengthy chain of copy-on-write snapshots or clones can have a negative impact on performance, but improves space utilization. 0 indicates unlimited clone depth.
gpfs_mou = None	(String) Specifies the path of the GPFS directory where Block Storage volume and snapshot files are stored.
gpfs_spa = True	(Boolean) Specifies that volumes are created as sparse files which initially consume no space. If set to False, the volume is created as a fully allocated file, in which case, creation may take a significantly longer time.
gpfs_sto = system	(String) Specifies the storage pool that volumes are assigned to. By default, the system storage pool is used.

Note

The `gpfs_images_share_mode` flag is only valid if the Image Service is configured to use Spectrum Scale with the `gpfs_images_dir` flag. When the value of this flag is `copy_on_write`, the paths specified by the `gpfs_mount_point_base` and `gpfs_images_dir` flags must both reside in the same GPFS file system and in the same GPFS file set.

Mode 2 Remote Spectrum Scale Driver with Local Compute Access

When Spectrum Scale is running on compute nodes, but not on the Block Storage node. For example, Spectrum Scale filesystem is only available to Compute service as Local filesystem where as Block Storage service accesses Spectrum Scale remotely. In this case, `cinder-volume` service running Spectrum Scale driver access storage system over SSH and creates volume backing files to make them available on the compute nodes. This mode is typically deployed when the cinder and glance services are running inside a Linux container. The container host should have Spectrum Scale client running and GPFS filesystem mount path should be bind mounted into the Linux containers.

Note

Note that the user IDs present in the containers should match as that in the host machines. For example, the containers running cinder and glance services should be privileged containers.

To use Spectrum Scale driver in this deployment mode, set the `volume_driver` in the `cinder.conf` as:

```
volume_driver = cinder.volume.drivers.ibm.gpfs.GPFSRemoteDriver
```

The following table contains the configuration options supported by the Spectrum Scale driver in this deployment mode.

Table 37: Description of Spectrum Scale Remote volume driver configuration options

Configuration option = Default value	Description
[DEFAULT]	
gpfs_hosts =	(List) Comma-separated list of IP address or hostnames of GPFS nodes.
gpfs_hosts_ = $state_patl ssh_known_l	(String) File containing SSH host keys for the gpfs nodes with which driver needs to communicate. Default=$state_path/ssh_known_hosts
gpfs_images = None	(String) Specifies the path of the Image service repository in GPFS. Leave undefined if not storing images in GPFS.
gpfs_images = None	(String) Specifies the type of image copy to be used. Set this when the Image service repository also uses GPFS so that image files can be transferred efficiently from the Image service to the Block Storage service. There are two valid values: copy specifies that a full copy of the image is made; copy_on_write specifies that copy-on-write optimization strategy is used and unmodified blocks of the image file are shared efficiently.
gpfs_max_cl = 0	(Integer) Specifies an upper limit on the number of indirections required to reach a specific block due to snapshots or clones. A lengthy chain of copy-on-write snapshots or clones can have a negative impact on performance, but improves space utilization. 0 indicates unlimited clone depth.
gpfs_mount_ = None	(String) Specifies the path of the GPFS directory where Block Storage volume and snapshot files are stored.
gpfs_privat =	(String) Filename of private key to use for SSH authentication.
gpfs_sparse = True	(Boolean) Specifies that volumes are created as sparse files which initially consume no space. If set to False, the volume is created as a fully allocated file, in which case, creation may take a significantly longer time.
gpfs_ssh_pc = 22	(Port number) SSH port to use.
gpfs_storag = system	(String) Specifies the storage pool that volumes are assigned to. By default, the system storage pool is used.
gpfs_strict = False	(Boolean) Option to enable strict gpfs host key checking while connecting to gpfs nodes. Default=False
gpfs_user_l = root	(String) Username for GPFS nodes.
gpfs_user_p =	(String) Password for GPFS node user.

Note

The gpfs_images_share_mode flag is only valid if the Image Service is configured to use Spectrum Scale with the gpfs_images_dir flag. When the value of this flag is copy_on_write, the paths specified by the gpfs_mount_point_base and gpfs_images_dir flags must both reside in the same GPFS file system and in the same GPFS file set.

Mode 3 Remote Spectrum Scale Access

When both Compute and Block Storage nodes are not running Spectrum Scale software and do not have access to Spectrum Scale file system directly as local filesystem. In this case, we create an NFS export on the volume path and make it available on the cinder node and on compute nodes.

Optionally, if one wants to use the copy-on-write optimization to create bootable volumes from glance images, one need to also export the glance images path and mount it on the nodes where glance and cinder services are running. The cinder and glance services will access the GPFS filesystem through NFS.

To use Spectrum Scale driver in this deployment mode, set the `volume_driver` in the `cinder.conf` as:

```
volume_driver = cinder.volume.drivers.ibm.gpfs.GPFSNFSDriver
```

The following table contains the configuration options supported by the Spectrum Scale driver in this deployment mode.

Table 38: Description of Spectrum Scale NFS volume driver configuration options

Configuration option = Default value	Description
[DEFAULT]	
gpfs_image = None	(String) Specifies the path of the Image service repository in GPFS. Leave undefined if not storing images in GPFS.
gpfs_image = None	(String) Specifies the type of image copy to be used. Set this when the Image service repository also uses GPFS so that image files can be transferred efficiently from the Image service to the Block Storage service. There are two valid values: copy specifies that a full copy of the image is made; copy_on_write specifies that copy-on-write optimization strategy is used and unmodified blocks of the image file are shared efficiently.
gpfs_max_c = 0	(Integer) Specifies an upper limit on the number of indirections required to reach a specific block due to snapshots or clones. A lengthy chain of copy-on-write snapshots or clones can have a negative impact on performance, but improves space utilization. 0 indicates unlimited clone depth.
gpfs_mount = None	(String) Specifies the path of the GPFS directory where Block Storage volume and snapshot files are stored.
gpfs_spars = True	(Boolean) Specifies that volumes are created as sparse files which initially consume no space. If set to False, the volume is created as a fully allocated file, in which case, creation may take a significantly longer time.
gpfs_stora = system	(String) Specifies the storage pool that volumes are assigned to. By default, the system storage pool is used.
nas_host =	(String) IP address or Hostname of NAS system.
nas_login = admin	(String) User name to connect to NAS system.
nas_passwo =	(String) Password to connect to NAS system.
nas_privat =	(String) Filename of private key to use for SSH authentication.
nas_ssh_po = 22	(Port number) SSH port to use to connect to NAS system.
nfs_mount_ = $state_pat mnt	(String) Base dir containing mount points for NFS shares.
nfs_shares = /etc/ cinder/ nfs_shares	(String) File with the list of available NFS shares.

Additionally, all the options of the base NFS driver are applicable for GPFSNFSDriver. The above table lists the basic configuration options which are needed for initialization of the driver.

Note

The `gpfs_images_share_mode` flag is only valid if the Image Service is configured to use Spectrum Scale with the `gpfs_images_dir` flag. When the value of this flag is `copy_on_write`, the paths specified by the `gpfs_mount_point_base` and `gpfs_images_dir` flags must both reside in the same GPFS file system and in the same GPFS file set.

Volume creation options

It is possible to specify additional volume configuration options on a per-volume basis by specifying volume metadata. The volume is created using the specified options. Changing the metadata after the volume is created has no effect. The following table lists the volume creation options supported by the GPFS volume driver.

Table 39: **Volume Create Options for Spectrum Scale Volume Drivers**

Metadata Item Name	Description
fstype	Specifies whether to create a file system or a swap area on the new volume. If fstype=swap is specified, the mkswap command is used to create a swap area. Otherwise the mkfs command is passed the specified file system type, for example ext3, ext4 or ntfs.
fslabel	Sets the file system label for the file system specified by fstype option. This value is only used if fstype is specified.
data_pool_name	Specifies the GPFS storage pool to which the volume is to be assigned. Note: The GPFS storage pool must already have been created.
replicas	Specifies how many copies of the volume file to create. Valid values are 1, 2, and, for Spectrum Scale V3.5.0.7 and later, 3. This value cannot be greater than the value of the MaxDataReplicasattribute of the file system.
dio	Enables or disables the Direct I/O caching policy for the volume file. Valid values are yes and no.
write_affinity_depth	Specifies the allocation policy to be used for the volume file. Note: This option only works if allow-write-affinity is set for the GPFS data pool.
block_group_factor	Specifies how many blocks are laid out sequentially in the volume file to behave as a single large block. Note: This option only works if allow-write-affinity is set for the GPFS data pool.
write_affinity_failure_group	Specifies the range of nodes (in GPFS shared nothing architecture) where replicas of blocks in the volume file are to be written. See Spectrum Scale documentation for more details about this option.

This example shows the creation of a 50GB volume with an `ext4` file system labeled `newfs` and direct IO enabled:

```
$ openstack volume create --property fstype=ext4 fslabel=newfs dio=yes \
  --size 50 VOLUME
```

Note that if the metadata for the volume is changed later, the changes do not reflect in the backend. User will have to manually change the volume attributes corresponding to metadata on Spectrum Scale filesystem.

Operational notes for GPFS driver

Volume snapshots are implemented using the GPFS file clone feature. Whenever a new snapshot is created, the snapshot file is efficiently created as a read-only clone parent of the volume, and the volume file uses copy-on-write optimization strategy to minimize data movement.

Similarly when a new volume is created from a snapshot or from an existing volume, the same approach is taken. The same approach is also used when a new volume is created from an Image service image, if the source image is in raw format, and `gpfs_images_share_mode` is set to `copy_on_write`.

The Spectrum Scale driver supports encrypted volume back end feature. To encrypt a volume at rest, specify the extra specification `gpfs_encryption_rest = True`.

IBM Storage Driver for OpenStack

Introduction

The IBM Storage Driver for OpenStack is a software component of the OpenStack cloud environment that enables utilization of storage resources provided by supported IBM storage systems.

The driver was validated on storage systems, as detailed in the Supported storage systems section below.

After the driver is configured on the OpenStack Cinder nodes, storage volumes can be allocated by the Cinder nodes to the Nova nodes. Virtual machines on the Nova nodes can then utilize these storage resources.

Concept diagram

This figure illustrates how an IBM storage system is connected to the OpenStack cloud environment and provides storage resources when the IBM Storage Driver for OpenStack is configured on the OpenStack Cinder nodes. The OpenStack cloud is connected to the IBM storage system over Fibre Channel. Remote cloud users can issue requests for storage resources from the OpenStack cloud. These requests are transparently handled by the IBM Storage Driver, which communicates with the IBM storage system and controls the storage volumes on it. The IBM storage resources are then provided to the Nova nodes in the OpenStack cloud.

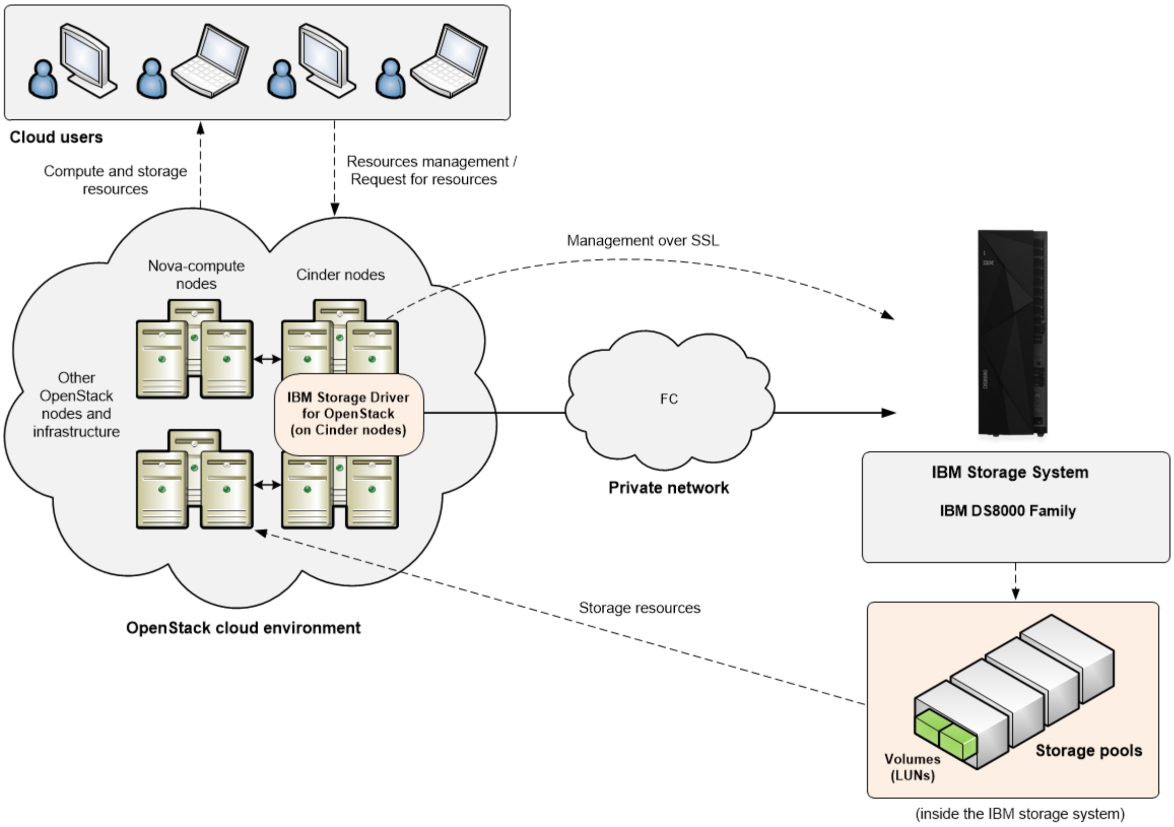
Compatibility and requirements

This section specifies the compatibility and requirements of the IBM Storage Driver for OpenStack.

Supported storage systems

The IBM Storage Driver for OpenStack supports the IBM storage systems, as detailed in the following table.

Storage system	Microcode version	Connectivity
IBM DS8870	7.5 SP4 or later, 7.5 with RESTful API patch	Fibre Channel (FC)
IBM DS8880	8.1 or later	Fibre Channel (FC)



Copy Services license

Copy Services features help you implement storage solutions to keep your business running 24 hours a day, 7 days a week by providing image caching, replication and cloning functions. The Copy Services license is based on usable capacity of the volumes involved in Copy Services functionality.

The Copy Services license is available for the following license scopes: FB and ALL (both FB and CKD).

The Copy Services license includes the following features:

- Global Mirror
- Metro Mirror
- Metro/Global Mirror
- Point-in-Time Copy/FlashCopyő
- z/OSő Global Mirror
- z/OS Metro/Global Mirror Incremental Resync (RMZ)

The Copy Services license feature codes are ordered in increments up to a specific capacity. For example, if you require 160 TB of capacity, order 10 of feature code 8251 (10 TB each up to 100 TB capacity), and 4 of feature code 8252 (15 TB each, for an extra 60 TB).

The Copy Services license includes the following feature codes.

Feature Code	Feature code for licensed function indicator
8250	CS - inactive
8251	CS - 10 TB (up to 100 TB capacity)
8252	CS - 15 TB (from 100.1 TB to 250 TB capacity)
8253	CS - 25 TB (from 250.1 TB to 500 TB capacity)
8254	CS - 75 TB (from 500.1 to 1250 TB capacity)
8255	CS - 175 TB (from 1250.1 TB to 3000 TB capacity)
8256	CS - 300 TB (from 3000.1 TB to 6000 TB capacity)
8260	CS - 500 TB (from 6000.1 TB to 10,000 TB capacity)

The following ordering rules apply when you order the Copy Services license:

- The Copy Services license should be ordered based on the total usable capacity of all volumes involved in one or more Copy Services relationships.
- The licensed authorization must be equal to or less that the total usable capacity allocated to the volumes that participate in Copy Services operations.
- You must purchase features for both the source (primary) and target (secondary) storage system.

Required software on the OpenStack Cinder and Nova nodes

The IBM Storage Driver makes use of the following software on the OpenStack Cinder and Nova-compute nodes.

Software	Installed on
Ubuntu Server (16.04), x64 Red Hat Enterprise Linux (RHEL) 7.x, x64 CentOS Linux 7.x, x64 KVM for IBM z Systems	All OpenStack Cinder nodes
IBM Storage Host Attachment Kit for Linux	All OpenStack Cinder and Nova compute nodes that connect to storage systems and use RHEL 7.x or CentOS Linux 7.x
Linux patch package	All OpenStack Cinder nodes
sysfsutils utility	All OpenStack Cinder nodes on FC network

Configuration

Configure the driver manually by changing the `cinder.conf` file as follows:

```
volume_driver = cinder.volume.drivers.ibm.ibm_storage.IBMStorageDriver
```

Configuration Description for DS8000

Table 40: Description of IBM Storage driver configuration options

Configuration option = Default value	Description
[DEFAULT]	
`ds8k_devadd_u` =	(String) Mapping between IODevice address and unit address.
`ds8k_host_typ` = `auto`	(String) Set to zLinux if your OpenStack version is prior to Liberty and youre connecting to zLinux systems. Otherwise set to auto. Valid values for this parameter are: auto, AMDLinuxRHEL, AMDLinuxSuse, AppleOSX, Fujitsu, Hp, HpTru64, HpVms, LinuxDT, LinuxRF, LinuxRHEL, LinuxSuse, Novell, SGI, SVC, SanFsAIX, SanFsLinux, Sun, VMWare, Win2000, Win2003, Win2008, Win2012, iLinux, nSeries, pLinux, pSeries, pSeriesPowerswap, zLinux, iSeries.
`ds8k_ssid_pre` = `FF`	(String) Set the first two digits of SSID
`proxy` = `cinder.volume.drivers.ibm.ibm_storage.proxy.IBMStoragePrc`	(String) Proxy driver that connects to the IBM Storage Array
`san_clusterna` =	(String) Cluster name to use for creating volumes
`san_ip` =	(String) IP address of SAN controller
`san_login` = `admin`	(String) Username for SAN controller
`san_password` =	(String) Password for SAN controller

Replication parameters

Parameter	Description	Applicable to
replication _device	Volume replication parameters	DS8000
backend_id	IP address or host name of the target storage system	DS8000
san_login	User name to be used during replication procedure	DS8000
san_password	Password to be used during replication procedure (base64-encoded)	DS8000
san_clustername	Pool name on the target storage system	DS8000
port_pairs	ID pairs of IO ports, participating in replication	DS8000
lss_range_for _cg	LSS range to reserve for consistency groups	DS8000

Security

The following information provides an overview of security for the IBM Storage Driver for OpenStack.

Configuring Cinder nodes for trusted communication

The IBM Storage Driver for OpenStack communicates with DS8000 over HTTPS, using self-signed certificate or certificate signed by a certificate authority (CA). Configure a trusted communication link to ensure a successful attachment of a Cinder node to a DS8000 storage system, as detailed in the following sections.

Configuring trusted communication link

Before configuring a DS8000 backend, complete the following steps to establish the chain of trust.

1. In your operating system shell, run this command to obtain the certificate: `openssl x509 -in <(openssl s_client -connect <host fqdn>:8452 -prexit 2>/dev/null) -text -out <host fqdn>.pem`

   If the certificate is self-signed, the following information is displayed:

   ```
   ---
   Certificate chain
   0 s:/CN=ds8000.ibm.com
   i:/CN=ds8000.ibm.com
   ---
   ```

2. Create an exception by moving the certificate `<fqdn>.pem to the /opt/ibm/ds8k_certs/<host>.pem` file.

3. Verify that the <host fqdn> is the same as configured in san_ip.

4. If the certificate subject and issuer are different, the certificate is signed by a CA, as illustrated below:

   ```
   ---
   Certificate chain
   0 s:/C=US/ST=New York/L=Armonk/O=IBM/OU=EI/CN=www.ibm.com
   i:/C=US/O=GeoTrust Inc./CN=GeoTrust SSL CA - G3
   1 s:/C=US/O=GeoTrust Inc./CN=GeoTrust SSL CA - G3
   i:/C=US/O=GeoTrust Inc./CN=GeoTrust Global CA
   ---
   ```

5. Add a public certificate to trusted CA certificate store to complete the chain of trust, as explained below.

6. Verify trusted communication link, as explained below.

Adding a public certificate to trusted CA certificate store

Add the CA public certificate to the trusted CA certificates store on the Cinder node, according to procedures for the operating system in use.

1. For RHEL 7.x or CentOS 7.x, place the certificate to be trusted (in PEM format) into the /etc/pki/ca-trust/source/anchors/ directory. Then, run the `sudo update-ca-trust` command.

2. For Ubuntu 18.04, place the certificate to be trusted (in PEM format) into the /usr/local/share/ca-certificates/ directory. Rename the file, using the `*.crt` extension. Then, run the `sudo update-ca-certificates` command.
3. For Python requests library with certifi, run the `cat ca_public_certificate.pem` command to append the certificate to the location of the certifi trust store file. For example:

```
cat ca_public_certificate.pem >> /usr/local/lib/python3.6/
dist-packages/certifi/cacert.pem.
```

Verifying trusted communication link

Verify the chain of trust has been established successfully.

1. Obtain the location of the Python library requests trust store, according to the installation type.
2. RHEL 7.x or CentOS 7.x:

```
# python3
    Python 3.6.8 (default, Aug  7 2019, 17:28:10)
    [GCC 4.8.5 20150623 (Red Hat 4.8.5-39)] on linux
    Type "help", "copyright", "credits" or "license" for
    more information.
    >>> import requests
    >>> print(requests.certs.where())
    /etc/pki/ca-trust/extracted/openssl/
    ca-bundle.trust.crt
```

3. Ubuntu 18.04:

```
# python3
    Python 3.6.9 (default, Nov  7 2019, 10:44:02)
    [GCC 8.3.0] on linux
    Type "help", "copyright", "credits" or "license" for
    more information.
    >>> import requests
    >>> print(requests.certs.where())
    /etc/ssl/certs/ca-certificates.crt
```

4. Python requests library with certifi:

```
# python3
    Python 3.6.9 (default, Nov  7 2019, 10:44:02)
    [GCC 8.3.0] on linux
    Type "help", "copyright", "credits" or "license" for
    more information.
    >>> import requests
    >>> print(requests.certs.where())
    /usr/local/lib/python3.6/dist-packages/
    certifi/cacert.pem
```

5. Run the `openssl s_client -CAfile <location> -connect <host fqdn>:8452 </dev/null` command. The following return codes indicate a successful or failed attempt in establishing a trusted communication link.

- Verify return code: 0 (ok): success.
- Verify return code: 21 (unable to verify the first certificate), or any other non-zero value: failure.

Troubleshooting

Refer to this information to troubleshoot technical problems that you might encounter when using the IBM Storage Driver for OpenStack.

Checking the Cinder log files

The Cinder log files record operation information that might be useful for troubleshooting.

To achieve optimal and clear logging of events, activate the verbose logging level in the cinder.conf file, located in the `/etc/cinder` folder. Add the following line in the file, save the file, and then restart the cinder-volume service:

```
verbose = True
debug = True
```

To turn off the verbose logging level, change `True` to `False`, save the file, and then restart the cinder-volume service.

Check the log files on a periodic basis to ensure that the IBM Storage Driver is functioning properly. To check the log file on a Cinder node, go to the /var/log/cinder folder and open the activity log file named cinder-volume.log or volume.log.

Best practices

This section contains the general guidance and best practices.

Configuring volume replication (DS8000 Family)

Volume replication is required for disaster recovery and high-availability applications running on top of OpenStack-based clouds. The IBM Storage Driver for OpenStack supports synchronous (Metro Mirror) volume replication for DS8000 storage systems.

1. Verify that:
   - Master and remote storage pools exist on DS8000 systems.
   - Reliable communication link is established between the primary and secondary sites, including physical connection and PPRC path.
   - Metro Mirror replication is enabled on DS8000 storage systems.
2. Perform the following procedure, replacing the values in the example with your own:

```
enabled_backends = ibm_ds8k_1, ibm_ds8k_2
[ibm_ds8k_1]
proxy = cinder.volume.drivers.ds8k_proxy.DS8KProxy
volume_backend_name = ibm_ds8k_1
san_clustername = P2,P3
san_password = actual_password
san_login = actual_username
```

(continues on next page)

(continued from previous page)

```
san_ip = host_fqdn
volume_driver = cinder.volume.drivers.ibm.ibm_storage.IBMStorageDriver
chap = disabled
connection_type = fibre_channel
replication_device = connection_type: fibre_channel,
backend_id: bar, san_ip: host_fqdn,
san_login: actual_username, san_password: actual_password,
san_clustername: P4, port_pairs: I0236-I0306; I0237-I0307

[ibm_ds8k_2]
proxy = cinder.volume.drivers.ibm.ds8k_proxy.DS8KProxy
volume_backend_name = ibm_ds8k_2
san_clustername = P4,P5
san_password = actual_password
san_login = actual_username
san_ip = 10.0.0.1
volume_driver = cinder.volume.drivers.ibm.ibm_storage.IBMStorageDriver
chap = disabled
connection_type = fibre_channel
```

Configuring groups

The IBM Storage Driver for OpenStack supports volume grouping. These groups can be assigned a group type, and used for replication and group snapshotting.

Replication groups

For better control over replication granularity, the user can employ volume grouping. This enables volume group replication and failover without affecting the entire backend. The user can choose between a generic group replication and consistency group (CG) replication. For consistency group replication, the driver utilizes the storage capabilities to handle CGs and replicate them to a remote site. On the other hand, in generic group replication, the driver replicates each volume individually. In addition, the user can select the replication type.

To configure group replication:

1. Create sync replicated consistency-group.

   - Create a volume type for replication.

```
#cinder type-create rep-vol-1
```

   - Create a volume type for replication.

```
#cinder type-key rep-vol-1
set replication_type='<is> sync'
replication_enabled='<is> True'
```

   - Create a group type.

```
#cinder group-type-create rep-gr-1
```

- Configure the group type.

```
#cinder group-type-key rep-gr-1 set group_replication_enabled='<is> True'␣
↪replication_type='<is> sync'
```

- Create a replicated group, using existing group type and volume type.

```
#cinder group-create rep-gr-1 rep-vol-1 --name replicated-gr-1
```

2. Create a volume and add it to the group.

- Create a replicated volume.

```
#cinder create --name vol-1 --volume-type rep-vol-1 1
```

- Add the volume to the group.

```
#cinder group-update --add-volumes 91492ed9-c3cf-4732-a525-60e146510b90␣
↪replicated-gr-1
```

Note

You can also create the volume directly into the group by using the group-id parameter, followed by ID of a group that the new volume belongs to. This function is supported by API version 3.13 and later.

3. Enable replication.

```
#cinder group-enable-replication replicated-gr-1
```

4. Disable replication.

```
#cinder group-disable-replication replicated-gr-1
```

5. Fail over the replicated group.

```
#cinder group-failover-replication replicated-gr-1
```

Consistency groups

Consistency groups are mostly the same as replication groups, but with additional support of group snapshots (`consistent_group_snapshot_enabled` parameter). See configuration example below.

```
#cinder group-type-create cg1
#cinder group-type-show cg1
#cinder group-type-key cg1 set consistent_group_snapshot_enabled="<is> True"
#cinder group-create --name cg1 IBM-DS8K_ibm.com_P0_P1_fibre_channel_not_thin,
IBM-DS8K_ibm.com_P0_P1_fibre_channel_thin,
IBM-DS8K_ibm.com_P0_P1_fibre_channel_not_thin_replica,
IBM-DS8K_ibm.com_P0_P1_fibre_channel_thin_replica
```

Using volume types for volume allocation control (DS8000 Family)

For better controls over volume placement granularity, you can use volume types. This enables volumes to be created on specific LSSes or pools. You can combine both types.

- Storage pool

```
#cinder type-key pool-1_2 set drivers:storage_pool_ids='P1,P2'
```

- LSS

```
#cinder type-key lss80_81 set drivers:storage_lss_ids='80,81'
```

IBM Storage Virtualize family volume driver

The volume management driver for Storage Virtualize family offers various block storage services. It provides OpenStack Compute instances with access to IBM Storage Virtualize family storage products. These products include the SAN Volume Controller, Storwize and FlashSystem family members built with IBM Storage Virtualize (including FlashSystem 5xxx, 7xxx, 9xxx).

For specific product publications, see IBM Documentation.

Note

IBM Storage Virtualize family is formerly known as IBM Storwize. As a result, the product code contains Storwize terminology and prefixes.

Supported operations

The IBM Storage Virtualize family volume driver supports the following block storage service volume operations:

- Create, list, delete, attach (map), and detach (unmap) volumes.
- Create, list, and delete volume snapshots.
- Copy an image to a volume.
- Copy a volume to an image.
- Clone a volume.
- Extend a volume.
- Retype a volume.
- Create a volume from a snapshot.
- Create, list, and delete consistency group.
- Create, list, and delete consistency group snapshot.
- Modify consistency group (add or remove volumes).
- Create consistency group from source (source can be a CG or CG snapshot)
- Manage an existing volume.
- Failover-host for replicated back ends.

- Failback-host for replicated back ends.
- Create, list, and delete replication group.
- Enable, disable replication group.
- Failover, failback replication group.

Configure the Storage Virtualize family system

Network configuration

The Storage Virtualize family system must be configured for iSCSI, Fibre Channel, or both.

If using iSCSI, each Storage Virtualize family node should have at least one iSCSI IP address. The Storage Virtualize family driver uses an iSCSI IP address associated with the volumes preferred node (if available) to attach the volume to the instance, otherwise it uses the first available iSCSI IP address of the system. The driver obtains the iSCSI IP address directly from the storage system. You do not need to provide these iSCSI IP addresses directly to the driver.

Note

If using iSCSI, ensure that the compute nodes have iSCSI network access to the Storage Virtualize family system.

If using Fibre Channel (FC), each Storage Virtualize family node should have at least one WWPN port configured. The driver uses all available WWPNs to attach the volume to the instance. The driver obtains the WWPNs directly from the storage system. You do not need to provide these WWPNs directly to the driver.

Note

If using FC, ensure that the compute nodes have FC connectivity to the Storage Virtualize family system.

iSCSI CHAP authentication

If using iSCSI for data access and the `storwize_svc_iscsi_chap_enabled` is set to `True`, the driver will associate randomly-generated CHAP secrets with all hosts on the Storage Virtualize family. The compute nodes use these secrets when creating iSCSI connections.

Warning

CHAP secrets are added to existing hosts as well as newly-created ones. If the CHAP option is enabled, hosts will not be able to access the storage without the generated secrets.

Note

Not all OpenStack Compute drivers support CHAP authentication. Please check compatibility before using.

> **Note**
>
> CHAP secrets are passed from OpenStack Block Storage to Compute in clear text. This communication should be secured to ensure that CHAP secrets are not discovered.

Configure storage pools

The IBM Storage Virtualize family driver can allocate volumes in multiple pools. The pools should be created in advance and be provided to the driver using the `storwize_svc_volpool_name` configuration flag in the form of a comma-separated list. For the complete list of configuration flags, see *Storage Virtualize family driver options in cinder.conf*.

Configure user authentication for the driver

The driver requires access to the Storage Virtualize family system management interface. The driver communicates with the management using SSH. The driver should be provided with the Storage Virtualize family management IP using the `san_ip` flag, and the management port should be provided by the `san_ssh_port` flag. By default, the port value is configured to be port 22 (SSH). Also, you can set the secondary management IP using the `storwize_san_secondary_ip` flag.

> **Note**
>
> Make sure the compute node running the cinder-volume management driver has SSH network access to the storage system.

To allow the driver to communicate with the Storage Virtualize family system, you must provide the driver with a user on the storage system. The driver has two authentication methods: password-based authentication and SSH key pair authentication. The user should have an Administrator role. It is suggested to create a new user for the management driver. Please consult with your storage and security administrator regarding the preferred authentication method and how passwords or SSH keys should be stored in a secure manner.

> **Note**
>
> When creating a new user on the Storage Virtualize family system, make sure the user belongs to the Administrator group or to another group that has an Administrator role.

If using password authentication, assign a password to the user on the Storage Virtualize family system. The driver configuration flags for the user and password are `san_login` and `san_password`, respectively.

If you are using the SSH key pair authentication, create SSH private and public keys using the instructions below or by any other method. Associate the public key with the user by uploading the public key: select the *choose file* option in the Storage Virtualize family management GUI under *SSH public key*. Alternatively, you may associate the SSH public key using the command-line interface; details can be found in the Storage Virtualize family documentation. The private key should be provided to the driver using the `san_private_key` configuration flag.

Create a SSH key pair with OpenSSH

You can create an SSH key pair using OpenSSH, by running:

```
$ ssh-keygen -t rsa
```

The command prompts for a file to save the key pair. For example, if you select `key` as the filename, two files are created: `key` and `key.pub`. The `key` file holds the private SSH key and `key.pub` holds the public SSH key.

The command also prompts for a pass phrase, which should be empty.

The private key file should be provided to the driver using the `san_private_key` configuration flag. The public key should be uploaded to the Storage Virtualize family system using the storage management GUI or command-line interface.

> **Note**
>
> Ensure that Cinder has read permissions on the private key file.

Configure the Storage Virtualize family driver

Enable the Storage Virtualize family driver

Set the volume driver to the Storage Virtualize family driver by setting the `volume_driver` option in the `cinder.conf` file as follows:

iSCSI:

```
[svc1234]
volume_driver = cinder.volume.drivers.ibm.storwize_svc.storwize_svc_iscsi.
↪StorwizeSVCISCSIDriver
san_ip = 1.2.3.4
san_login = superuser
san_password = passw0rd
storwize_svc_volpool_name = cinder_pool1
volume_backend_name = svc1234
```

FC:

```
[svc1234]
volume_driver = cinder.volume.drivers.ibm.storwize_svc.storwize_svc_fc.
↪StorwizeSVCFCDriver
san_ip = 1.2.3.4
san_login = superuser
san_password = passw0rd
storwize_svc_volpool_name = cinder_pool1
volume_backend_name = svc1234
```

Replication configuration

Add the following to the back-end specification to specify another storage to replicate to:

```
replication_device = backend_id:rep_svc,
                     san_ip:1.2.3.5,
                     san_login:superuser,
                     san_password:passw0rd,
                     pool_name:cinder_pool1
```

The backend_id is a unique name of the remote storage, the san_ip, san_login, and san_password is authentication information for the remote storage. The pool_name is the pool name for the replication target volume.

Note

Only one replication_device can be configured for one back end storage since only one replication target is supported now.

Storage Virtualize family driver options in cinder.conf

The following options specify default values for all volumes. Some can be over-ridden using volume types, which are described below.

Note

IBM Storage Virtualize family is formerly known as IBM Storwize. As a result, the product code contains Storwize terminology and prefixes.

Configuration option = Default value	Description
[DEFAULT]	
san_ip =	(String) IP address of SAN controller.
san_login = admin	(String) Username for SAN controller.
san_password =	(String) Password for SAN controller.
san_private_key =	(String) Filename of private key to use for SSH authentica
san_ssh_port = 22	(Port number) SSH port to use with SAN.
ssh_conn_timeout = 30	(Integer) SSH connection timeout in seconds.
ssh_min_pool_conn = 1	(Integer) Minimum SSH connections in the pool.
ssh_max_pool_conn = 5	(Integer) Maximum SSH connections in the pool.
storwize_san_secondary_ip = None	(String) Specifies secondary management IP or hostname
storwize_svc_allow_tenant_qos = False	(Boolean) Allow tenants to specify QoS on create.
storwize_svc_flashcopy_rate = 50	(Integer) Specifies the Storage Virtualize family FlashCop
storwize_svc_clean_rate = 50	(Integer) Specifies the Storwize cleaning rate for the mapp
storwize_svc_flashcopy_timeout = 120	(Integer) Maximum number of seconds to wait for FlashC
storwize_svc_iscsi_chap_enabled = True	(Boolean) Configure CHAP authentication for iSCSI conn
storwize_svc_multihostmap_enabled = True	(Boolean) DEPRECATED: This option no longer has any

Configuration option = Default value	Description
storwize_svc_multipath_enabled = False	(Boolean) Connect with multipath (FC only; iSCSI multip
storwize_svc_stretched_cluster_partner = None	(String) If operating in stretched cluster mode, specify the
storwize_svc_vol_autoexpand = True	(Boolean) Storage system autoexpand parameter for volum
storwize_svc_vol_compression = False	(Boolean) Storage system compression option for volumes
storwize_svc_vol_easytier = True	(Boolean) Enable Easy Tier for volumes.
storwize_svc_vol_grainsize = 256	(Integer) Storage system grain size parameter for volumes
storwize_svc_vol_iogrp = 0	(Integer) The I/O group in which to allocate volumes
storwize_svc_vol_nofmtdisk = False	(Boolean) Specifies that the volume not be formatted durin
storwize_svc_vol_rsize = 2	(Integer) Storage system space-efficiency parameter for vo
storwize_svc_vol_warning = 0	(Integer) Storage system threshold for volume capacity wa
storwize_svc_volpool_name = volpool	(List) Comma separated list of storage system storage poo
storwize_svc_mirror_pool = None	(String) Specifies the name of the pool in which mirrored
storwize_svc_retain_aux_volume = False	(Boolean) Defines an optional parameter to retain an auxil
storwize_peer_pool = None	(String) Specifies the name of the peer pool for a HyperSw
storwize_preferred_host_site = {}	(Dictionary) Specifies the site information for host. One W
cycle_period_seconds = 300	(Integer) Defines an optional cycle period that applies to C
storwize_portset = None	(String) Specifies the name of the portset in which the hos

Note the following:

- The authentication requires either a password (`san_password`) or SSH private key (`san_private_key`). One must be specified. If both are specified, the driver uses only the SSH private key.
- The driver creates thin-provisioned volumes by default. The `storwize_svc_vol_rsize` flag defines the initial physical allocation percentage for thin-provisioned volumes, or if set to `-1`, the driver creates full allocated volumes. More details about the available options are available in the Storage Virtualize family documentation.

Placement with volume types

The IBM Storage Virtualize family exposes capabilities that can be added to the `extra specs` of volume types, and used by the filter scheduler to determine placement of new volumes. Make sure to prefix these keys with `capabilities:` to indicate that the scheduler should use them. The following `extra specs` are supported:

- `capabilities:volume_backend_name` - Specify a specific back-end where the volume should be created. The back-end name is a concatenation of the name of the Storage Virtualize family storage system as shown in `lssystem`, an underscore, and the name of the pool (mdisk group). For example:

  ```
  capabilities:volume_backend_name=myV7000_openstackpool
  ```

- `capabilities:compression_support` - Specify a back-end according to compression support. A value of `True` should be used to request a back-end that supports compression, and a value of `False` will request a back-end that does not support compression. If you do not have constraints on compression support, do not set this key. Note that specifying `True` does not enable compression; it only requests that the volume be placed on a back-end that supports compression. Example syntax:

```
capabilities:compression_support='<is> True'
```

Note

Currently, the compression_enabled() API that indicates compression_license support is not fully functional. It does not work on all storage types. Additional functionalities will be added in a later release.

- `capabilities:easytier_support` - Similar semantics as the `compression_support` key, but for specifying according to support of the Easy Tier feature. Example syntax:

  ```
  capabilities:easytier_support='<is> True'
  ```

- `capabilities:pool_name` - Specify a specific pool to create volume if only multiple pools are configured. pool_name should be one value configured in storwize_svc_volpool_name flag. Example syntax:

  ```
  capabilities:pool_name=cinder_pool2
  ```

Configure per-volume creation options

Volume types can also be used to pass options to the IBM Storage Virtualize family driver, which over-ride the default values set in the configuration file. Contrary to the previous examples where the `capabilities` scope was used to pass parameters to the Cinder scheduler, options can be passed to the Storage Virtualize family driver with the `drivers` scope.

The following `extra specs` keys are supported by the Storage Virtualize family driver:

- rsize
- warning
- autoexpand
- grainsize
- compression
- easytier
- multipath
- iogrp
- mirror_pool
- volume_topology
- peer_pool
- flashcopy_rate
- clean_rate
- cycle_period_seconds

These keys have the same semantics as their counterparts in the configuration file. They are set similarly; for example, `rsize=2` or `compression=False`.

Example: Volume types

In the following example, we create a volume type to specify a controller that supports compression, and enable compression:

```
$ openstack volume type create compressed
$ openstack volume type set --property capabilities:compression_support='<is>␣
↪True' --property drivers:compression=True compressed
```

We can then create a 50GB volume using this type:

```
$ openstack volume create "compressed volume" --type compressed --size 50
```

In the following example, create a volume type that enables synchronous replication (metro mirror):

```
$ openstack volume type create ReplicationType
$ openstack volume type set --property replication_type="<in> metro" \
  --property replication_enabled='<is> True' --property volume_backend_
↪name=svc234 ReplicationType
```

In the following example, we create a volume type to support stretch cluster volume or mirror volume:

```
$ openstack volume type create mirror_vol_type
$ openstack volume type set --property volume_backend_name=svc1 \
  --property drivers:mirror_pool=pool2 mirror_vol_type
```

Volume types can be used, for example, to provide users with different

- performance levels (such as, allocating entirely on an HDD tier, using Easy Tier for an HDD-SDD mix, or allocating entirely on an SSD tier)
- resiliency levels (such as, allocating volumes in pools with different RAID levels)
- features (such as, enabling/disabling Real-time Compression, replication volume creation)

QOS

The Storage Virtualize family driver provides QOS support for storage volumes by controlling the I/O amount. QOS is enabled by editing the `etc/cinder/cinder.conf` file and setting the `storwize_svc_allow_tenant_qos` to `True`.

There are three ways to set the Storage Virtualize family `IOThrotting` parameter for storage volumes:

- Add the `qos:IOThrottling` key into a QOS specification and associate it with a volume type.
- Add the `qos:IOThrottling` key into an extra specification with a volume type.
- Add the `qos:IOThrottling` key to the storage volume metadata.

Note

If you are changing a volume type with QOS to a new volume type without QOS, the QOS configuration settings will be removed.

Operational notes for the Storage Virtualize family driver

Migrate volumes

In the context of OpenStack block storages volume migration feature, the IBM Storage Virtualize family driver enables the storages virtualization technology. When migrating a volume from one pool to another, the volume will appear in the destination pool almost immediately, while the storage moves the data in the background.

Note

To enable this feature, both pools involved in a given volume migration must have the same values for `extent_size`. If the pools have different values for `extent_size`, the data will still be moved directly between the pools (not host-side copy), but the operation will be synchronous.

Extend volumes

The IBM Storage Virtualize family driver allows for extending a volumes size, but only for volumes without snapshots.

Snapshots and clones

Snapshots are implemented using FlashCopy with no background copy (space-efficient). Volume clones (volumes created from existing volumes) are implemented with FlashCopy, but with background copy enabled. This means that volume clones are independent, full copies. While this background copy is taking place, attempting to delete or extend the source volume will result in that operation waiting for the copy to complete.

Volume retype

The IBM Storage Virtualize family driver enables you to modify volume types. When you modify volume types, you can also change these extra specs properties:

- rsize
- warning
- autoexpand
- grainsize
- compression
- easytier
- iogrp
- nofmtdisk
- mirror_pool
- volume_topology
- peer_pool
- flashcopy_rate
- cycle_period_seconds

Note

When you change the `rsize`, `grainsize` or `compression` properties, volume copies are asynchronously synchronized on the array.

Note

To change the `iogrp` property, IBM Storage Virtualize family firmware version 6.4.0 or later is required.

Replication operation

Configure replication in volume type

A volume is only replicated if the volume is created with a volume-type that has the extra spec `replication_enabled` set to `<is> True`. Three types of replication are supported now, global mirror(async), global mirror with change volume(async) and metro mirror(sync). It can be specified by a volume-type that has the extra spec `replication_type` set to `<in> global`, `<in> gmcv` or `<in> metro`. If no `replication_type` is specified, global mirror will be created for replication.

If `replication_type` set to `<in> gmcv`, cycle_period_seconds can be set as the cycling time perform global mirror relationship with multi cycling mode. Default value is 300. Example syntax:

```
$ cinder type-create gmcv_type
$ cinder type-key gmcv_type set replication_enabled='<is> True' \
  replication_type="<in> gmcv" drivers:cycle_period_seconds=500
```

Note

It is better to establish the partnership relationship between the replication source storage and the replication target storage manually on the storage back end before replication volume creation.

Failover host

The `failover-host` command is designed for the case where the primary storage is down.

```
$ cinder failover-host cinder@svciscsi --backend_id target_svc_id
```

If a failover command has been executed and the primary storage has been restored, it is possible to do a failback by simply specifying default as the `backend_id`:

```
$ cinder failover-host cinder@svciscsi --backend_id default
```

Note

Before you perform a failback operation, synchronize the data from the replication target volume to the primary one on the storage back end manually, and do the failback only after the synchronization is done since the synchronization may take a long time. If the synchronization is not done manually, Storage Virtualize family block storage service driver will perform the synchronization and do the failback after the synchronization is finished.

Replication group

Before creating replication group, a group-spec which key `consistent_group_replication_enabled` set to <is> `True` should be set in group type. Volume type used to create group must be replication enabled, and its `replication_type` should be set either <in> `global` or <in> `metro`. The failover_group api allows group to be failed over and back without failing over the entire host. Example syntax:

- Create replication group

```
$ cinder group-type-create rep-group-type-example
$ cinder group-type-key rep-group-type-example set consistent_group_replication_enabled='<is> True'
$ cinder type-create type-global
$ cinder type-key type-global set replication_enabled='<is> True' replication_type='<in> global'
$ cinder group-create rep-group-type-example type-global --name global-group
```

- Failover replication group

```
$ cinder group-failover-replication --secondary-backend-id target_svc_id group_id
```

- Failback replication group

```
$ cinder group-failover-replication --secondary-backend-id default group_id
```

Note

Optionally, allow-attached-volume can be used to failover the in-use volume, but fail over/back an in-use volume is not recommended. If the user does failover operation to an in-use volume, the volume status remains in-use after failover. But the in-use replication volume would change to read-only since the primary volume is changed to auxiliary side and the instance is still attached to the master volume. As a result please detach the replication volume first and attach again if user want to reuse the in-use replication volume as read-write.

HyperSwap Volumes

A HyperSwap volume is created with a volume-type that has the extra spec `drivers:volume_topology` set to `hyperswap`. To support HyperSwap volumes, IBM Storage Virtualize family firmware version 7.6.0 or later is required. Add the following to the back-end configuration to specify the host preferred site for HyperSwap volume. FC:

```
storwize_preferred_host_site = site1:20000090fa17311e&ff00000000000001,
                               site2:20000089762sedce&ff00000000000000
```

iSCSI:

```
storwize_preferred_host_site = site1:iqn.1993-08.org.debian:01:eac5ccc1aaa&
↪iqn.1993-08.org.debian:01:be53b7e236be,
                               site2:iqn.1993-08.org.debian:01:eac5ccc1bbb&
↪iqn.1993-08.org.debian:01:abcdefg9876w
```

The site1 and site2 are names of the two host sites used in Storage Virtualize family storage systems. The WWPNs and IQNs are the connectors used for host mapping in the Storage Virtualize family.

```
$ cinder type-create hyper_type
$ cinder type-key hyper_type set drivers:volume_topology=hyperswap \
  drivers:peer_pool=Pool_site2
```

Note

The property `rsize` is considered as `buffersize` for the HyperSwap volume. The HyperSwap property `iogrp` is selected by storage.

A group is created as a HyperSwap group with a group-type that has the group spec `hyperswap_group_enabled` set to `<is> True`.

INFINIDAT InfiniBox Block Storage driver

The INFINIDAT Block Storage volume driver provides iSCSI and Fibre Channel support for INFINIDAT InfiniBox storage systems.

This section explains how to configure the INFINIDAT driver.

Supported operations

- Create, delete, attach, and detach volumes.
- Create, list, and delete volume snapshots.
- Create a volume from a snapshot.
- Copy a volume to an image.
- Copy an image to a volume.
- Clone a volume.
- Extend a volume.
- Get volume statistics.
- Create, modify, delete, and list consistency groups.
- Create, modify, delete, and list snapshots of consistency groups.
- Create consistency group from consistency group or consistency group snapshot.
- Revert a volume to a snapshot.
- Manage and unmanage volumes and snapshots.
- List manageable volumes and snapshots.

- Attach a volume to multiple instances at once (multi-attach).
- Host and storage assisted volume migration.
- Efficient non-disruptive volume backup.

External package installation

The driver requires the `infinisdk` package for communicating with InfiniBox systems. Install the package from PyPI using the following command:

```
$ pip install infinisdk
```

Setting up the storage array

Create a storage pool object on the InfiniBox array in advance. The storage pool will contain volumes managed by OpenStack. Mixing OpenStack APIs and non-OpenStack methods are not supported when used to attach the same hosts via the same protocol. For example, it is not possible to create boot-from-SAN volumes and OpenStack volumes for the same host with Fibre Channel. Instead, use a different protocol for one of the volumes. Refer to the InfiniBox manuals for details on pool management.

Driver configuration

Edit the `cinder.conf` file, which is usually located under the following path `/etc/cinder/cinder.conf`.

- Add a section for the INFINIDAT driver back end.
- Under the `[DEFAULT]` section, set the `enabled_backends` parameter with the name of the new back-end section.

Configure the driver back-end section with the parameters below.

- Configure the driver name by setting the following parameter:

  ```
  volume_driver = cinder.volume.drivers.infinidat.InfiniboxVolumeDriver
  ```

- Configure the management IP of the InfiniBox array by adding the following parameter:

  ```
  san_ip = InfiniBox management IP
  ```

- Verify that the InfiniBox array can be managed via an HTTPS connection. And the `driver_use_ssl` parameter should be set to `true` to enable use of the HTTPS protocol. HTTP can also be used if `driver_use_ssl` is set to (or defaults to) `false`. To suppress requests library SSL certificate warnings, set the `suppress_requests_ssl_warnings` parameter to `true`.

  ```
  driver_use_ssl = true/false
  suppress_requests_ssl_warnings = true/false
  ```

  These parameters defaults to `false`.

- Configure user credentials.

  The driver requires an InfiniBox user with administrative privileges. We recommend creating a dedicated OpenStack user account that holds a pool admin user role. Refer to the InfiniBox manuals for details on user account management. Configure the user credentials by adding the following parameters:

```
san_login = infinibox_username
san_password = infinibox_password
```

- Configure the name of the InfiniBox pool by adding the following parameter:

```
infinidat_pool_name = Pool defined in InfiniBox
```

- The back-end name is an identifier for the back end. We recommend using the same name as the name of the section. Configure the back-end name by adding the following parameter:

```
volume_backend_name = back-end name
```

- Thin provisioning.

  The INFINIDAT driver supports creating thin or thick provisioned volumes. Configure thin or thick provisioning by adding the following parameter:

```
san_thin_provision = true/false
```

  This parameter defaults to `true`.

- Configure the connectivity protocol.

  The InfiniBox driver supports connection to the InfiniBox system in both the fibre channel and iSCSI protocols. Configure the desired protocol by adding the following parameter:

```
infinidat_storage_protocol = iscsi/fc
```

  This parameter defaults to `fc`.

- Configure iSCSI netspaces.

  When using the iSCSI protocol to connect to InfiniBox systems, you must configure one or more iSCSI network spaces in the InfiniBox storage array. Refer to the InfiniBox manuals for details on network space management. Configure the names of the iSCSI network spaces to connect to by adding the following parameter:

```
infinidat_iscsi_netspaces = iscsi_netspace
```

  Multiple network spaces can be specified by a comma separated string.

  This parameter is ignored when using the FC protocol.

- Configure CHAP

  InfiniBox supports CHAP authentication when using the iSCSI protocol. To enable CHAP authentication, add the following parameter:

```
use_chap_auth = true
```

  To manually define the username and password, add the following parameters:

```
chap_username = username
chap_password = password
```

If the CHAP username or password are not defined, they will be auto-generated by the driver.

The CHAP parameters are ignored when using the FC protocol.

- Volume compression

  Volume compression is available for all supported InfiniBox versions. By default, compression for all newly created volumes is inherited from its parent pool at creation time. All pools are created by default with compression enabled.

  To explicitly enable or disable compression for all newly created volumes, add the following configuration parameter:

  ```
  infinidat_use_compression = true/false
  ```

  Or leave this configuration parameter unset (commented out) for all created volumes to inherit their compression setting from their parent pool at creation time. The default value is unset.

After modifying the `cinder.conf` file, restart the `cinder-volume` service.

Create a new volume type for each distinct `volume_backend_name` value that you added in the `cinder.conf` file. The example below assumes that the same `volume_backend_name=infinidat-pool-a` option was specified in all of the entries, and specifies that the volume type `infinidat` can be used to allocate volumes from any of them. Example of creating a volume type:

```
$ openstack volume type create infinidat
$ openstack volume type set --property volume_backend_name=infinidat-
↪pool-a infinidat
```

Configuration example

```
[DEFAULT]
enabled_backends = infinidat-pool-a

[infinidat-pool-a]
volume_driver = cinder.volume.drivers.infinidat.InfiniboxVolumeDriver
volume_backend_name = infinidat-pool-a
driver_use_ssl = true
suppress_requests_ssl_warnings = true
san_ip = 10.1.2.3
san_login = openstackuser
san_password = openstackpass
san_thin_provision = true
infinidat_pool_name = pool-a
infinidat_storage_protocol = iscsi
infinidat_iscsi_netspaces = default_iscsi_space
```

Driver-specific options

The following table contains the configuration options that are specific to the INFINIDAT driver.

Table 42: Description of INFINIDAT InfiniBox configuration options

Configuration option = Default value	Description
`infinidat_i` = []	(List of String) List of names of network spaces to use for iSCSI connectivity
`infinidat_p` = None	(String) Name of the pool from which volumes are allocated
`infinidat_s` = fc	(String(choices=[iscsi, fc])) Protocol for transferring data between host and storage back-end.
`infinidat_u` = None	(Boolean) Specifies whether to enable (true) or disable (false) compression for all newly created volumes. Leave this unset (commented out) for all created volumes to inherit their compression setting from their parent pool at creation time. The default value is unset.

Infortrend volume driver

The Infortrend volume driver is a Block Storage driver providing iSCSI and Fibre Channel support for Infortrend storages.

Supported operations

The Infortrend volume driver supports the following volume operations:

- Create, delete, attach, and detach volumes.
- Create and delete a snapshot.
- Create a volume from a snapshot.
- Copy an image to a volume.
- Copy a volume to an image.
- Clone a volume.
- Extend a volume
- Retype a volume.
- Manage and unmanage a volume.
- Migrate a volume with back-end assistance.
- Live migrate an instance with volumes hosted on an Infortrend backend.

System requirements

To use the Infortrend volume driver, the following settings are required:

Set up Infortrend storage

- Create logical volumes in advance.
- Host side setting `Peripheral device type` should be `No Device Present (Type=0x7f)`.

Set up cinder-volume node

- Install JRE 7 or later.
- Download the Infortrend storage CLI from the release page. Choose the raidcmd_ESDS10.jar file, whichs under v2.1.3 on the github releases page, and assign it to the default path `/opt/bin/Infortrend/`.

Driver configuration

On `cinder-volume` nodes, set the following in your `/etc/cinder/cinder.conf`, and use the following options to configure it:

Driver options

Table 43: Description of Infortrend volume driver configuration options

Configuration option = Default value	Description
[DEFAULT]	
`infortrend_cli_max_r` = `5`	(Integer) The maximum retry times if a command fails.
`infortrend_cli_path` = `/opt/bin/Infortrend/raidcmd_ESDS10.jar`	(String) The Infortrend CLI absolute path.
`infortrend_cli_timec` = `60`	(Integer) The timeout for CLI in seconds.
`infortrend_cli_cache` = `False`	(Boolean) The Infortrend CLI cache. Make sure the array is only managed by Openstack, and it is only used by one cinder-volume node. Otherwise, never enable it! The data might be asynchronous if there were any other operations.
`infortrend_pools_nam` = `None`	(String) The Infortrend logical volumes name list. It is separated with comma.
`infortrend_iqn_prefi` = `iqn.2002-10.com.infortrend`	(String) Infortrend iqn prefix for iSCSI.
`infortrend_slots_a_c` = `None`	(String) Infortrend raid channel ID list on Slot A for OpenStack usage. It is separated with comma.
`infortrend_slots_b_c` = `None`	(String) Infortrend raid channel ID list on Slot A for OpenStack usage. It is separated with comma.
`java_path` = `/usr/bin/java`	(String) The Java absolute path.

iSCSI configuration example

```
[DEFAULT]
default_volume_type = IFT-ISCSI
enabled_backends = IFT-ISCSI

[IFT-ISCSI]
volume_driver = cinder.volume.drivers.infortrend.infortrend_iscsi_cli.
↪InfortrendCLIISCSIDriver
volume_backend_name = IFT-ISCSI
infortrend_pools_name = POOL-1,POOL-2
san_ip = MANAGEMENT_PORT_IP
san_password = MANAGEMENT_PASSWORD
infortrend_slots_a_channels_id = 0,1,2,3
infortrend_slots_b_channels_id = 0,1,2,3
```

Fibre Channel configuration example

```
[DEFAULT]
default_volume_type = IFT-FC
enabled_backends = IFT-FC

[IFT-FC]
volume_driver = cinder.volume.drivers.infortrend.infortrend_fc_cli.
↪InfortrendCLIFCDriver
volume_backend_name = IFT-FC
infortrend_pools_name = POOL-1,POOL-2,POOL-3
san_ip = MANAGEMENT_PORT_IP
san_password = MANAGEMENT_PASSWORD
infortrend_slots_a_channels_id = 4,5
```

Multipath configuration

- Enable multipath for image transfer in `/etc/cinder/cinder.conf` for each back end or in `[backend_defaults]` section as a common configuration for all backends.

  ```
  use_multipath_for_image_xfer = True
  ```

  Restart the `cinder-volume` service.

- Enable multipath for volume attach and detach in `/etc/nova/nova.conf`.

  ```
  [libvirt]
  ...
  volume_use_multipath = True
  ...
  ```

  Restart the `nova-compute` service.

Extra spec usage

- `infortrend:provisioning` - Defaults to `full` provisioning, the valid values are thin and full.
- `infortrend:tiering` - Defaults to use `all` tiering, the valid values are subsets of 0, 1, 2, 3.

  If multi-pools are configured in `cinder.conf`, it can be specified for each pool, separated by semicolon.

  For example:

  `infortrend:provisioning`: `POOL-1:thin; POOL-2:full`

  `infortrend:tiering`: `POOL-1:all; POOL-2:0; POOL-3:0,1,3`

For more details, see Infortrend documents.

Inspur AS13000 series volume driver

Inspur AS13000 series volume driver provides OpenStack Compute instances with access to Inspur AS13000 series storage system.

Inspur AS13000 storage can be used with iSCSI connection.

This documentation explains how to configure and connect the block storage nodes to Inspur AS13000 series storage.

Driver options

The following table contains the configuration options supported by the Inspur AS13000 iSCSI driver.

Table 44: Description of Inspur AS13000 configuration options

Configuration option = Default value	Description
`as13000_ipsan_]` = `[Pool0]`	(List of String) The Storage Pools Cinder should use, a comma separated list.
`as13000_meta_p` = `None`	(String) The pool which is used as a meta pool when creating a volume, and it should be a replication pool at present. If not set, the driver will choose a replication pool from the value of as13000_ipsan_pools.
`as13000_token_` = `3300`	(Integer(min=600, max=3600)) The effective time of token validity in seconds.

Supported operations

- Create, list, delete, attach (map), and detach (unmap) volumes.
- Create, list and delete volume snapshots.
- Create a volume from a snapshot.
- Copy an image to a volume.
- Copy a volume to an image.
- Clone a volume.

- Extend a volume.

Configure Inspur AS13000 iSCSI backend

This section details the steps required to configure the Inspur AS13000 storage cinder driver.

1. In the `cinder.conf` configuration file under the `[DEFAULT]` section, set the enabled_backends parameter.

```
[DEFAULT]
enabled_backends = AS13000-1
```

2. Add a backend group section for backend group specified in the enabled_backends parameter.
3. In the newly created backend group section, set the following configuration options:

```
[AS13000-1]
# The driver path
volume_driver = cinder.volume.drivers.inspur.as13000.as13000_driver.
↪AS13000Driver
# Management IP of Inspur AS13000 storage array
san_ip = 10.0.0.10
# The Rest API port
san_api_port = 8088
# Management username of Inspur AS13000 storage array
san_login = root
# Management password of Inspur AS13000 storage array
san_password = passw0rd
# The Pool used to allocated volumes
as13000_ipsan_pools = Pool0
# The Meta Pool to use, should be a replication Pool
as13000_meta_pool = Pool_Rep
# Backend name
volume_backend_name = AS13000
```

4. Save the changes to the `/etc/cinder/cinder.conf` file and restart the `cinder-volume` service.

Inspur InStorage family volume driver

Inspur InStorage family volume driver provides OpenStack Compute instances with access to Inspur Instorage family storage system.

Inspur InStorage storage system can be used with FC or iSCSI connection.

This documentation explains how to configure and connect the block storage nodes to Inspur InStorage family storage system.

Supported operations

- Create, list, delete, attach (map), and detach (unmap) volumes.
- Create, list and delete volume snapshots.
- Create a volume from a snapshot.
- Copy an image to a volume.

- Copy a volume to an image.
- Clone a volume.
- Extend a volume.
- Retype a volume.
- Manage and unmanage a volume.
- Create, list, and delete consistency group.
- Create, list, and delete consistency group snapshot.
- Modify consistency group (add or remove volumes).
- Create consistency group from source.
- Failover and Failback support.

Configure Inspur InStorage iSCSI/FC backend

This section details the steps required to configure the Inspur InStorage Cinder Driver for single FC or iSCSI backend.

1. In the `cinder.conf` configuration file under the `[DEFAULT]` section, set the enabled_backends parameter with the iSCSI or FC back-end group

   - For Fibre Channel:

   ```
   [DEFAULT]
   enabled_backends = instorage-fc-1
   ```

   - For iSCSI:

   ```
   [DEFAULT]
   enabled_backends = instorage-iscsi-1
   ```

2. Add a back-end group section for back-end group specified in the enabled_backends parameter
3. In the newly created back-end group section, set the following configuration options:

   - For Fibre Channel:

   ```
   [instorage-fc-1]
   # Management IP of Inspur InStorage storage array
   san_ip = 10.0.0.10
   # Management Port of Inspur InStorage storage array, by default set␣
   ↪to 22
   san_ssh_port = 22
   # Management username of Inspur InStorage storage array
   san_login = username
   # Management password of Inspur InStorage storage array
   san_password = password
   # Private key for Inspur InStorage storage array
   san_private_key = path/to/the/private/key
   # The Pool used to allocated volumes
   instorage_mcs_volpool_name = Pool0
   ```

(continues on next page)

(continued from previous page)

```
# The driver path
volume_driver = cinder.volume.drivers.inspur.instorage.instorage_fc.
↪InStorageMCSFCDriver
# Backend name
volume_backend_name = instorage_fc
```

- For iSCSI:

```
[instorage-iscsi-1]
# Management IP of Inspur InStorage storage array
san_ip = 10.0.0.10
# Management Port of Inspur InStorage storage array, by default set␣
↪to 22
san_ssh_port = 22
# Management username of Inspur InStorage storage array
san_login = username
# Management password of Inspur InStorage storage array
san_password = password
# Private key for Inspur InStorage storage array
san_private_key = path/to/the/private/key
# The Pool used to allocated volumes
instorage_mcs_volpool_name = Pool0
# The driver path
volume_driver = cinder.volume.drivers.inspur.instorage.instorage_
↪iscsi.InStorageMCSISCSIDriver
# Backend name
volume_backend_name = instorage_iscsi
```

Note

When both `san_password` and `san_private_key` are provide, the driver will use private key prefer to password.

4. Save the changes to the `/etc/cinder/cinder.conf` file and restart the `cinder-volume` service.

Intel Rack Scale Design (RSD) driver

The Intel Rack Scale Design volume driver is a block storage driver providing NVMe-oF support for RSD storage.

System requirements

To use the RSD driver, the following requirements are needed:

- The driver only supports RSD API at version 2.4 or later.
- The driver requires rsd-lib.
- `cinder-volume` should be running on one of the composed node in RSD, and have access to the PODM url.

- All the `nova-compute` services should be running on the composed nodes in RSD.
- All the `cinder-volume` and `nova-compute` nodes should have installed `dmidecode` and the latest `nvme-cli` with connect/disconnect subcommands.

Supported operations

- Create, delete volumes.
- Attach, detach volumes.
- Copy an image to a volume.
- Copy a volume to an image.
- Create, delete snapshots.
- Create a volume from a snapshot.
- Clone a volume.
- Extend a volume.
- Get volume statistics.

Configuration

On `cinder-volume` nodes, using the following configurations in your `/etc/cinder/cinder.conf`:

```
volume_driver = cinder.volume.drivers.rsd.RSDDriver
```

The following table contains the configuration options supported by the RSD driver:

Table 45: Description of RSD configuration options

Configuration option = Default value	Description
`podm_password` = <>	(String) Password of PODM service
`podm_url` = <>	(String) URL of PODM service
`podm_username` = <>	(String) Username of PODM service

Kaminario K2 all-flash array iSCSI and FC volume drivers

Kaminarios K2 all-flash array leverages a unique software-defined architecture that delivers highly valued predictable performance, scalability and cost-efficiency.

Kaminarios K2 all-flash iSCSI and FC arrays can be used in OpenStack Block Storage for providing block storage using `KaminarioISCSIDriver` class and `KaminarioFCDriver` class respectively.

This documentation explains how to configure and connect the block storage nodes to one or more K2 all-flash arrays.

Driver requirements

- Kaminarios K2 all-flash iSCSI and/or FC array
- K2 REST API version >= 2.2.0
- K2 version 5.8 or later are supported

- `krest` python library(version 1.3.1 or later) should be installed on the Block Storage node using **`sudo pip install krest`**
- The Block Storage Node should also have a data path to the K2 array for the following operations:
  - Create a volume from snapshot
  - Clone a volume
  - Copy volume to image
  - Copy image to volume
  - Retype dedup without replication<->nodedup without replication

Supported operations

- Create, delete, attach, and detach volumes.
- Create and delete volume snapshots.
- Create a volume from a snapshot.
- Copy an image to a volume.
- Copy a volume to an image.
- Clone a volume.
- Extend a volume.
- Retype a volume.
- Manage and unmanage a volume.
- Replicate volume with failover and failback support to K2 array.

Limitations and known issues

If your OpenStack deployment is not setup to use multipath, the network connectivity of the K2 all-flash array will use a single physical port.

This may significantly limit the following benefits provided by K2:

- available bandwidth
- high-availability
- non disruptive-upgrade

The following steps are required to setup multipath access on the Compute and the Block Storage nodes

1. Install multipath software on both Compute and Block Storage nodes.

   For example:

   ```
   # apt-get install sg3-utils multipath-tools
   ```

2. In the `[libvirt]` section of the `nova.conf` configuration file, specify `volume_use_multipath=True`. This option is valid for both iSCSI and FC drivers. In versions prior to Newton, the option was called `iscsi_use_multipath`.

   Additional resources: Kaminario Host Configuration Guide for Linux (for configuring multipath)

3. Restart the compute service for the changes to take effect.

```
# service nova-compute restart
```

Configure single Kaminario iSCSI/FC back end

This section details the steps required to configure the Kaminario Cinder Driver for single FC or iSCSI backend.

1. In the `cinder.conf` configuration file under the `[DEFAULT]` section, set the `scheduler_default_filters` parameter:

```
[DEFAULT]
scheduler_default_filters = DriverFilter,CapabilitiesFilter
```

   See following documents for more information: *Cinder Scheduler Filters* and *Configure and use driver filter and weighing for scheduler*.

2. Under the `[DEFAULT]` section, set the enabled_backends parameter with the iSCSI or FC back-end group

```
[DEFAULT]
# For iSCSI
enabled_backends = kaminario-iscsi-1

# For FC
# enabled_backends = kaminario-fc-1
```

3. Add a back-end group section for back-end group specified in the enabled_backends parameter
4. In the newly created back-end group section, set the following configuration options:

```
[kaminario-iscsi-1]
# Management IP of Kaminario K2 All-Flash iSCSI/FC array
san_ip = 10.0.0.10
# Management username of Kaminario K2 All-Flash iSCSI/FC array
san_login = username
# Management password of Kaminario K2 All-Flash iSCSI/FC array
san_password = password
# Enable Kaminario K2 iSCSI/FC driver
volume_driver = cinder.volume.drivers.kaminario.kaminario_iscsi.
↪KaminarioISCSIDriver
# volume_driver = cinder.volume.drivers.kaminario.kaminario_fc.
↪KaminarioFCDriver

# Backend name
# volume_backend_name = kaminario_fc_1
volume_backend_name = kaminario_iscsi_1

# K2 driver calculates max_oversubscription_ratio on setting below
# option as True. Default value is False
# auto_calc_max_oversubscription_ratio = False
```

(continues on next page)

(continued from previous page)

```
# Set a limit on total number of volumes to be created on K2 array, for␣
↪example:
# filter_function = "capabilities.total_volumes < 250"

# For replication, replication_device must be set and the replication␣
↪peer must be configured
# on the primary and the secondary K2 arrays
# Syntax:
#     replication_device = backend_id:<s-array-ip>,login:<s-username>,
↪password:<s-password>,rpo:<value>
# where:
#     s-array-ip is the secondary K2 array IP
#     rpo must be either 60(1 min) or multiple of 300(5 min)
# Example:
# replication_device = backend_id:10.0.0.50,login:kaminario,
↪password:kaminario,rpo:300

# Suppress requests library SSL certificate warnings on setting this␣
↪option as True
# Default value is 'False'
# suppress_requests_ssl_warnings = False
```

5. Restart the Block Storage services for the changes to take effect:

```
# service cinder-api restart
# service cinder-scheduler restart
# service cinder-volume restart
```

Setting multiple Kaminario iSCSI/FC back ends

The following steps are required to configure multiple K2 iSCSI/FC backends:

1. In the `cinder.conf` file under the [DEFAULT] section, set the enabled_backends parameter with the comma-separated iSCSI/FC back-end groups.

```
[DEFAULT]
enabled_backends = kaminario-iscsi-1, kaminario-iscsi-2, kaminario-iscsi-3
```

2. Add a back-end group section for each back-end group specified in the enabled_backends parameter
3. For each back-end group section, enter the configuration options as described in the above section `Configure single Kaminario iSCSI/FC back end`

   See *Configure multiple-storage back ends* for additional information.
4. Restart the cinder volume service for the changes to take effect.

```
# service cinder-volume restart
```

Creating volume types

Create volume types for supporting volume creation on the multiple K2 iSCSI/FC backends. Set following extras-specs in the volume types:

- volume_backend_name : Set value of this spec according to the value of `volume_backend_name` in the back-end group sections. If only this spec is set, then dedup Kaminario cinder volumes will be created without replication support

```
$ openstack volume type create kaminario_iscsi_dedup_noreplication
$ openstack volume type set --property volume_backend_name=kaminario_
↪iscsi_1 \
  kaminario_iscsi_dedup_noreplication
```

- kaminario:thin_prov_type : Set this spec in the volume type for creating nodedup Kaminario cinder volumes. If this spec is not set, dedup Kaminario cinder volumes will be created.
- kaminario:replication : Set this spec in the volume type for creating replication supported Kaminario cinder volumes. If this spec is not set, then Kaminario cinder volumes will be created without replication support.

```
$ openstack volume type create kaminario_iscsi_dedup_replication
$ openstack volume type set --property volume_backend_name=kaminario_
↪iscsi_1 \
  kaminario:replication=enabled kaminario_iscsi_dedup_replication

$ openstack volume type create kaminario_iscsi_nodedup_replication
$ openstack volume type set --property volume_backend_name=kaminario_
↪iscsi_1 \
  kaminario:replication=enabled kaminario:thin_prov_type=nodedup \
  kaminario_iscsi_nodedup_replication

$ openstack volume type create kaminario_iscsi_nodedup_noreplication
$ openstack volume type set --property volume_backend_name=kaminario_
↪iscsi_1 \
  kaminario:thin_prov_type=nodedup kaminario_iscsi_nodedup_noreplication
```

Supported retype cases

The following are the supported retypes for Kaminario cinder volumes:

- Nodedup-noreplication <> Nodedup-replication

```
$ cinder retype volume-id new-type
```

- Dedup-noreplication <> Dedup-replication

```
$ cinder retype volume-id new-type
```

- Dedup-noreplication <> Nodedup-noreplication

```
$ cinder retype --migration-policy on-demand volume-id new-type
```

For non-supported cases, try combinations of the **`cinder retype`** command.

Driver options

The following table contains the configuration options that are specific to the Kaminario K2 FC and iSCSI Block Storage drivers.

Table 46: Description of Kaminario configuration options

Configuration option = Default value	Description
`auto_calc_max_oversubscriptio` = `False`	(Boolean) K2 driver will calculate max_oversubscription_ratio on setting this option as True.
`disable_discovery` = `False`	(Boolean) Disabling iSCSI discovery (sendtargets) for multipath connections on K2 driver.

KIOXIA Kumoscale NVMeOF Driver

KIOXIA Kumoscale volume driver provides OpenStack Compute instances with access to KIOXIA Kumoscale NVMeOF storage systems.

This documentation explains how to configure Cinder for use with the KIOXIA Kumoscale storage backend system.

Driver options

The following table contains the configuration options supported by the KIOXIA Kumoscale NVMeOF driver.

Table 47: Description of KIOXIA Kumoscale configuration options

Configuration option = Default value	Description
`kioxia_block_size` = `4096`	(Integer) Volume block size in bytes - 512 or 4096 (Default).
`kioxia_cafile` = `None`	(String) Cert for provisioner REST API SSL
`kioxia_desired_bw_per_gb` = `0`	(Integer) Desired bandwidth in B/s per GB.
`kioxia_desired_iops_per_gb` = `0`	(Integer) Desired IOPS/GB.
`kioxia_max_bw_per_gb` = `0`	(Integer) Upper limit for bandwidth in B/s per GB.
`kioxia_max_iops_per_gb` = `0`	(Integer) Upper limit for IOPS/GB.
`kioxia_max_replica_down_time` = `0`	(Integer) Replicated volume max downtime for replica in minutes.
`kioxia_num_replicas` = `1`	(Integer) Number of volume replicas.
`kioxia_provisioning_type` = `THICK`	(String(choices=[THICK, THIN])) Thin or thick volume, Default thick.
`kioxia_same_rack_allowed` = `False`	(Boolean) Can more than one replica be allocated to same rack.
`kioxia_snap_reserved_space_percent` = `0`	(Integer) Percentage of the parent volume to be used for log.
`kioxia_snap_vol_reserved_space_per` = `0`	(Integer) Writable snapshot percentage of parent volume used for log.
`kioxia_snap_vol_span_allowed` = `True`	(Boolean) Allow span in snapshot volume - Default True.
`kioxia_span_allowed` = `True`	(Boolean) Allow span - Default True.
`kioxia_token` = `None`	(String) KumoScale Provisioner auth token.
`kioxia_url` = `None`	(String) KumoScale provisioner REST API URL
`kioxia_vol_reserved_space_percentag` = `0`	(Integer) Thin volume reserved capacity allocation percentage.
`kioxia_writable` = `False`	(Boolean) Volumes from snapshot writeable or not.

Supported operations

- Create, list, delete, attach and detach volumes
- Create, list and delete volume snapshots
- Create a volume from a snapshot
- Copy an image to a volume.
- Copy a volume to an image.
- Create volume from snapshot
- Clone a volume
- Extend a volume

Configure KIOXIA Kumoscale NVMeOF backend

This section details the steps required to configure the KIOXIA Kumoscale storage cinder driver.

1. In the `cinder.conf` configuration file under the `[DEFAULT]` section, set the enabled_backends parameter.

```
[DEFAULT]
enabled_backends = kumoscale-1
```

2. Add a backend group section for the backend group specified in the enabled_backends parameter.

3. In the newly created backend group section, set the following configuration options:

```
[kumoscale-1]
# Backend name
volume_backend_name=kumoscale-1
# The driver path
volume_driver=cinder.volume.drivers.kioxia.kumoscale.
↪KumoScaleBaseVolumeDriver
# Kumoscale provisioner URL
kioxia_url=https://70.0.0.13:30100
# Kumoscale provisioner cert file
kioxia_cafile=/etc/kioxia/ssdtoolbox.pem
# Kumoscale provisioner token
token=eyJhbGciOiJIUzI1NiJ9...
```

Lenovo Fibre Channel and iSCSI drivers

The `LenovoFCDriver` and `LenovoISCSIDriver` Cinder drivers allow Lenovo S-Series arrays to be used for block storage in OpenStack deployments.

System requirements

To use the Lenovo drivers, the following are required:

- Lenovo S2200, S3200, DS2200, DS4200 or DS6200 array with:
  - iSCSI or FC host interfaces
  - G22x firmware or later
- Network connectivity between the OpenStack host and the array management interfaces
- HTTPS or HTTP must be enabled on the array

Supported operations

- Create, delete, attach, and detach volumes.
- Create, list, and delete volume snapshots.
- Create a volume from a snapshot.
- Copy an image to a volume.
- Copy a volume to an image.

- Clone a volume.
- Extend a volume.
- Migrate a volume with back-end assistance.
- Retype a volume.
- Manage and unmanage a volume.

Note

The generic grouping functionality supported in the G265 and later firmware is not supported by OpenStack Cinder due to differences in the grouping models used in Cinder and the S-Series firmware.

Configuring the array

1. Verify that the array can be managed using an HTTPS connection. HTTP can also be used if `hpmsa_api_protocol=http` is placed into the appropriate sections of the `cinder.conf` file, but this option is deprecated and will be removed in a future release.

   Confirm that virtual pools A and B are present if you plan to use virtual pools for OpenStack storage.

2. Edit the `cinder.conf` file to define a storage back-end entry for each storage pool on the array that will be managed by OpenStack. Each entry consists of a unique section name, surrounded by square brackets, followed by options specified in `key=value` format.

   - The `lenovo_pool_name` value specifies the name of the storage pool on the array.
   - The `volume_backend_name` option value can be a unique value, if you wish to be able to assign volumes to a specific storage pool on the array, or a name that is shared among multiple storage pools to let the volume scheduler choose where new volumes are allocated.
   - The rest of the options will be repeated for each storage pool in a given array:
     - `volume_driver` specifies the Cinder driver name.
     - `san_ip` specifies the IP addresses or host names of the arrays management controllers.
     - `san_login` and `san_password` specify the username and password of an array user account with `manage` privileges.
     - `driver_use_ssl` should be set to `true` to enable use of the HTTPS protocol.
     - `lenovo_iscsi_ips` specifies the iSCSI IP addresses for the array if using the iSCSI transport protocol.

   In the examples below, two back ends are defined, one for pool A and one for pool B, and a common `volume_backend_name` is used so that a single volume type definition can be used to allocate volumes from both pools.

   Example: iSCSI example back-end entries

```
[pool-a]
lenovo_pool_name = A
volume_backend_name = lenovo-array
volume_driver = cinder.volume.drivers.lenovo.lenovo_iscsi.
```

(continues on next page)

(continued from previous page)

```
↪LenovoISCSIDriver
san_ip = 10.1.2.3
san_login = manage
san_password = !manage
lenovo_iscsi_ips = 10.2.3.4,10.2.3.5
driver_use_ssl = true

[pool-b]
lenovo_pool_name = B
volume_backend_name = lenovo-array
volume_driver = cinder.volume.drivers.lenovo.lenovo_iscsi.
↪LenovoISCSIDriver
san_ip = 10.1.2.3
san_login = manage
san_password = !manage
lenovo_iscsi_ips = 10.2.3.4,10.2.3.5
driver_use_ssl = true
```

Example: Fibre Channel example back-end entries

```
[pool-a]
lenovo_pool_name = A
volume_backend_name = lenovo-array
volume_driver = cinder.volume.drivers.lenovo.lenovo_fc.LenovoFCDriver
san_ip = 10.1.2.3
san_login = manage
san_password = !manage
driver_use_ssl = true

[pool-b]
lenovo_pool_name = B
volume_backend_name = lenovo-array
volume_driver = cinder.volume.drivers.lenovo.lenovo_fc.LenovoFCDriver
san_ip = 10.1.2.3
san_login = manage
san_password = !manage
driver_use_ssl = true
```

3. If HTTPS is not enabled in the array, add `lenovo_api_protocol = http` in each of the back-end definitions.
4. If HTTPS is enabled, you can enable certificate verification with the option `driver_ssl_cert_verify = True`. You may also use the `driver_ssl_cert_path` option to specify the path to a CA_BUNDLE file containing CAs other than those in the default list.
5. Modify the `[DEFAULT]` section of the `cinder.conf` file to add an `enabled_backends` parameter specifying the back-end entries you added, and a `default_volume_type` parameter specifying the name of a volume type that you will create in the next step.

   Example: [DEFAULT] section changes

```
[DEFAULT]
# ...
enabled_backends = pool-a,pool-b
default_volume_type = lenovo
```

6. Create a new volume type for each distinct `volume_backend_name` value that you added to the `cinder.conf` file. The example below assumes that the same `volume_backend_name=lenovo-array` option was specified in all of the entries, and specifies that the volume type `lenovo` can be used to allocate volumes from any of them.

   Example: Creating a volume type

```
$ openstack volume type create lenovo
$ openstack volume type set --property volume_backend_name=lenovo-array␣
↪lenovo
```

7. After modifying the `cinder.conf` file, restart the `cinder-volume` service.

Driver-specific options

The following table contains the configuration options that are specific to the Lenovo drivers.

Table 48: Description of Lenovo configuration options

Configuration option = Default value	Description
`lenovo_iscsi_ips = []`	(List of String) List of comma-separated target iSCSI IP addresses.
`lenovo_pool_name = A`	(String) Pool or Vdisk name to use for volume creation.
`lenovo_pool_type = virtual`	(String(choices=[linear, virtual])) linear (for VDisk) or virtual (for Pool).
`lenovo_api_protocol = https`	(String(choices=[http, https])) Lenovo api interface protocol. **DEPRECATED**
`lenovo_verify_certificate = False`	(Boolean) Whether to verify Lenovo array SSL certificate. **DEPRECATED**
`lenovo_verify_certificate_pat = None`	(String) Lenovo array SSL certificate path. **DEPRECATED**

Lightbits Cinder Driver

The Lightbits(TM) OpenStack driver enables OpenStack clusters to use Lightbits clustered storage servers. This documentation explains how to configure Cinder for use with the Lightbits storage backend system.

Supported operations

- Create volume
- Delete volume
- Attach volume
- Detach volume

- Create image from volume
- Live migration
- Volume replication
- Thin provisioning
- Multi-attach
- Supported vendor driver
- Extend volume
- Create snapshot
- Delete snapshot
- Create volume from snapshot
- Create volume from volume (clone)
- Active active deployment support
- Volume retype (host assisted)

Lightbits OpenStack Driver Components

The Lightbits OpenStack driver has three components:

- Cinder driver
- Nova libvirt volume driver
- os_brick initiator connector

In addition, it requires the Lightbits `discovery-client`, provided with product. The os_brick connector uses the Lightbits `discovery-client` to communicate with Lightbits NVMe/TCP discovery services.

The Cinder Driver

The Cinder driver integrates with Cinder and performs REST operations against the Lightbits storage cluster. To enable the driver, add the following to Cinders configuration file

```
enabled_backends = lightos,<any other storage backend you use>
```

and

```
[lightos]
volume_driver = cinder.volume.drivers.lightos.LightOSVolumeDriver
volume_backend_name = lightos
lightos_api_address = <TARGET_ACCESS_IPS>
lightos_api_port = 443
lightos_jwt=<LIGHTOS_JWT>
lightos_default_num_replicas = 3
lightos_default_compression_enabled = False
lightos_api_service_timeout=30
```

- `TARGET_ACCESS_IPS` are the Lightbits cluster nodes access IPs. Multiple nodes should be separated by commas. For example: `lightos_api_address = 192.168.67.78,192.168.34.56,192.168.12.17`. These IPs are where the driver looks for the Lightbits clusters REST API servers.
- `LIGHTOS_JWT` is the JWT (JSON Web Token) that is located at the Lightbits installation controller. You can find the jwt at `~/lightos-default-admin-jwt`.
- The default number of replicas for volumes is 3, and valid values for `lightos_default_num_replicas` are 1, 2, or 3.
- The default compression setting is False (i.e., data is uncompressed). The default compression setting can also be True to indicate that new volumes should be created compressed, assuming no other compression setting is specified via the volume type. To control compression on a per-volume basis, create volume types for compressed and uncompressed, and use them as appropriate.
- The default time to wait for API service response is 30 seconds per API endpoint.

Creating volumes with non-default compression and number of replicas settings can be done through the volume types mechanism. To create a new volume type with compression enabled:

```
$ openstack volume type create --property compression='<is> True' volume-with-
↪compression
```

To create a new volume type with one replica:

```
$ openstack volume type create --property lightos:num_replicas=1 volume-with-
↪one-replica
```

To create a new type for a compressed volume with three replicas:

```
$ openstack volume type create --property compression='<is> True' --property␣
↪lightos:num_replicas=3 volume-with-three-replicas-and-compression
```

Then create a new volume with one of these volume types:

```
$ openstack volume create --size <size> --type <type name> <vol name>
```

NVNe/TCP and Asymmetric Namespace Access (ANA)

The Lightbits clusters expose their volumes using NVMe/TCP Asynchronous Namespace Access (ANA). ANA is a relatively new feature in the NVMe/TCP stack in Linux but it is fully supported in Ubuntu 20.04. Each compute host in the OpenStack cluster needs to be ANA-capable to provide OpenStack VMs with Lightbits volumes over NVMe/TCP. For more information on how to set up the compute nodes to use ANA, see the CentOS Linux Cluster Client Software Installation section of the Lightbits(TM) Cluster Installation and Initial Configuration Guide.

Note

In the current version, if any of the cluster nodes changes its access IPs, the Cinder drivers configuration file should be updated with the cluster nodes access IPs and restarted. As long as the Cinder driver can access at least one cluster access IP it will work, but will be susceptible to cluster node failures.

Driver options

The following table contains the configuration options supported by the Lightbits Cinder driver.

Table 49: Description of Lightbits cluster configuration options

Configuration option = Default value	Description
lightos_api = None	(List of IPAddress) The IP addresses of the LightOS API servers separated by commas.
lightos_api = 443	(Port(min=0, max=65535)) The TCP/IP port at which the LightOS API endpoints listen. Port 443 is used for HTTPS and other values are used for HTTP.
lightos_api = 5	(Integer) The maximum number of calls to the LightOS when creating snapshots. The default is 5 calls.
lightos_api = 30	(Integer) The default amount of time (in seconds) to wait for an API endpoint response.
lightos_def = False	(Boolean) Set to True to create new volumes compressed assuming no other compression setting is specified via the volumes type.
lightos_def = 3	(Integer(min=1, max=3)) The default number of replicas to create for each volume.
lightos_jwt = None	(String) JWT to be used for volume and snapshot operations with the LightOS cluster. Do not set this parameter if the cluster is installed with multi-tenancy disabled.
lightos_use = True	(Boolean) IPACL work in conjunction with the standard NVME ACL. A host must be in both the IPACL and the ACL of a volume to access that volume. Cinder always sets the volume's ACL. If lightos_use_ipacl is set to True, Cinder will also add the host's IP addresses to a volume IPACL. If set to False, any IP address may access the volume. The default is True.

Active active deployment support

To enable active-active deployment, follow these steps:

1. Activate the active-active mode by setting the cluster option in the DEFAULT section.
2. Configure the Distributed Lock Manager (DLM) such as Redis or etcd in the coordination section.

These options should be added to the cinder.conf file:

```
[DEFAULT]
cluster = <cluster_name>

[coordination]
backend_url = <coordination_backend_url>
```

For more detailed instructions, please refer to the guidelines at:: https://docs.openstack.org/cinder/latest/contributor/high_availability.html

LINSTOR driver

The LINSTOR driver allows Cinder to use DRBD/LINSTOR instances.

External package installation

The driver requires the `python-linstor` package for communication with the LINSTOR Controller. Install the package from PYPI using the following command:

```
$ python -m pip install python-linstor
```

Configuration

Set the following option in the `cinder.conf` file for the DRBD transport:

```
volume_driver = cinder.volume.drivers.linstordrv.LinstorDrbdDriver
```

Or use the following for iSCSI transport:

```
volume_driver = cinder.volume.drivers.linstordrv.LinstorIscsiDriver
```

The following table contains the configuration options supported by the LINSTOR driver:

Table 50: Description of LINSTOR configuration options

Configuration option = Default value	Description
`linstor_autoplace_` = `0`	(Integer) Autoplace replication count on volume deployment. 0 = Full cluster replication without autoplace, 1 = Single node deployment without replication, 2 or greater = Replicated deployment with autoplace.
`linstor_controller` = `True`	(Boolean) True means Cinder node is a diskless LINSTOR node.
`linstor_default_bl` = `4096`	(Integer) Default Block size for Image restoration. When using iSCSI transport, this option specifies the block size.
`linstor_default_st` = `DfltStorPool`	(String) Default Storage Pool name for LINSTOR.
`linstor_default_ur` = `linstor://localhost`	(String) Default storage URI for LINSTOR.
`linstor_default_vo` = `drbd-vg`	(String) Default Volume Group name for LINSTOR. Not Cinder Volume.
`linstor_volume_dow` = `4096`	(Float) Default volume downscale size in KiB = 4 MiB.

MacroSAN Fibre Channel and iSCSI drivers

The `MacroSANFCDriver` and `MacroSANISCSIDriver` Cinder drivers allow the MacroSAN Storage arrays to be used for Block Storage in OpenStack deployments.

System requirements

To use the MacroSAN drivers, the following are required:

- MacroSAN Storage arrays with: - iSCSI or FC host interfaces - Enable RESTful service on the MacroSAN Storage Appliance. (The service is automatically turned on in the device. You can check if *python /odsp/scripts/devop/devop.py* is available via *ps -aux|grep python*.)
- Network connectivity between the OpenStack host and the array management interfaces
- HTTPS or HTTP must be enabled on the array

When creating a volume from image, install the `multipath` tool and add the following configuration keys for each backend section or in `[backend_defaults]` section as a common configuration for all backends in `/etc/cinder/cinder.conf` file:

```
[cinder-iscsi-a]
use_multipath_for_image_xfer = True
```

When creating a instance from image, install the `multipath` tool and add the following configuration keys in the `[libvirt]` configuration group of the `/etc/nova/nova.conf` file:

```
iscsi_use_multipath = True
```

Supported operations

- Create, delete, attach, and detach volumes.
- Create, list, and delete volume snapshots.
- Create a volume from a snapshot.
- Copy an image to a volume.
- Copy a volume to an image.
- Clone a volume.
- Extend a volume.
- Volume Migration (Host Assisted).
- Volume Migration (Storage Assisted).
- Retype a volume.
- Manage and unmanage a volume.
- Manage and unmanage a snapshot.
- Volume Replication.
- Thin Provisioning.

Configuring the array

1. Verify that the array can be managed via an HTTPS connection.

   Confirm that virtual pools A and B are present if you plan to use virtual pools for OpenStack storage.

2. Edit the `cinder.conf` file to define a storage backend entry for each storage pool on the array that will be managed by OpenStack. Each entry consists of a unique section name, surrounded by square brackets, followed by options specified in a `key=value` format.

   - The `volume_backend_name` option value can be a unique value, if you wish to be able to assign volumes to a specific storage pool on the array, or a name that is shared among multiple storage pools to let the volume scheduler choose where new volumes are allocated.

   In the examples below, two back ends are defined, one for pool A and one for pool B.

   - Add the following configuration keys in the configuration group of enabled_backends of the `/etc/cinder/cinder.conf` file:

   iSCSI example back-end entries

```
[DEFAULT]
enabled_backends = cinder-iscsi-a, cinder-iscsi-b
rpc_response_timeout = 300

[cinder-iscsi-a]
# Storage protocol.
iscsi_protocol = iscsi

#iSCSI target user-land tool.
iscsi_helper = tgtadm

# The iSCSI driver to load
volume_driver = cinder.volume.drivers.macrosan.driver.MacroSANISCSIDriver.

# Name to give this storage back-end.
volume_backend_name = macrosan

#Choose attach/detach volumes in cinder using multipath for volume to␣
↪image and image to volume transfers.
use_multipath_for_image_xfer = True

# IP address of the Storage if attaching directly.
san_ip = 172.17.251.142, 172.17.251.143

# Storage user name.
san_login = openstack

# Storage user password.
san_password = openstack

#Choose using thin-lun or thick lun. When set san_thin_provision to True,
↪you must set
#macrosan_thin_lun_extent_size, macrosan_thin_lun_low_watermark, macrosan_
↪thin_lun_high_watermark.
san_thin_provision = False

#The name of Pool in the Storage.
macrosan_pool = Pool-a
```

(continues on next page)

(continued from previous page)

```
#The default ports used for initializing connection.
#Separate the controller by semicolons (``;``)
#Separate the ports by comma (``,``)
macrosan_client_default = eth-1:0:0, eth-1:0:1; eth-2:0:0, eth-2:0:1

#The switch to force detach volume when deleting
macrosan_force_unmap_itl = True

#Set snapshot's resource ratio
macrosan_snapshot_resource_ratio = 1

#Calculate the time spent on the operation in the log file.
macrosan_log_timing = True

# =============Optional settings=============

#Set the thin lun's extent size when the san_thin_provision is True.
macrosan_thin_lun_extent_size = 8

#Set the thin lun's low watermark when the san_thin_provision is True.
#macrosan_thin_lun_low_watermark = 8

#Set the thin lun's high watermark when the san_thin_provision is True.
macrosan_thin_lun_high_watermark = 40

#The setting of Symmetrical Dual Active Storage
macrosan_sdas_ipaddrs = 172.17.251.142, 172.17.251.143
macrosan_sdas_username = openstack
macrosan_sdas_password = openstack

#The setting of Replication Storage. When you set ip, you must set
#the macrosan_replication_destination_ports parameter.
macrosan_replication_ipaddrs = 172.17.251.142, 172.17.251.143
macrosan_replication_username = openstack
macrosan_replication_password = openstack

##The ports used for the Replication Storage.
#Separate the controller by semicolons (``,``)
#Separate the ports by semicolons (``/``)
macrosan_replication_destination_ports = eth-1:0:0/eth-1:0:1, eth-2:0:0/
↪eth-2:0:1

#Macrosan iscsi_clients list. You can configure multiple clients.␣
↪Separate the ports by semicolons (``/``)
macrosan_client = (devstack; controller1name; eth-1:0:0/eth-1:0:1; eth-
↪2:0:0/eth-2:0:1), (dev; controller2name; eth-1:0:0/eth-1:0:1; eth-2:0:0/
↪eth-2:0:1)
```

(continues on next page)

(continued from previous page)

```
[cinder-iscsi-b]
iscsi_protocol = iscsi
iscsi_helper = tgtadm
volume_driver = cinder.volume.drivers.macrosan.driver.MacroSANISCSIDriver
volume_backend_name = macrosan
use_multipath_for_image_xfer = True
san_ip = 172.17.251.142, 172.17.251.143
san_login = openstack
san_password = openstack
macrosan_pool = Pool-b
san_thin_provision = False
macrosan_force_unmap_itl = True
macrosan_snapshot_resource_ratio = 1
macrosan_log_timing = True
macrosan_client_default = eth-1:0:0, eth-1:0:1; eth-2:0:0, eth-2:0:1

macrosan_thin_lun_extent_size = 8
macrosan_thin_lun_low_watermark = 8
macrosan_thin_lun_high_watermark = 40
macrosan_sdas_ipaddrs = 172.17.251.142, 172.17.251.143
macrosan_sdas_username = openstack
macrosan_sdas_password = openstack
macrosan_replication_ipaddrs = 172.17.251.142, 172.17.251.143
macrosan_replication_username = openstack
macrosan_replication_password = openstack
macrosan_replication_destination_ports = eth-1:0:0, eth-2:0:0
macrosan_client = (devstack; controller1name; eth-1:0:0; eth-2:0:0), (dev;
↪ controller2name; eth-1:0:0; eth-2:0:0)
```

Fibre Channel example backend entries

```
[DEFAULT]
enabled_backends = cinder-fc-a, cinder-fc-b
rpc_response_timeout = 300

[cinder-fc-a]
volume_driver = cinder.volume.drivers.macrosan.driver.MacroSANFCDriver
volume_backend_name = macrosan
use_multipath_for_image_xfer = True
san_ip = 172.17.251.142, 172.17.251.143
san_login = openstack
san_password = openstack
macrosan_pool = Pool-a
san_thin_provision = False
macrosan_force_unmap_itl = True
macrosan_snapshot_resource_ratio = 1
macrosan_log_timing = True

#FC Zoning mode configured.
zoning_mode = fabric
```

(continues on next page)

(continued from previous page)

```
#The number of ports used for initializing connection.
macrosan_fc_use_sp_port_nr = 1

#In the case of an FC connection, the configuration item associated with␣
↪the port is maintained.
macrosan_fc_keep_mapped_ports = True

# =============Optional settings=============

macrosan_thin_lun_extent_size = 8
macrosan_thin_lun_low_watermark = 8
macrosan_thin_lun_high_watermark = 40
macrosan_sdas_ipaddrs = 172.17.251.142, 172.17.251.143
macrosan_sdas_username = openstack
macrosan_sdas_password = openstack
macrosan_replication_ipaddrs = 172.17.251.142, 172.17.251.143
macrosan_replication_username = openstack
macrosan_replication_password = openstack
macrosan_replication_destination_ports = eth-1:0:0, eth-2:0:0


[cinder-fc-b]
volume_driver = cinder.volume.drivers.macrosan.driver.MacroSANFCDriver
volume_backend_name = macrosan
use_multipath_for_image_xfer = True
san_ip = 172.17.251.142, 172.17.251.143
san_login = openstack
san_password = openstack
macrosan_pool = Pool-b
san_thin_provision = False
macrosan_force_unmap_itl = True
macrosan_snapshot_resource_ratio = 1
macrosan_log_timing = True
zoning_mode = fabric
macrosan_fc_use_sp_port_nr = 1
macrosan_fc_keep_mapped_ports = True

macrosan_thin_lun_extent_size = 8
macrosan_thin_lun_low_watermark = 8
macrosan_thin_lun_high_watermark = 40
macrosan_sdas_ipaddrs = 172.17.251.142, 172.17.251.143
macrosan_sdas_username = openstack
macrosan_sdas_password = openstack
macrosan_replication_ipaddrs = 172.17.251.142, 172.17.251.143
macrosan_replication_username = openstack
macrosan_replication_password = openstack
macrosan_replication_destination_ports = eth-1:0:0, eth-2:0:0
```

3. After modifying the `cinder.conf` file, restart the `cinder-volume` service.

4. Create and use volume types.

   Create and use sdas volume types

```
$ openstack volume type create sdas
$ openstack volume type set --property sdas=True sdas
```

   Create and use replication volume types

```
$ openstack volume type create replication
$ openstack volume type set --property replication_enabled=True␣
↪replication
```

Configuration file parameters

This section describes mandatory and optional configuration file parameters of the MacroSAN volume driver.

Table 51: **Mandatory parameters**

Parameter	Default value	Description	Applicable to
volume_backer	-	indicates the name of the backend	All
volume_driver	`cinder.volume.drivers.lvm.LVMVolume`	indicates the loaded driver	All
use_multip	`False`	Chose attach/detach volumes in cinder using multipath for volume to image and image to volume transfers.	All
san_thin_pr	`True`	Default volume type setting, True is thin lun, and False is thick lun.	All
macrosan_f	`True`	Force detach volume when deleting	All
macrosan_l	`True`	Calculate the time spent on the operation in the log file.	All
macrosan_s	`1`	Set snapshots resource ratio.	All
iscsi_helper	`tgtadm`	iSCSI target user-land tool to use.	iSCSI
iscsi_protoc	`iscsi`	Determines the iSCSI protocol for new iSCSI volumes, created with tgtadm.	iSCSI
macrosan_c	`None`	This is the default connection information for iscsi. This default configuration is used when no host related information is obtained.	iSCSI
zoning_mode	`True`	FC Zoning mode configured.	Fibre channel
macrosan_f	`1`	The use_sp_port_nr parameter is the number of online FC ports used by the single-ended memory when the FC connection is established in the switch non-all-pass mode. The maximum is 4.	Fibre channel
macrosan_f	`True`	In the case of an FC connection, the configuration item associated with the port is maintained.	Fibre channel

Table 52: **Optional parameters**

Parameter	Default value	Description	Applicable to
macrosan_sdas_ipad	-	The ip of Symmetrical Dual Active Storage	All
macrosan_sdas_user	-	The username of Symmetrical Dual Active Storage	All
macrosan_sdas_pass	-	The password of Symmetrical Dual Active Storage	All
macrosan_replicatio	-	The ip of replication Storage. When you set ip, you must set the macrosan_replication_destination_ports parameter.	All
macrosan_replicatio	-	The username of replication Storage	All
macrosan_replicatio	-	The password of replication Storage	All
macrosan_replicatio	-	The ports of replication storage when using replication storage.	All
macrosan_thin_lun_	`8`	Set the thin luns extent size when the san_thin_provision is True.	All
macrosan_thin_lun_	`5`	Set the thin luns low watermark when the san_thin_provision is True.	All
macrosan_thin_lun_	`20`	Set the thin luns high watermark when the san_thin_provision is True.	All
macrosan_client	`True`	Macrosan iscsi_clients list. You can configure multiple clients. You can configure it in this format: (hostname; client_name; sp1_iscsi_port; sp2_iscsi_port), E.g: (controller1; decive1; eth-1:0:0; eth-2:0:0),(controller2; decive2; eth-1:0:0/ eth-1:0:1; eth-2:0:0/ eth-2:0:1)	All

Important

Client_name has the following requirements:
[a-zA-Z0-9.-_:], the maximum number of characters is 31

The following are the MacroSAN driver specific options that may be set in *cinder.conf*:

Table 53: Description of MacroSAN configuration options

Configuration option = Default value	Description
macrosan_clien = None	(List of String) Macrosan iscsi_clients list. You can configure multiple clients. You can configure it in this format: (host; client_name; sp1_iscsi_port; sp2_iscsi_port), (host; client_name; sp1_iscsi_port; sp2_iscsi_port) Important warning, Client_name has the following requirements: [a-zA-Z0-9.-_:], the maximum number of characters is 31 E.g: (controller1; device1; eth-1:0; eth-2:0), (controller2; device2; eth-1:0/eth-1:1; eth-2:0/eth-2:1),
macrosan_clien = None	(String) This is the default connection ports name for iscsi. This default configuration is used when no host related information is obtained.E.g: eth-1:0/eth-1:1; eth-2:0/eth-2:1
macrosan_fc_ke = True	(Boolean) In the case of an FC connection, the configuration item associated with the port is maintained.
macrosan_fc_us = 1	(Integer(max=4)) The use_sp_port_nr parameter is the number of online FC ports used by the single-ended memory when the FC connection is established in the switch non-all-pass mode. The maximum is 4
macrosan_force = True	(Boolean) Force disconnect while deleting volume
macrosan_log_t = True	(Boolean) Whether enable log timing
macrosan_pool = None	(String) Pool to use for volume creation
macrosan_repli = eth-1:0/ eth-1:1, eth-2:0/ eth-2:1	(List of String) Slave device
macrosan_repli = None	(List of String) MacroSAN replication devices ip addresses
macrosan_repli = None	(String) MacroSAN replication devices password
macrosan_repli = None	(String) MacroSAN replication devices username
macrosan_sdas_ = None	(List of String) MacroSAN sdas devices ip addresses
macrosan_sdas_ = None	(String) MacroSAN sdas devices password
macrosan_sdas_ = None	(String) MacroSAN sdas devices username
macrosan_snaps = 1.0	(Float) Set snapshots resource ratio
macrosan_thin_ = 8	(Integer) Set the thin luns extent size
macrosan_thin_ = 20	(Integer) Set the thin luns high watermark
macrosan_thin_ = 5	(Integer) Set the thin luns low watermark

NEC Storage M series driver

NEC Storage M series are dual-controller disk arrays which support online maintenance. This driver supports both iSCSI and Fibre Channel.

System requirements

Supported models:

Storage model	Storage control software (firmware)	Disk type
M110, M310, M510, M710	0979 or later	SSD/HDD hybrid
M310F, M710F	0979 or later	all flash
M120, M320	1028 or later	SSD/HDD hybrid
M320F	1028 or later	all flash

Requirements:

- NEC Storage M series requires firmware revision 1028 or later to create more than 1024 volumes in a pool.
- NEC Storage DynamicDataReplication license.
- (Optional) NEC Storage IO Load Manager license for QoS.

Supported operations

- Create, delete, attach, and detach volumes.
- Create, list, and delete volume snapshots.
- Create a volume from a snapshot.
- Copy an image to a volume.
- Copy a volume to an image.
- Clone a volume.
- Extend a volume.
- Migrate a volume.
- Get volume statistics.
- Efficient non-disruptive volume backup.
- Manage and unmanage a volume.
- Manage and unmanage a snapshot.
- Attach a volume to multiple instances at once (multi-attach).
- Revert a volume to a snapshot.

Preparation

Below is minimum preparation to a disk array. For details of each command, see the NEC Storage Manager Command Reference (IS052).

- Common (iSCSI and Fibre Channel)
  1. Initial setup
     - Set IP addresses for management and BMC with the network configuration tool.
     - Enter license keys. (iSMcfg licenserelease)
  2. Create pools
     - Create pools for volumes. (iSMcfg poolbind)
     - Create pools for snapshots. (iSMcfg poolbind)
  3. Create system volumes
     - Create a Replication Reserved Volume (RSV) in one of pools. (iSMcfg ldbind)
     - Create Snapshot Reserve Areas (SRAs) in each snapshot pool. (iSMcfg srabind)
  4. (Optional) Register SSH public key
- iSCSI only
  1. Set IP addresses of each iSCSI port. (iSMcfg setiscsiport)
  2. Create LD Sets for each node. (iSMcfg addldset)
  3. Register initiator names of each node to the corresponding LD Set. (iSMcfg addldsetinitiator)
- Fibre Channel only
  1. Start access control. (iSMcfg startacc)
  2. Create LD Sets for each node. (iSMcfg addldset)
  3. Register WWPNs of each node to the corresponding LD Set. (iSMcfg addldsetpath)

Configuration

Set the following in your `cinder.conf`, and use the following options to configure it.

If you use Fibre Channel:

```
[Storage1]
volume_driver = cinder.volume.drivers.nec.volume.MStorageFCDriver
```

If you use iSCSI:

```
[Storage1]
volume_driver = cinder.volume.drivers.nec.volume.MStorageISCSIDriver
```

Also, set `volume_backend_name`.

```
[DEFAULT]
volume_backend_name = Storage1
```

This table shows configuration options for NEC Storage M series driver.

Table 54: Description of NEC Storage M Series configuration options

Configuration option = Default value	Description
nec_actual_free_capac: = False	(Boolean) Return actual free capacity.
nec_auto_accesscontro: = True	(Boolean) Configure access control automatically.
nec_backend_max_ld_cou = 1024	(Integer) Maximum number of managing sessions.
nec_backup_ldname_forn = LX:%s	(String) M-Series Storage LD name format for snapshots.
nec_backup_pools = []	(List of String) M-Series Storage backup pool number to be used.
nec_cv_ldname_format = LX:__ControlVolume_%xl	(String) M-Series Storage Control Volume name format.
nec_diskarray_name = <>	(String) Diskarray name of M-Series Storage.
nec_ismcli_fip = None	(IPAddress) FIP address of M-Series Storage iSMCLI.
nec_ismcli_password = <>	(String) Password for M-Series Storage iSMCLI.
nec_ismcli_privkey = <>	(String) Filename of RSA private key for M-Series Storage iSMCLI.
nec_ismcli_user = <>	(String) User name for M-Series Storage iSMCLI.
nec_ismview_alloptimi: = False	(Boolean) Use legacy iSMCLI command with optimization.
nec_ismview_dir = /tmp/nec/cinder	(String) Output path of iSMview file.
nec_ldname_format = LX:%s	(String) M-Series Storage LD name format for volumes.
nec_ldset = <>	(String) M-Series Storage LD Set name for Compute Node.
nec_pools = []	(List of String) M-Series Storage pool numbers list to be used.
nec_queryconfig_view = False	(Boolean) Use legacy iSMCLI command.
nec_ssh_pool_port_numl = 22	(Integer) Port number of ssh pool.
nec_unpairthread_time(= 3600	(Integer) Timeout value of Unpairthread.
nec_iscsi_portals_per_ = 0	(Integer) Max number of iSCSI portals per controller. 0 => unlimited. This option is deprecated and may be removed in the next release. **DEPRECATED**

Required options

- **nec_ismcli_fip**

  FIP address of M-Series Storage.

- **nec_ismcli_user**

User name for M-Series Storage iSMCLI.

- **nec_ismcli_password**
  Password for M-Series Storage iSMCLI.
- **nec_ismcli_privkey**
  RSA secret key file name for iSMCLI (for public key authentication only). Encrypted RSA secret key file cannot be specified.
- **nec_diskarray_name**
  Diskarray name of M-Series Storage. This parameter must be specified to configure multiple groups (multi back end) by using the same storage device (storage device that has the same `nec_ismcli_fip`). Specify the disk array name targeted by the relevant config-group for this parameter.
- **nec_backup_pools**
  Specify one pool number where snapshots are created. Multiple pools are not supported.

Timeout configuration

- **rpc_response_timeout**
  Set the timeout value in seconds. If three or more volumes can be created at the same time, the reference value is 30 seconds multiplied by the number of volumes created at the same time. Also, Specify nova parameters below in `nova.conf` file.

```
[DEFAULT]
block_device_allocate_retries = 120
block_device_allocate_retries_interval = 10
```

- **timeout server (HAProxy configuration)**
  In addition, you need to edit the following value in the HAProxy configuration file (`/etc/haproxy/haproxy.cfg`) in an environment where HAProxy is used.

```
timeout server = 600 #Specify a value greater than rpc_response_
↪timeout.
```

Run the **service haproxy reload** command after editing the value to reload the HAProxy settings.

> **Note**
>
> The OpenStack environment set up using Red Hat OpenStack Platform Director may be set to use HAProxy.

Configuration example for /etc/cinder/cinder.conf

When using one config-group

- When using `nec_ismcli_password` to authenticate iSMCLI (Password authentication):

```
[DEFAULT]
enabled_backends = Storage1
```

(continues on next page)

(continued from previous page)

```
[Storage1]
volume_driver = cinder.volume.drivers.nec.volume.MStorageISCSIDriver
volume_backend_name = Storage1
nec_ismcli_fip = 192.168.1.10
nec_ismcli_user = sysadmin
nec_ismcli_password = sys123
nec_pools = 0
nec_backup_pools = 1
```

- When using `nec_ismcli_privkey` to authenticate iSMCLI (Public key authentication):

```
[DEFAULT]
enabled_backends = Storage1

[Storage1]
volume_driver = cinder.volume.drivers.nec.volume.MStorageISCSIDriver
volume_backend_name = Storage1
nec_ismcli_fip = 192.168.1.10
nec_ismcli_user = sysadmin
nec_ismcli_privkey = /etc/cinder/id_rsa
nec_pools = 0
nec_backup_pools = 1
```

When using multi config-group (multi-backend)

- Four config-groups (backends)

  Storage1, Storage2, Storage3, Storage4

- Two disk arrays

  200000255C3A21CC(192.168.1.10)
  Example for using config-group, Storage1 and Storage2

  2000000991000316(192.168.1.20)
  Example for using config-group, Storage3 and Storage4

```
[DEFAULT]
enabled_backends = Storage1,Storage2,Storage3,Storage4

[Storage1]
volume_driver = cinder.volume.drivers.nec.volume.MStorageISCSIDriver
volume_backend_name = Gold
nec_ismcli_fip = 192.168.1.10
nec_ismcli_user = sysadmin
nec_ismcli_password = sys123
nec_pools = 0
nec_backup_pools = 2
nec_diskarray_name = 200000255C3A21CC

[Storage2]
```

(continues on next page)

(continued from previous page)

```
volume_driver = cinder.volume.drivers.nec.volume.MStorageISCSIDriver
volume_backend_name = Silver
nec_ismcli_fip = 192.168.1.10
nec_ismcli_user = sysadmin
nec_ismcli_password = sys123
nec_pools = 1
nec_backup_pools = 3
nec_diskarray_name = 200000255C3A21CC

[Storage3]
volume_driver = cinder.volume.drivers.nec.volume.MStorageISCSIDriver
volume_backend_name = Gold
nec_ismcli_fip = 192.168.1.20
nec_ismcli_user = sysadmin
nec_ismcli_password = sys123
nec_pools = 0
nec_backup_pools = 2
nec_diskarray_name = 2000000991000316

[Storage4]
volume_driver = cinder.volume.drivers.nec.volume.MStorageISCSIDriver
volume_backend_name = Silver
nec_ismcli_fip = 192.168.1.20
nec_ismcli_user = sysadmin
nec_ismcli_password = sys123
nec_pools = 1
nec_backup_pools = 3
nec_diskarray_name = 2000000991000316
```

NEC Storage V series driver

NEC Storage V series driver provides Fibre Channel and iSCSI support for NEC V series storages.

System requirements

Supported models:

Storage model	Firmware version
V100, V300	93-04-21 or later

Required storage licenses:

- iStorage Local Replication Local Replication Software

Supported operations

- Create, delete, attach, and detach volumes.
- Create, list, and delete volume snapshots.
- Create a volume from a snapshot.
- Create, list, update, and delete consistency groups.
- Create, list, and delete consistency group snapshots.
- Copy a volume to an image.
- Copy an image to a volume.
- Clone a volume.
- Extend a volume.
- Migrate a volume.
- Get volume statistics.
- Efficient non-disruptive volume backup.
- Manage and unmanage a volume.
- Attach a volume to multiple instances at once (multi-attach).
- Revert a volume to a snapshot.

Note

A volume with snapshots cannot be extended in this driver.

Configuration

Set up NEC V series storage

You need to specify settings as described below for storage systems. For details about each setting, see the users guide of the storage systems.

Common resources:

- **`All resources`**
  All storage resources, such as DP pools and host groups, can not have a name including blank space in order for the driver to use them.
- **`User accounts`**
  Create a storage device account belonging to the Administrator User Group.
- **`DP Pool`**
  Create a DP pool that is used by the driver.
- **`Resource group`**
  If using a new resource group for exclusive use by an OpenStack system, create a new resource group, and assign the necessary resources, such as LDEVs, port, and host group (iSCSI target) to the created resource.

- **Ports**
  Enable Port Security for the ports used by the driver.

If you use iSCSI:

- **Ports**
  Assign an IP address and a TCP port number to the port.

Set up NEC V series storage volume driver

Set the volume driver to NEC V series storage driver by setting the volume_driver option in the cinder.conf file as follows:

If you use Fibre Channel:

```
[Storage1]
volume_driver = cinder.volume.drivers.nec.v.nec_v_fc.VStorageFCDriver
volume_backend_name = Storage1
san_ip = 1.2.3.4
san_api_port = 23451
san_login = userid
san_password = password
nec_v_storage_id = 123456789012
nec_v_pools = pool0
```

If you use iSCSI:

```
[Storage1]
volume_driver = cinder.volume.drivers.nec.v.nec_v_iscsi.VStorageISCSIDriver
volume_backend_name = Storage1
san_ip = 1.2.3.4
san_api_port = 23451
san_login = userid
san_password = password
nec_v_storage_id = 123456789012
nec_v_pools = pool0
```

This table shows configuration options for NEC V series storage driver.

Table 55: Descript
options

Configuration option = Default value	Description
`nec_v_async_copy_check_interval` = `10`	(Integer(min=1, max=600)) Interval in seconds to c
`nec_v_compute_target_ports` = `[]`	(List of String) IDs of the storage ports used to attac
`nec_v_copy_check_interval` = `3`	(Integer(min=1, max=600)) Interval in seconds to c
`nec_v_copy_speed` = `3`	(Integer(min=1, max=15)) Copy speed of storage sy
`nec_v_discard_zero_page` = `True`	(Boolean) Enable or disable zero page reclamation i
`nec_v_exec_retry_interval` = `5`	(Integer) Retry interval in seconds for REST API ex
`nec_v_extend_timeout` = `600`	(Integer) Maximum wait time in seconds for a volun
`nec_v_group_create` = `False`	(Boolean) If True, the driver will create host groups
`nec_v_group_delete` = `False`	(Boolean) If True, the driver will delete host groups

Table 5

Configuration option = Default value	Description
`nec_v_host_mode_options = []`	(List of String) Host mode option for host group or
`nec_v_ldev_range = None`	(String) Range of the LDEV numbers in the format
`nec_v_lock_timeout = 7200`	(Integer) Maximum wait time in seconds for storage
`nec_v_lun_retry_interval = 1`	(Integer) Retry interval in seconds for REST API ad
`nec_v_lun_timeout = 50`	(Integer) Maximum wait time in seconds for adding
`nec_v_pools = []`	(List of String) Pool number[s] or pool name[s] of t
`nec_v_rest_another_ldev_mapped_retry_timeout = 600`	(Integer) Retry time in seconds when new LUN allo
`nec_v_rest_connect_timeout = 30`	(Integer) Maximum wait time in seconds for REST
`nec_v_rest_disable_io_wait = True`	(Boolean) It may take some time to detach volume a
`nec_v_rest_get_api_response_timeout = 1800`	(Integer) Maximum wait time in seconds for a respo
`nec_v_rest_job_api_response_timeout = 1800`	(Integer) Maximum wait time in seconds for a respo
`nec_v_rest_keep_session_loop_interval = 180`	(Integer) Loop interval in seconds for keeping REST
`nec_v_rest_server_busy_timeout = 7200`	(Integer) Maximum wait time in seconds when RES
`nec_v_rest_tcp_keepalive = True`	(Boolean) Enables or disables use of REST API tcp
`nec_v_rest_tcp_keepcnt = 4`	(Integer) Maximum number of transmissions for TC
`nec_v_rest_tcp_keepidle = 60`	(Integer) Wait time in seconds for sending a first TC
`nec_v_rest_tcp_keepintvl = 15`	(Integer) Interval of transmissions in seconds for TC
`nec_v_rest_timeout = 30`	(Integer) Maximum wait time in seconds for REST
`nec_v_restore_timeout = 86400`	(Integer) Maximum wait time in seconds for the res
`nec_v_snap_pool = None`	(String) Pool number or pool name of the snapshot
`nec_v_state_transition_timeout = 900`	(Integer) Maximum wait time in seconds for a volun
`nec_v_storage_id = None`	(String) Product number of the storage system.
`nec_v_target_ports = []`	(List of String) IDs of the storage ports used to attac
`nec_v_zoning_request = False`	(Boolean) If True, the driver will configure FC zoni

Required options

- **san_ip**
  IP address of SAN controller
- **san_login**
  Username for SAN controller
- **san_password**
  Password for SAN controller
- **nec_v_storage_id**
  Product number of the storage system.
- **nec_v_pools**
  Pool number(s) or pool name(s) of the DP pool.

NetApp unified driver

The NetApp unified driver is a Block Storage driver that supports multiple storage families and protocols. Currently, the only storage family supported by this driver is the clustered Data ONTAP. The storage protocol refers to the protocol used to initiate data storage and access operations on those storage systems like NVMe, iSCSI and NFS. The NetApp unified driver can be configured to provision and manage OpenStack volumes on a given storage family using a specified storage protocol.

Also, the NetApp unified driver supports over subscription or over provisioning when thin provisioned

Block Storage volumes are in use. The OpenStack volumes can then be used for accessing and storing data using the storage protocol on the storage family system. The NetApp unified driver is an extensible interface that can support new storage families and protocols.

Note

With the Juno release of OpenStack, Block Storage has introduced the concept of storage pools, in which a single Block Storage back end may present one or more logical storage resource pools from which Block Storage will select a storage location when provisioning volumes.

In releases prior to Juno, the NetApp unified driver contained some scheduling logic that determined which NetApp storage container (namely, a FlexVol volume for Data ONTAP) that a new Block Storage volume would be placed into.

With the introduction of pools, all scheduling logic is performed completely within the Block Storage scheduler, as each NetApp storage container is directly exposed to the Block Storage scheduler as a storage pool. Previously, the NetApp unified driver presented an aggregated view to the scheduler and made a final placement decision as to which NetApp storage container the Block Storage volume would be provisioned into.

NetApp clustered Data ONTAP storage family

The NetApp clustered Data ONTAP storage family represents a configuration group which provides Compute instances access to clustered Data ONTAP storage systems. At present it can be configured in Block Storage to work with NVme, iSCSI and NFS storage protocols.

NetApp iSCSI configuration for clustered Data ONTAP

The NetApp iSCSI configuration for clustered Data ONTAP is an interface from OpenStack to clustered Data ONTAP storage systems. It provisions and manages the SAN block storage entity, which is a NetApp LUN that can be accessed using the iSCSI protocol.

The iSCSI configuration for clustered Data ONTAP is a direct interface from Block Storage to the clustered Data ONTAP instance and as such does not require additional management software to achieve the desired functionality. It uses NetApp APIs to interact with the clustered Data ONTAP instance.

Configuration options

Configure the volume driver, storage family, and storage protocol to the NetApp unified driver, clustered Data ONTAP, and iSCSI respectively by setting the `volume_driver`, `netapp_storage_family` and `netapp_storage_protocol` options in the `cinder.conf` file as follows:

```
volume_driver = cinder.volume.drivers.netapp.common.NetAppDriver
netapp_storage_family = ontap_cluster
netapp_storage_protocol = iscsi
netapp_vserver = openstack-vserver
netapp_server_hostname = myhostname
netapp_server_port = port
netapp_login = username
netapp_password = password
```

Note

To use the iSCSI protocol, you must override the default value of `netapp_storage_protocol` with `iscsi`. Note that this is not the same value that is reported by the driver to the scheduler as *storage_protocol*, which is always `iSCSI` (case sensitive).

Table 56: Description of NetApp cDOT iSCSI driver configuration options

Configuration option = Default value	Description
[DEFAULT]	
netapp = None	(String) Administrative user account name used to access the storage system or proxy server.
netapp = None	(String) This option defines the type of operating system that will access a LUN exported from Data ONTAP; it is assigned to the LUN at the time it is created.
netapp = enable	(String) This option determines if storage space is reserved for LUN allocation. If enabled, LUNs are thick provisioned. If space reservation is disabled, storage space is allocated on demand.
netapp = None	(String) Password for the administrative user account specified in the netapp_login option.
netapp = (.+)	(String) This option is used to restrict provisioning to the specified pools. Specify the value of this option to be a regular expression which will be applied to the names of objects from the storage backend which represent pools in Cinder. This option is only utilized when the storage protocol is configured to use NVMe, iSCSI or FC.
netapp = None	(Unknown) Multi opt of dictionaries to represent the aggregate mapping between source and destination back ends when using whole back end replication. For every source aggregate associated with a cinder pool (NetApp FlexVol), you would need to specify the destination aggregate on the replication target device. A replication target device is configured with the configuration option replication_device. Specify this option as many times as you have replication devices. Each entry takes the standard dict config form: netapp_replication_aggregate_map = backend_id:<name_of_replication_device_section>,src_aggr_name1:dest_aggr_name1,src_aggr_name2:dest_agg
netapp = None	(String) The hostname (or IP address) for the storage system or proxy server.
netapp = None	(Integer) The TCP port to use for communication with the storage system or proxy server. If not specified, Data ONTAP drivers will use 80 for HTTP and 443 for HTTPS.
netapp = 1.2	(Floating point) The quantity to be multiplied by the requested volume size to ensure enough space is available on the virtual storage server (Vserver) to fulfill the volume creation request. Note: this option is deprecated and will be removed in favor of reserved_percentage in the Mitaka release.
netapp = 3600	(Integer) The maximum time in seconds to wait for existing SnapMirror transfers to complete before aborting during a failover.
netapp = ontap_	(String) The storage family type used on the storage system; the only valid value is ontap_cluster for using clustered Data ONTAP.
netapp = None	(String) The storage protocol to be used on the data path with the storage system.
netapp	(String) The transport protocol used when communicating with the storage system or proxy

Note

If you specify an account in the `netapp_login` that only has virtual storage server (Vserver) administration privileges (rather than cluster-wide administration privileges), some advanced features of the NetApp unified driver will not work and you may see warnings in the Block Storage logs.

Note

The driver supports iSCSI CHAP uni-directional authentication. To enable it, set the `use_chap_auth` option to `True`.

Tip

For more information on these options and other deployment and operational scenarios, visit the NetApp OpenStack website.

NetApp NVMe/TCP configuration for clustered Data ONTAP

The NetApp NVMe/TCP configuration for clustered Data ONTAP is an interface from OpenStack to clustered Data ONTAP storage systems. It provisions and manages the SAN block storage entity, which is a NetApp namespace that can be accessed using the NVMe/TCP protocol.

The NVMe/TCP configuration for clustered Data ONTAP is a direct interface from Block Storage to the clustered Data ONTAP instance and as such does not require additional management software to achieve the desired functionality. It uses NetApp APIs to interact with the clustered Data ONTAP instance.

Configuration options

Configure the volume driver, storage family, and storage protocol to the NetApp unified driver, clustered Data ONTAP, and NVMe respectively by setting the `volume_driver`, `netapp_storage_family` and `netapp_storage_protocol` options in the `cinder.conf` file as follows:

```
volume_driver = cinder.volume.drivers.netapp.common.NetAppDriver
netapp_storage_family = ontap_cluster
netapp_storage_protocol = nvme
netapp_vserver = openstack-vserver
netapp_server_hostname = myhostname
netapp_server_port = port
netapp_login = username
netapp_password = password
```

Note

To use the NVMe/TCP protocol, you must override the default value of `netapp_storage_protocol` with `nvme`. Note that this is not the same value that is reported by the driver to the scheduler as *storage_protocol*, which is always `NVMe` (case sensitive).

Note

If you specify an account in the `netapp_login` that only has virtual storage server (Vserver) administration privileges (rather than cluster-wide administration privileges), some advanced features of the NetApp unified driver will not work and you may see warnings in the Block Storage logs.

Note

The driver only supports the minimal Cinder driver features: create/delete volume and snapshots, extend volume, attack/detach volume, create volume from volume and create volume from image/snapshot.

Tip

For more information on these options and other deployment and operational scenarios, visit the NetApp OpenStack website.

NetApp NFS configuration for clustered Data ONTAP

The NetApp NFS configuration for clustered Data ONTAP is an interface from OpenStack to a clustered Data ONTAP system for provisioning and managing OpenStack volumes on NFS exports provided by the clustered Data ONTAP system that are accessed using the NFS protocol.

The NFS configuration for clustered Data ONTAP is a direct interface from Block Storage to the clustered Data ONTAP instance and as such does not require any additional management software to achieve the desired functionality. It uses NetApp APIs to interact with the clustered Data ONTAP instance.

Configuration options

Configure the volume driver, storage family, and storage protocol to NetApp unified driver, clustered Data ONTAP, and NFS respectively by setting the `volume_driver`, `netapp_storage_family`, and `netapp_storage_protocol` options in the `cinder.conf` file as follows:

```
volume_driver = cinder.volume.drivers.netapp.common.NetAppDriver
netapp_storage_family = ontap_cluster
netapp_storage_protocol = nfs
netapp_vserver = openstack-vserver
netapp_server_hostname = myhostname
netapp_server_port = port
netapp_login = username
netapp_password = password
nfs_shares_config = /etc/cinder/nfs_shares
```

Table 57: Description of NetApp cDOT NFS driver configuration options

Configuration option = Default value	Description
[DEFAULT]	
expiry_ = 720	(Integer) This option specifies the threshold for last access time for images in the NFS image cache. When a cache cleaning cycle begins, images in the cache that have not been accessed in the last M minutes, where M is the value of this parameter, will be deleted from the cache to create free space on the NFS share.
netapp_ = None	(String) This option specifies the path of the NetApp copy offload tool binary. Ensure that the binary has execute permissions set which allow the effective user of the cinder-volume process to execute the file.
netapp_ = None	(String) This option defines the type of operating system for all initiators that can access a LUN. This information is used when mapping LUNs to individual hosts or groups of hosts.
netapp_ = None	(String) Administrative user account name used to access the storage system or proxy server.
netapp_ = None	(String) This option defines the type of operating system that will access a LUN exported from Data ONTAP; it is assigned to the LUN at the time it is created.
netapp_ = None	(String) Password for the administrative user account specified in the netapp_login option.
netapp_ = (.+)	(String) This option is used to restrict provisioning to the specified pools. Specify the value of this option to be a regular expression which will be applied to the names of objects from the storage backend which represent pools in Cinder. This option is only utilized when the storage protocol is configured to use iSCSI or FC.
netapp_ = None	(Unknown) Multi opt of dictionaries to represent the aggregate mapping between source and destination back ends when using whole back end replication. For every source aggregate associated with a cinder pool (NetApp FlexVol), you would need to specify the destination aggregate on the replication target device. A replication target device is configured with the configuration option replication_device. Specify this option as many times as you have replication devices. Each entry takes the standard dict config form: netapp_replication_aggregate_map = backend_id:<name_of_replication_device_section>,src_aggr_name1:dest_aggr_name1,src_aggr_name2:dest_agg
netapp_ = None	(String) The hostname (or IP address) for the storage system or proxy server.
netapp_ = None	(Integer) The TCP port to use for communication with the storage system or proxy server. If not specified, Data ONTAP drivers will use 80 for HTTP and 443 for HTTPS.
netapp_ = 3600	(Integer) The maximum time in seconds to wait for existing SnapMirror transfers to complete before aborting during a failover.
netapp_ = ontap_	(String) The storage family type used on the storage system; the only valid value is ontap_cluster for using clustered Data ONTAP.
netapp_	(String) The storage protocol to be used on the data path with the storage system.

Note

Additional NetApp NFS configuration options are shared with the generic NFS driver. These options can be found here: *Description of NFS storage configuration options*.

Note

If you specify an account in the `netapp_login` that only has virtual storage server (Vserver) administration privileges (rather than cluster-wide administration privileges), some advanced features of the NetApp unified driver will not work and you may see warnings in the Block Storage logs.

NetApp NFS Copy Offload client

A feature was added in the Icehouse release of the NetApp unified driver that enables Image service images to be efficiently copied to a destination Block Storage volume. When the Block Storage and Image service are configured to use the NetApp NFS Copy Offload client, a controller-side copy will be attempted before reverting to downloading the image from the Image service. This improves image provisioning times while reducing the consumption of bandwidth and CPU cycles on the host(s) running the Image and Block Storage services. This is due to the copy operation being performed completely within the storage cluster.

The NetApp NFS Copy Offload client can be used in either of the following scenarios:

- The Image service is configured to store images in an NFS share that is exported from a NetApp FlexVol volume *and* the destination for the new Block Storage volume will be on an NFS share exported from a different FlexVol volume than the one used by the Image service. Both FlexVols must be located within the same cluster.
- The source image from the Image service has already been cached in an NFS image cache within a Block Storage back end. The cached image resides on a different FlexVol volume than the destination for the new Block Storage volume. Both FlexVols must be located within the same cluster.

To use this feature, you must configure the Image service, as follows:

- Set the `default_store` configuration option to `file`.
- Set the `filesystem_store_datadir` configuration option to the path to the Image service NFS export.
- Set the `show_image_direct_url` configuration option to `True`.
- Set the `show_multiple_locations` configuration option to `True`.
- Set the `filesystem_store_metadata_file` configuration option to a metadata file. The metadata file should contain a JSON object that contains the correct information about the NFS export used by the Image service.

To use this feature, you must configure the Block Storage service, as follows:

- Set the `netapp_copyoffload_tool_path` configuration option to the path to the NetApp Copy Offload binary.

  Important

This feature requires that:

- The storage system must have Data ONTAP v8.2 or greater installed.
- The vStorage feature must be enabled on each storage virtual machine (SVM, also known as a Vserver) that is permitted to interact with the copy offload client.
- To configure the copy offload workflow, enable NFS v4.0 or greater and export it from the SVM.

Tip

To download the NetApp copy offload binary to be utilized in conjunction with the `netapp_copyoffload_tool_path` configuration option, please visit the Utility Toolchest page at the NetApp Support portal (login is required).

Tip

For more information on these options and other deployment and operational scenarios, visit the NetApp OpenStack website.

NetApp-supported extra specs for clustered Data ONTAP

Extra specs enable vendors to specify extra filter criteria. The Block Storage scheduler uses the specs when the scheduler determines which volume node should fulfill a volume provisioning request. When you use the NetApp unified driver with a clustered Data ONTAP storage system, you can leverage extra specs with Block Storage volume types to ensure that Block Storage volumes are created on storage back ends that have certain properties. An example of this is when you configure QoS, mirroring, or compression for a storage back end.

Extra specs are associated with Block Storage volume types. When users request volumes of a particular volume type, the volumes are created on storage back ends that meet the list of requirements. An example of this is the back ends that have the available space or extra specs. Use the specs in the following table to configure volumes. Define Block Storage volume types by using the **`openstack volume type set`** command.

Table 58: Description of extra specs options for NetApp Unified Driver with Clustered Data ONTAP

Extra spec	Type	Description
`netapp_r`	Strin	Limit the candidate volume list based on one of the following raid types: `raid4, raid_dp`.
`netapp_d`	Strin	Limit the candidate volume list based on one of the following disk types: `ATA, BSAS, EATA, FCAL, FSAS, LUN, MSATA, SAS, SATA, SCSI, XATA, XSAS, or SSD.`
`netapp:q`	Strin	Specify the name of a QoS policy group, which defines measurable Service Level Objectives, that should be applied to the OpenStack Block Storage volume at the time of volume creation. Ensure that the QoS policy group object within Data ONTAP should be defined before an OpenStack Block Storage volume is created, and that the QoS policy group is not associated with the destination FlexVol volume.
`netapp:q`	Bool	Set to <is> True in order to instruct the driver to use an Adaptive QoS policy group for the netapp:qos_policy_group setting. Leave this unset or set to <is> False in order to use a standard QoS policy group for the netapp:qos_policy_group setting.
`netapp_m`	Bool	Limit the candidate volume list to only the ones that are mirrored on the storage controller.
`netapp_u`	Bool	Limit the candidate volume list to only the ones that are not mirrored on the storage controller.
`netapp_d`	Bool	Limit the candidate volume list to only the ones that have deduplication enabled on the storage controller.
`netapp_n`	Bool	Limit the candidate volume list to only the ones that have deduplication disabled on the storage controller.
`netapp_c`	Bool	Limit the candidate volume list to only the ones that have compression enabled on the storage controller.
`netapp_n`	Bool	Limit the candidate volume list to only the ones that have compression disabled on the storage controller.
`netapp_t`	Bool	Limit the candidate volume list to only the ones that support thin provisioning on the storage controller.
`netapp_t`	Bool	Limit the candidate volume list to only the ones that support thick provisioning on the storage controller.

NexentaStor 4.x NFS and iSCSI drivers

NexentaStor is an Open Source-driven Software-Defined Storage (OpenSDS) platform delivering unified file (NFS and SMB) and block (FC and iSCSI) storage services, runs on industry standard hardware, scales from tens of terabytes to petabyte configurations, and includes all data management functionality by default.

For NexentaStor 4.x user documentation, visit https://nexenta.com/products/downloads/nexentastor.

[1] Please note that this extra spec has a colon (`:`) in its name because it is used by the driver to assign the QoS policy group to the OpenStack Block Storage volume after it has been provisioned.

[2] In the Juno release, these negative-assertion extra specs are formally deprecated by the NetApp unified driver. Instead of using the deprecated negative-assertion extra specs (for example, `netapp_unmirrored`) with a value of `true`, use the corresponding positive-assertion extra spec (for example, `netapp_mirrored`) with a value of `false`.

Supported operations

- Create, delete, attach, and detach volumes.
- Create, list, and delete volume snapshots.
- Create a volume from a snapshot.
- Copy an image to a volume.
- Copy a volume to an image.
- Clone a volume.
- Extend a volume.
- Migrate a volume.
- Change volume type.

Nexenta iSCSI driver

The Nexenta iSCSI driver allows you to use a NexentaStor appliance to store Compute volumes. Every Compute volume is represented by a single zvol in a predefined Nexenta namespace. The Nexenta iSCSI volume driver should work with all versions of NexentaStor.

The NexentaStor appliance must be installed and configured according to the relevant Nexenta documentation. A volume and an enclosing namespace must be created for all iSCSI volumes to be accessed through the volume driver. This should be done as specified in the release-specific NexentaStor documentation.

The NexentaStor Appliance iSCSI driver is selected using the normal procedures for one or multiple backend volume drivers.

You must configure these items for each NexentaStor appliance that the iSCSI volume driver controls:

1. Make the following changes on the volume node `/etc/cinder/cinder.conf` file.

```
# Enable Nexenta iSCSI driver
volume_driver=cinder.volume.drivers.nexenta.iscsi.NexentaISCSIDriver

# IP address of NexentaStor host (string value)
nexenta_host=HOST-IP

# Username for NexentaStor REST (string value)
nexenta_user=USERNAME

# Port for Rest API (integer value)
nexenta_rest_port=8457

# Password for NexentaStor REST (string value)
nexenta_password=PASSWORD

# Volume on NexentaStor appliance (string value)
nexenta_volume=volume_name
```

Note

nexenta_volume represents a zpool which is called volume on NS appliance. It must be pre-created before enabling the driver.

1. Save the changes to the `/etc/cinder/cinder.conf` file and restart the `cinder-volume` service.

Nexenta NFS driver

The Nexenta NFS driver allows you to use NexentaStor appliance to store Compute volumes via NFS. Every Compute volume is represented by a single NFS file within a shared directory.

While the NFS protocols standardize file access for users, they do not standardize administrative actions such as taking snapshots or replicating file systems. The OpenStack Volume Drivers bring a common interface to these operations. The Nexenta NFS driver implements these standard actions using the ZFS management plane that is already deployed on NexentaStor appliances.

The Nexenta NFS volume driver should work with all versions of NexentaStor. The NexentaStor appliance must be installed and configured according to the relevant Nexenta documentation. A single-parent file system must be created for all virtual disk directories supported for OpenStack. This directory must be created and exported on each NexentaStor appliance. This should be done as specified in the release-specific NexentaStor documentation.

You must configure these items for each NexentaStor appliance that the NFS volume driver controls:

1. Make the following changes on the volume node `/etc/cinder/cinder.conf` file.

```
# Enable Nexenta NFS driver
volume_driver=cinder.volume.drivers.nexenta.nfs.NexentaNfsDriver

# Path to shares config file
nexenta_shares_config=/home/ubuntu/shares.cfg
```

Note

Add your list of Nexenta NFS servers to the file you specified with the `nexenta_shares_config` option. For example, this is how this file should look:

```
192.168.1.200:/volumes/VOLUME_NAME/NFS_SHARE http://USER:PASSWORD@192.168.
↪1.200:8457
192.168.1.201:/volumes/VOLUME_NAME/NFS_SHARE http://USER:PASSWORD@192.168.
↪1.201:8457
192.168.1.202:/volumes/VOLUME_NAME/NFS_SHARE http://USER:PASSWORD@192.168.
↪1.202:8457
```

Each line in this file represents an NFS share. The first part of the line is the NFS share URL, the second line is the connection URL to the NexentaStor Appliance.

Driver options

Nexenta Driver supports these options:

Table 59: Description of Nexenta driver configuration options

Configuration option = Default value	Description
[DEFAULT]	
nexenta_blocksize = 4096	(Integer) Block size for datasets
nexenta_chunksize = 32768	(Integer) NexentaEdge iSCSI LUN object chunk size
nexenta_client_address =	(String) NexentaEdge iSCSI Gateway client address for non-VIP service
nexenta_dataset_compres = on	(String) Compression value for new ZFS folders.
nexenta_dataset_dedup = off	(String) Deduplication value for new ZFS folders.
nexenta_dataset_descrip =	(String) Human-readable description for the folder.
nexenta_host =	(String) IP address of Nexenta SA
nexenta_iscsi_target_po = 3260	(Integer) Nexenta target portal port
nexenta_mount_point_bas = $state_path/mnt	(String) Base directory that contains NFS share mount points
nexenta_nbd_symlinks_di = /dev/disk/by-path	(String) NexentaEdge logical path of directory to store symbolic links to NBDs
nexenta_nms_cache_volro = True	(Boolean) If set True cache NexentaStor appliance volroot option value.
nexenta_password = nexenta	(String) Password to connect to Nexenta SA
nexenta_rest_port = 0	(Integer) HTTP(S) port to connect to Nexenta REST API server. If it is equal zero, 8443 for HTTPS and 8080 for HTTP is used
nexenta_rest_protocol = auto	(String) Use http or https for REST connection (default auto)
nexenta_rrmgr_compressi = 0	(Integer) Enable stream compression, level 1..9. 1 - gives best speed; 9 - gives best compression.
nexenta_rrmgr_connectio = 2	(Integer) Number of TCP connections.
nexenta_rrmgr_tcp_buf_s = 4096	(Integer) TCP Buffer size in KiloBytes.
nexenta_shares_config = /etc/cinder/nfs_shares	(String) File with the list of available nfs shares
nexenta_sparse = False	(Boolean) Enables or disables the creation of sparse datasets
nexenta_sparsed_volumes = True	(Boolean) Enables or disables the creation of volumes as sparsed files that take no space. If disabled (False), volume is created as a regular file, which takes a long time.
nexenta_target_group_pr = cinder/	(String) Prefix for iSCSI target groups on SA
nexenta_target_prefix = iqn.1986-03.com.sun:02:cinder-	(String) IQN prefix for iSCSI targets
nexenta_use_https = True	(Boolean) Use secure HTTP for REST connection (default True)
nexenta_user = admin	(String) User name to connect to Nexenta SA
nexenta_volume = cinder	(String) SA Pool that holds all volumes

NexentaStor 5.x NFS and iSCSI drivers

NexentaStor is an Open Source-driven Software-Defined Storage (OpenSDS) platform delivering unified file (NFS and SMB) and block (FC and iSCSI) storage services. NexentaStor runs on industry standard hardware, scales from tens of terabytes to petabyte configurations, and includes all data management functionality by default.

For user documentation, see the Nexenta Documentation Center.

Supported operations

- Create, delete, attach, and detach volumes.
- Create, list, and delete volume snapshots.
- Create a volume from a snapshot.
- Copy an image to a volume.
- Copy a volume to an image.
- Clone a volume.
- Extend a volume.
- Migrate a volume.
- Change volume type.
- Get volume statistics.
- Revert a volume to a snapshot.
- Manage and unmanage volumes and snapshots.
- List manageable volumes and snapshots.
- Create, modify, delete, and list consistency groups.
- Create, modify, delete, and list snapshots of consistency groups.
- Create consistency group from consistency group or consistency group snapshot.
- Support consistency groups capability to generic volume groups.
- Attach a volume to multiple servers simultaneously (multiattach).

iSCSI driver

The NexentaStor appliance must be installed and configured according to the relevant Nexenta documentation. A pool and an enclosing namespace must be created for all iSCSI volumes to be accessed through the volume driver. This should be done as specified in the release-specific NexentaStor documentation.

The NexentaStor Appliance iSCSI driver is selected using the normal procedures for one or multiple back-end volume drivers.

You must configure these items for each NexentaStor appliance that the iSCSI volume driver controls:

1. Make the following changes on the volume node `/etc/cinder/cinder.conf` file.

```
# Enable Nexenta iSCSI driver
volume_driver=cinder.volume.drivers.nexenta.ns5.iscsi.NexentaISCSIDriver

# IP address of NexentaStor host (string value)
nexenta_host=HOST-IP

# Port for Rest API (integer value)
nexenta_rest_port=8443

# Username for NexentaStor Rest (string value)
nexenta_user=USERNAME

# Password for NexentaStor Rest (string value)
nexenta_password=PASSWORD

# Pool on NexentaStor appliance (string value)
nexenta_volume=volume_name

# Name of a parent Volume group where cinder created zvols will reside␣
↪(string value)
nexenta_volume_group = iscsi
```

Note

nexenta_volume represents a zpool, which is called pool on NS 5.x appliance. It must be pre-created before enabling the driver.

Volume group does not need to be pre-created, the driver will create it if does not exist.

2. Save the changes to the `/etc/cinder/cinder.conf` file and restart the `cinder-volume` service.

NFS driver

The Nexenta NFS driver allows you to use NexentaStor appliance to store Compute volumes via NFS. Every Compute volume is represented by a single NFS file within a shared directory.

While the NFS protocols standardize file access for users, they do not standardize administrative actions such as taking snapshots or replicating file systems. The OpenStack Volume Drivers bring a common interface to these operations. The Nexenta NFS driver implements these standard actions using the ZFS management plane that already is deployed on NexentaStor appliances.

The NexentaStor appliance must be installed and configured according to the relevant Nexenta documentation. A single-parent file system must be created for all virtual disk directories supported for OpenStack. Create and export the directory on each NexentaStor appliance.

You must configure these items for each NexentaStor appliance that the NFS volume driver controls:

1. Make the following changes on the volume node `/etc/cinder/cinder.conf` file.

```
# Enable Nexenta NFS driver
volume_driver=cinder.volume.drivers.nexenta.ns5.nfs.NexentaNfsDriver

```

(continues on next page)

(continued from previous page)

```
# IP address or Hostname of NexentaStor host (string value)
nas_host=HOST-IP

# Port for Rest API (integer value)
nexenta_rest_port=8443

# Path to parent filesystem (string value)
nas_share_path=POOL/FILESYSTEM

# Recommended NFS options
nas_mount_options=vers=3,minorversion=0,timeo=100,nolock
```

2. Create filesystem on appliance and share via NFS. For example:

```
"securityContexts": [
   {"readWriteList": [{"allow": true, "etype": "fqnip", "entity": "1.1.1.1
↪"}],
    "root": [{"allow": true, "etype": "fqnip", "entity": "1.1.1.1"}],
    "securityModes": ["sys"]}]
```

3. Create ACL for the filesystem. For example:

```
{"type": "allow",
"principal": "everyone@",
"permissions": ["list_directory","read_data","add_file","write_data",
"add_subdirectory","append_data","read_xattr","write_xattr","execute",
"delete_child","read_attributes","write_attributes","delete","read_acl",
"write_acl","write_owner","synchronize"],
"flags": ["file_inherit","dir_inherit"]}
```

Driver options

Nexenta Driver supports these options:

Table 60: Description of NexentaStor 5 driver configuration options

Configuration option = Default value	Description
[DEFAULT]	
nexenta_dataset_com = on	(String) Compression value for new ZFS folders.
nexenta_dataset_ded = off	(String) Deduplication value for new ZFS folders.
nexenta_dataset_des =	(String) Human-readable description for the folder.
nexenta_host =	(String) IP address of Nexenta SA
nexenta_iscsi_targe = 3260	(Integer) Nexenta target portal port
nexenta_mount_point = $state_path/mnt	(String) Base directory that contains NFS share mount points
nexenta_ns5_blocksi = 32	(Integer) Block size for datasets
nexenta_rest_port = 0	(Integer) HTTP(S) port to connect to Nexenta REST API server. If it is equal zero, 8443 for HTTPS and 8080 for HTTP is used
nexenta_rest_protoc = auto	(String) Use http or https for REST connection (default auto)
nexenta_sparse = False	(Boolean) Enables or disables the creation of sparse datasets
nexenta_sparsed_vol = True	(Boolean) Enables or disables the creation of volumes as sparsed files that take no space. If disabled (False), volume is created as a regular file, which takes a long time.
nexenta_use_https = True	(Boolean) Use secure HTTP for REST connection (default True)
nexenta_user = admin	(String) User name to connect to Nexenta SA
nexenta_volume = cinder	(String) SA Pool that holds all volumes
nexenta_volume_grou = iscsi	(String) Volume group for ns5

Nimble & Alletra 6k Storage volume driver

Nimble Storage fully integrates with the OpenStack platform through the Nimble Cinder driver, allowing a host to configure and manage Nimble and Alletra 6k Storage array features through Block Storage interfaces.

Support for iSCSI storage protocol is available with NimbleISCSIDriver Volume Driver class and Fibre Channel with NimbleFCDriver.

Support for the Liberty release and above is available from Nimble OS 2.3.8 or later.

Support for the Ocata release and above is available from Nimble OS 3.6 or later.

For Xena release, Nimble OS 5.3 or later is used and Alletra OS 6.0 or later is used.

Nimble and Alletra 6k Storage Cinder driver does not support port binding with multiple interfaces on the same subnet due to existing limitation in os-brick. This is partially referenced in the bug https://bugs.launchpad.net/os-brick/+bug/1722432 but does not resolve for multiple software iscsi ifaces.

Supported operations

- Create, delete, clone, attach, and detach volumes
- Create and delete volume snapshots
- Create a volume from a snapshot
- Copy an image to a volume
- Copy a volume to an image
- Extend a volume
- Get volume statistics
- Manage and unmanage a volume
- Enable encryption and default performance policy for a volume-type extra-specs
- Force backup of an in-use volume
- Retype a volume
- Create a Thinly Provisioned Volume
- Attach a volume to multiple servers simultaneously (multiattach)
- Volume Revert to Snapshot
- Create, list, update, and delete consistency groups
- Create, list, and delete consistency group snapshots
- Consistency group replication

Nimble and Alletra 6k Storage driver configuration

Update the file `/etc/cinder/cinder.conf` with the given configuration. Note: These parameters apply to Alletra 6k Storage as well.

In case of a basic (single back-end) configuration, add the parameters within the `[default]` section as follows.

```
[default]
san_ip = NIMBLE_MGMT_IP
san_login = NIMBLE_USER
san_password = NIMBLE_PASSWORD
use_multipath_for_image_xfer = True
volume_driver = NIMBLE_VOLUME_DRIVER
san_thin_provision = True
```

In case of multiple back-end configuration, for example, configuration which supports multiple Nimble Storage arrays or a single Nimble Storage array with arrays from other vendors, use the following parameters.

```
[default]
enabled_backends = Nimble-Cinder

[Nimble-Cinder]
san_ip = NIMBLE_MGMT_IP
san_login = NIMBLE_USER
san_password = NIMBLE_PASSWORD
use_multipath_for_image_xfer = True
volume_driver = NIMBLE_VOLUME_DRIVER
volume_backend_name = NIMBLE_BACKEND_NAME
```

In case of multiple back-end configuration, Nimble Storage volume type is created and associated with a back-end name as follows.

Note

Single back-end configuration users do not need to create the volume type.

```
$ openstack volume type create NIMBLE_VOLUME_TYPE
$ openstack volume type set --property volume_backend_name=NIMBLE_BACKEND_
↪NAME NIMBLE_VOLUME_TYPE
```

This section explains the variables used above:

NIMBLE_MGMT_IP
Management IP address of Nimble/Alletra 6k Storage array/group.

NIMBLE_USER
Nimble/Alletra 6k Storage account login with minimum `power user` (admin) privilege if RBAC is used.

NIMBLE_PASSWORD
Password of the admin account for Nimble/Alletra 6k array.

NIMBLE_VOLUME_DRIVER
Use either cinder.volume.drivers.hpe.nimble.NimbleISCSIDriver for iSCSI or cinder.volume.drivers.hpe.nimble.NimbleFCDriver for Fibre Channel.

NIMBLE_BACKEND_NAME
A volume back-end name which is specified in the `cinder.conf` file. This is also used while assigning a back-end name to the Nimble volume type.

NIMBLE_VOLUME_TYPE
The Nimble volume-type which is created from the CLI and associated with `NIMBLE_BACKEND_NAME`.

Note

Restart the `cinder-api`, `cinder-scheduler`, and `cinder-volume` services after updating the `cinder.conf` file.

Nimble driver extra spec options

The Nimble volume driver also supports the following extra spec options:

nimble:encryption=yes
Used to enable encryption for a volume-type.

nimble:perfpol-name=PERF_POL_NAME
PERF_POL_NAME is the name of a performance policy which exists on the Nimble/Alletra 6k array and should be enabled for every volume in a volume type.

Note

When upgrading to OpenStack deployment to Victoria or later, do unset `nimble:multi-initiator` extra-spec and set `multiattach='<is> True'`.

nimble:dedupe=true
Used to enable dedupe support for a volume-type.

nimble:iops-limit=IOPS_LIMIT
Used to set the IOPS_LIMIT between 256 and 4294967294 for all volumes created for this volume-type.

nimble:folder=FOLDER_NAME
FOLDER_NAME is the name of the folder which exists on the Nimble/Alletra 6k array and should be enabled for every volume in a volume type

These extra-specs can be enabled by using the following command:

```
$ openstack volume type set --property KEY=VALUE VOLUME_TYPE
```

`VOLUME_TYPE` is the Nimble volume type and `KEY` and `VALUE` are the options mentioned above.

Configuration options

The Nimble/Alletra 6k storage driver supports these configuration options:

Table 61: Description of Nimble configuration options

Configuration option = Default value	Description
`nimble_pool_name` = `default`	(String) Nimble Controller pool name
`nimble_subnet_label` = `*`	(String) Nimble Subnet Label
`nimble_verify_cert_path` = `None`	(String) Path to Nimble Array SSL certificate
`nimble_verify_certificate` = `False`	(Boolean) Whether to verify Nimble SSL Certificate

Multipathing

In OpenStack environments where Cinder block device multipathing is desired there are a few things to consider.

Configuring mulitpathing varies by system depending on the environment. In a scenario where solely Nimble devices are being created by Cinder, the following `/etc/multipath.conf` file may be used:

```
defaults {
    user_friendly_names yes
    find_multipaths     no
}

blacklist {
    devnode "^(ram|raw|loop|fd|md|dm-|sr|scd|st)[0-9]*"
    devnode "^hd[a-z]"
    device {
        vendor  ".*"
        product ".*"
    }
}

blacklist_exceptions {
    device {
        vendor  "Nimble"
        product "Server"
    }
}

devices {
    device {
        vendor               "Nimble"
        product              "Server"
        path_grouping_policy group_by_prio
        prio                 "alua"
        hardware_handler     "1 alua"
        path_selector        "service-time 0"
        path_checker         tur
        features             "1 queue_if_no_path"
        no_path_retry        30
        failback             immediate
        fast_io_fail_tmo     5
        dev_loss_tmo         infinity
        rr_min_io_rq         1
        rr_weight            uniform
    }
}
```

After making changes to `/etc/multipath.conf`, the multipath subsystem needs to be reconfigured:

```
# multipathd reconfigure
```

Tip

The latest best practices for Nimble devices can be found in the HPE Nimble Storage Linux Integration Guide found on https://infosight.hpe.com

Important

OpenStack Cinder is currently not compatible with the HPE Nimble Storage Linux Toolkit (NLT)

Nova needs to be configured to pickup the actual multipath device created on the host.

In `/etc/nova/nova.conf`, add the following to the `[libvirt]` section:

```
[libvirt]
volume_use_multipath = True
```

Note

In versions prior to Newton, the option was called `iscsi_use_multipath`

After editing the Nova configuration file, the `nova-conductor` service needs to be restarted.

Tip

Depending on which particular OpenStack distribution is being used, Nova may use a different configuration file than the default.

To validate that instances get properly connected to the multipath device, inspect the instance devices:

```
# virsh dumpxml <Instance ID | Instance Name | Instance UUID>
```

Consistency group replication

To enable consistency group replication, follow below steps:

1. Add *replication_device* to storage backend settings in *cinder.conf*, then restart Cinder Volume service.

   Example of *cinder.conf* for volume replications:

   ```
   [nimble]
   san_ip = xxx.xxx.xxx.xxx
   ...
   replication_device = backend_id:nimblevsagroup2,
                        san_ip:10.132.239.66,
                        san_login:admin,
                        san_password:admin,
                        schedule_name:sched-one,
                        downstream_partner:nimblevsagroup2,
                        period:15,
                        period_unit:minutes
   ```

   - Only one *replication_device* can be configured for each primary backend.
   - Keys *backend_id*, *san_ip*, *san_login*, *san_password*, *schedule_name* and *downstream_partner* are mandatory.

   - Other parameters are optional (if not given, then default values will be used): period:1 period_unit:days num_retain:10 num_retain_replica:1 at_time:00:00 until_time:23:59 days=all replicate_every:1 alert_threshold:24:00

2. Create a volume type with properties *replication_enabled=<is> True* and *consistent_group_snapshot_enabled=<is> True*

```
$ cinder type-create nimble
$ cinder type-key nimble set volume_backend_name='nimble'
$ cinder type-key nimble set replication_enabled='<is> True'
$ cinder type-key nimble set consistent_group_snapshot_enabled='<is> True'
```

3. Create a consistency group type with properties *consistent_group_snapshot_enabled=<is> True* and *consistent_group_replication_enabled=<is> True*.

```
$ cinder --os-volume-api-version 3.38 group-type-create repl_type
$ cinder --os-volume-api-version 3.38 group-type-key repl_type set
consistent_group_snapshot_enabled='<is> True' consistent_group_replication_enabled='<is> True'
```

4. Create a group type with volume types support replication.

```
$ cinder --os-volume-api-version 3.38 group-create --name grp_1 repl_type nimble
```

5. Create volume in the consistency group.

```
$ cinder --os-volume-api-version 3.38 create --volume-type nimble --group-id {grp_1-id}
--name {volume-name} {size}
```

6. Enable consistency group replication.

```
$ cinder --os-volume-api-version 3.38 group-enable-replication grp_1
```

7. Disable consistency group replication.

```
$ cinder --os-volume-api-version 3.38 group-disable-replication grp_1
```

8. Failover consistency group replication.

```
$ cinder --os-volume-api-version 3.38 group-failover-replication
--secondary-backend-id nimblevsagroup2 grp_1
```

9. Failback consistency group replication.

```
$ cinder --os-volume-api-version 3.38 group-failover-replication
--secondary-backend-id default grp_1
```

Open-E JovianDSS iSCSI driver

The `JovianISCSIDriver` allows usage of Open-E JovianDSS Data Storage Solution to be used as Block Storage in OpenStack deployments.

Supported operations

- Create, delete, attach, and detach volumes.
- Create, list, and delete volume snapshots.
- Create a volume from a snapshot.
- Copy an image to a volume.
- Copy a volume to an image.
- Clone a volume.
- Extend a volume.
- Migrate a volume with back-end assistance.

Configuring

Edit with your favourite editor Cinder config file. It can be found at /etc/cinder/cinder.conf

Add the field enabled_backends with value open-e-jdss-0:

```
enabled_backends = open-e-jdss-0
```

Provide settings to Open-E JovianDSS driver by adding open-e-jdss-0 description:

```
[open-e-jdss-0]
backend_name = Open-EJovianDSS
chap_password_len = 14
driver_use_ssl = True
driver_ssl_cert_verify = True
driver_ssl_cert_path = /etc/cinder/jdss.crt
iscsi_target_prefix = iqn.2016-04.com.open-e.cinder:
jovian_pool = Pool-0
jovian_block_size = 64K
san_api_port = 82
target_port = 3260
volume_driver = cinder.volume.drivers.open_e.iscsi.JovianISCSIDriver
san_hosts = 192.168.0.40
san_login = admin
san_password = admin
san_thin_provision = True
```

Table 62: **Open-E JovianDSS configuration options**

Option	Default value	Description
backend_name	Open-EJovianDSS	Name of the back end
chap_password_len	12	Length of the unique generated CHAP password.
driver_use_ssl	True	Use SSL to send requests to Open-E JovianDSS[1]
driver_ssl_cert_ve	True	Verify authenticity of Open-E JovianDSS[1] certificate
driver_ssl_cert_pa	None	Path to the Open-E JovianDSS[1] certificate for verification
iscsi_target_prefi	iqn.2016-04.com.open-e:01:cinder-	Prefix that will be used to form target name for volume
jovian_pool	Pool-0	Pool name that is going to be used. Must be created in [2]
jovian_block_size	64K	Block size for newly created volumes
san_api_port	82	Rest port according to the settings in [1]
target_port	3260	Port for iSCSI connections
volume_driver		Location of the driver source code
san_hosts		Comma separated list of IP address of the Open-E JovianDSS
san_login	admin	Must be set according to the settings in [1]
san_password	admin	Open-E Jovian DSS password [1], **should be changed**
san_thin_provision	False	Using thin provisioning for new volumes

1. Open-E JovianDSS Web interface/System Settings/REST Access
2. Pool can be created by going to Open-E JovianDSS Web interface/Storage

More info about Open-E JovianDSS

Multiple Pools

In order to add another Open-E JovianDSS Pool, create a copy of Open-E JovianDSS config in cinder.conf file.

For instance if you want to add `Pool-1` located on the same host as `Pool-0`. You extend `cinder.conf` file like:

```
enabled_backends = open-e-jdss-0, open-e-jdss-1

[open-e-jdss-0]
backend_name = open-e-jdss-0
chap_password_len = 14
driver_use_ssl = True
driver_ssl_cert_verify = False
iscsi_target_prefix = iqn.2016-04.com.open-e.cinder:
jovian_pool = Pool-0
jovian_block_size = 64K
san_api_port = 82
```

(continues on next page)

(continued from previous page)

```
target_port = 3260
volume_driver = cinder.volume.drivers.open_e.iscsi.JovianISCSIDriver
san_hosts = 192.168.0.40
san_login = admin
san_password = admin
san_thin_provision = True

[open-e-jdss-1]
backend_name = open-e-jdss-1
chap_password_len = 14
driver_use_ssl = True
driver_ssl_cert_verify = False
iscsi_target_prefix = iqn.2016-04.com.open-e.cinder:
jovian_pool = Pool-1
jovian_block_size = 64K
san_api_port = 82
target_port = 3260
volume_driver = cinder.volume.drivers.open_e.iscsi.JovianISCSIDriver
san_hosts = 192.168.0.50
san_login = admin
san_password = admin
san_thin_provision = True
```

HA Cluster

To utilize High Availability feature of Open-E JovianDSS:

1. Guide on configuring Pool to high availability cluster
2. Set `jovian_hosts` with list of `virtual IPs` associated with this Pool

For instance if you have `Pool-2` with 2 virtual IPs 192.168.21.100 and 192.168.31.100 the configuration file will look like:

```
[open-e-jdss-2]
backend_name = open-e-jdss-2
chap_password_len = 14
driver_use_ssl = True
driver_ssl_cert_verify = False
iscsi_target_prefix = iqn.2016-04.com.open-e.cinder:
jovian_pool = Pool-0
jovian_block_size = 64K
san_api_port = 82
target_port = 3260
volume_driver = cinder.volume.drivers.open_e.iscsi.JovianISCSIDriver
san_hosts = 192.168.21.100, 192.168.31.100
san_login = admin
san_password = admin
san_thin_provision = True
```

Feedback

Please address problems and proposals to andrei.perepiolkin@open-e.com

ProphetStor Fibre Channel and iSCSI drivers

ProhetStor Fibre Channel and iSCSI drivers add support for ProphetStor Flexvisor through the Block Storage service. ProphetStor Flexvisor enables commodity x86 hardware as software-defined storage leveraging well-proven ZFS for disk management to provide enterprise grade storage services such as snapshots, data protection with different RAID levels, replication, and deduplication.

The `DPLFCDriver` and `DPLISCSIDriver` drivers run volume operations by communicating with the ProphetStor storage system over HTTPS.

Supported operations

- Create, delete, attach, and detach volumes.
- Create, list, and delete volume snapshots.
- Create a volume from a snapshot.
- Copy an image to a volume.
- Copy a volume to an image.
- Clone a volume.
- Extend a volume.

Enable the Fibre Channel or iSCSI drivers

The `DPLFCDriver` and `DPLISCSIDriver` are installed with the OpenStack software.

1. Query storage pool id to configure `dpl_pool` of the `cinder.conf` file.

   a. Log on to the storage system with administrator access.

   ```
   $ ssh root@STORAGE_IP_ADDRESS
   ```

   b. View the current usable pool id.

   ```
   $ flvcli show pool list
   - d5bd40b58ea84e9da09dcf25a01fdc07 : default_pool_dc07
   ```

   c. Use `d5bd40b58ea84e9da09dcf25a01fdc07` to configure the `dpl_pool` of `/etc/cinder/cinder.conf` file.

   Note

   Other management commands can be referenced with the help command **`flvcli -h`**.

2. Make the following changes on the volume node `/etc/cinder/cinder.conf` file.

```
# IP address of SAN controller (string value)
san_ip=STORAGE IP ADDRESS

# Username for SAN controller (string value)
san_login=USERNAME

# Password for SAN controller (string value)
san_password=PASSWORD

# Use thin provisioning for SAN volumes? (boolean value)
san_thin_provision=true

# The port that the iSCSI daemon is listening on. (integer value)
iscsi_port=3260

# DPL pool uuid in which DPL volumes are stored. (string value)
dpl_pool=d5bd40b58ea84e9da09dcf25a01fdc07

# DPL port number. (integer value)
dpl_port=8357

# Uncomment one of the next two option to enable Fibre channel or iSCSI
# FIBRE CHANNEL(uncomment the next line to enable the FC driver)
#volume_driver=cinder.volume.drivers.prophetstor.dpl_fc.DPLFCDriver
# iSCSI (uncomment the next line to enable the iSCSI driver)
#volume_driver=cinder.volume.drivers.prophetstor.dpl_iscsi.DPLISCSIDriver
```

3. Save the changes to the `/etc/cinder/cinder.conf` file and restart the `cinder-volume` service.

The ProphetStor Fibre Channel or iSCSI drivers are now enabled on your OpenStack system. If you experience problems, review the Block Storage service log files for errors.

The following table contains the options supported by the ProphetStor storage driver.

Table 63: Description of ProphetStor Fibre Channel and iSCSi drivers configuration options

Configuration option = Default value	Description
[DEFAULT]	
`dpl_pool` =	(String) DPL pool uuid in which DPL volumes are stored.
`dpl_port` = `8357`	(Port number) DPL port number.
`iscsi_port` = `3260`	(Port number) The port that the iSCSI daemon is listening on
`san_ip` =	(String) IP address of SAN controller
`san_login` = `admin`	(String) Username for SAN controller
`san_password` =	(String) Password for SAN controller
`san_thin_provision` = `True`	(Boolean) Use thin provisioning for SAN volumes?

Pure Storage iSCSI, Fibre Channel and NVMe volume drivers

The Pure Storage FlashArray volume drivers for OpenStack Block Storage interact with configured Pure Storage arrays and support various operations.

Support for iSCSI storage protocol is available with the PureISCSIDriver Volume Driver class, Fibre Channel with the PureFCDriver and NVMe-ROCE or NVMe-TCP with the PureNVMEDriver.

iSCSI, Fibre Channel and NVMe-RoCE drivers are compatible with FlashArrays that support the REST API version 2.4 and higher (Purity 6.1.0 and newer). The NVMe-TCP driver is compatible with FlashArrays that are running Purity 6.4.2 and higher. Some features may require newer versions of Purity.

Limitations and known issues

If you do not set up the nodes hosting instances to use multipathing, all network connectivity will use a single physical port on the array. In addition to significantly limiting the available bandwidth, this means you do not have the high-availability and non-disruptive upgrade benefits provided by FlashArray. Multipathing must be used to take advantage of these benefits.

Supported operations

- Create, delete, attach, detach, retype, clone, and extend volumes.
- Create a volume from snapshot.
- Create, list, and delete volume snapshots.
- Create, list, update, and delete consistency groups.
- Create, list, and delete consistency group snapshots.
- Revert a volume to a snapshot.
- Manage and unmanage a volume.
- Manage and unmanage a snapshot.
- Get volume statistics.
- Create a thin provisioned volume.
- Replicate volumes to remote Pure Storage array(s)

QoS support for the Pure Storage drivers include the ability to set the following capabilities in the OpenStack Block Storage API `cinder.api.contrib.qos_spec_manage` qos specs extension module:

- **maxIOPS** - Maximum number of IOPs allowed for volume. Range: 100 - 100M
- **maxBWS** - Maximum bandwidth limit in MB/s. Range: 1 - 524288 (512GB/s)

The qos keys above must be created and asscoiated to a volume type. For information on how to set the key-value pairs and associate them with a volume type see the volume qos section in the OpenStack Client command list.

Configure OpenStack and Purity

You need to configure both your Purity array and your OpenStack cluster.

Note

These instructions assume that the `cinder-api` and `cinder-scheduler` services are installed and configured in your OpenStack cluster.

Configure the OpenStack Block Storage service

In these steps, you will edit the `cinder.conf` file to configure the OpenStack Block Storage service to enable multipathing and to use the Pure Storage FlashArray as back-end storage.

1. Install Pure Storage PyPI module. A requirement for the Pure Storage driver is the installation of the Pure Storage Python SDK version 1.47.0 or later from PyPI.

   ```
   $ pip install py-pure-client
   ```

2. Retrieve an API token from Purity. The OpenStack Block Storage service configuration requires an API token from Purity. Actions performed by the volume driver use this token for authorization. Also, Purity logs the volume drivers actions as being performed by the user who owns this API token.

   If you created a Purity user account that is dedicated to managing your OpenStack Block Storage volumes, copy the API token from that user account.

   Use the appropriate create or list command below to display and copy the Purity API token:

   - To create a new API token:

     ```
     $ pureadmin create --api-token USER
     ```

     The following is an example output:

     ```
     $ pureadmin create --api-token pureuser
     Name      API Token                              Created
     pureuser  902fdca3-7e3f-d2e4-d6a6-24c2285fe1d9  2014-08-04 14:50:30
     ```

   - To list an existing API token:

     ```
     $ pureadmin list --api-token --expose USER
     ```

     The following is an example output:

     ```
     $ pureadmin list --api-token --expose pureuser
     Name      API Token                              Created
     pureuser  902fdca3-7e3f-d2e4-d6a6-24c2285fe1d9  2014-08-04 14:50:30
     ```

3. Copy the API token retrieved (`902fdca3-7e3f-d2e4-d6a6-24c2285fe1d9` from the examples above) to use in the next step.

4. Edit the OpenStack Block Storage service configuration file. The following sample `/etc/cinder/cinder.conf` configuration lists the relevant settings for a typical Block Storage service using a single Pure Storage array:

   ```
   [DEFAULT]
   enabled_backends = puredriver-1
   ```

(continues on next page)

(continued from previous page)

```
default_volume_type = puredriver-1

[puredriver-1]
volume_backend_name = puredriver-1
volume_driver = PURE_VOLUME_DRIVER
san_ip = IP_PURE_MGMT
pure_api_token = PURE_API_TOKEN
use_multipath_for_image_xfer = True
```

Replace the following variables accordingly:

PURE_VOLUME_DRIVER
Use `cinder.volume.drivers.pure.PureISCSIDriver` for iSCSI, `cinder.volume.drivers.pure.PureFCDriver` for Fibre Channel or `cinder.volume.drivers.pure.PureNVMEDriver` for NVME connectivity.

If using the NVME driver, specify the `pure_nvme_transport` value. Supported values are `roce` or `tcp`.

IP_PURE_MGMT
The IP address of the Pure Storage arrays management interface or a domain name that resolves to that IP address.

PURE_API_TOKEN
The Purity Authorization token that the volume driver uses to perform volume management on the Pure Storage array.

Note

The volume driver automatically creates Purity host objects for initiators as needed. If CHAP authentication is enabled via the `use_chap_auth` setting, you must ensure there are no manually created host objects with IQNs that will be used by the OpenStack Block Storage service. The driver will only modify credentials on hosts that it manages.

Note

If using the PureFCDriver it is recommended to use the OpenStack Block Storage Fibre Channel Zone Manager.

Volume auto-eradication

To enable auto-eradication of deleted volumes, snapshots, and consistency groups on deletion, modify the following option in the `cinder.conf` file:

```
pure_eradicate_on_delete = true
```

By default, auto-eradication is disabled and all deleted volumes, snapshots, and consistency groups are retained on the Pure Storage array in a recoverable state for 24 hours from time of deletion.

Setting host personality

The host personality determines how the Purity system tunes the protocol used between the array and the initiator. To ensure the array works optimally with the host, set the personality to the name of the host operating or virtual memory system. Valid values are aix, esxi, hitachi-vsp, hpux, oracle-vm-server, solaris, and vms. If your system is not listed as one of the valid host personalities, do not set the option. By default, the host personality is not set.

To set the host personality, modify the following option in the `cinder.conf` file:

```
pure_host_personality = <personality>
```

Note

`pure_host_personality` is available from Purity REST API version 1.14, and affects only newly-created hosts.

SSL certification

To enable SSL certificate validation, modify the following option in the `cinder.conf` file:

```
driver_ssl_cert_verify = true
```

By default, SSL certificate validation is disabled.

To specify a non-default path to `CA_Bundle` file or directory with certificates of trusted CAs:

```
driver_ssl_cert_path = Certificate path
```

Replication configuration

Add the following to the back-end specification to specify another Flash Array to replicate to:

```
[puredriver-1]
replication_device = backend_id:PURE2_NAME,san_ip:IP_PURE2_MGMT,api_
↪token:PURE2_API_TOKEN,type:REPLICATION_TYPE
```

Where `PURE2_NAME` is the name of the remote Pure Storage system, `IP_PURE2_MGMT` is the management IP address of the remote array, and `PURE2_API_TOKEN` is the Purity Authorization token of the remote array.

The `REPLICATION_TYPE` value for the `type` key can be either `sync` or `async`

If the `type` is `sync` volumes will be created in a stretched Pod. This requires two arrays pre-configured with Active Cluster enabled. You can optionally specify `uniform` as `true` or `false`, this will instruct the driver that data paths are uniform between arrays in the cluster and data connections should be made to both upon attaching.

Note that more than one `replication_device` line can be added to allow for multi-target device replication.

To enable 3-site replication, ie. a volume that is synchronously replicated to one array and also asynchronously replicated to another then you must supply two, and only two, `replication_device` lines, where one has `type` of `sync` and one where `type` is `async`. Additionally, the parameter `pure_trisync_enabled` must be set `True`.

A volume is only replicated if the volume is of a volume-type that has the extra spec `replication_enabled` set to `<is> True`. You can optionally specify the `replication_type` key to specify `<in> sync` or `<in> async` or `<in> trisync` to choose the type of replication for that volume. If not specified it will default to `async`.

To create a volume type that specifies replication to remote back ends with async replication:

```
$ openstack volume type create ReplicationType
$ openstack volume type set --property replication_enabled='<is> True' ReplicationType
$ openstack volume type set --property replication_type='<in> async' ReplicationType
```

The following table contains the optional configuration parameters available for async replication configuration with the Pure Storage array.

Table 64: Pure Storage replication configuration options

Option	Description	Default
`pure_replica_interval_defaul`	Snapshot replication interval in seconds.	`3600`
`pure_replica_retention_short`	Retain all snapshots on target for this time (in seconds).	`14400`
`pure_replica_retention_long_`	Retain how many snapshots for each day.	`3`
`pure_replica_retention_long_`	Retain snapshots per day on target for this time (in days).	`7`
`pure_replication_pg_name`	Pure Protection Group name to use for async replication (will be created if it does not exist).	`cinder-group`
`pure_replication_pod_name`	Pure Pod name to use for sync replication (will be created if it does not exist).	`cinder-pod`

Note

`failover-host` is only supported from the primary array to any of the multiple secondary arrays, but subsequent `failover-host` is only supported back to the original primary array.

Note

`pure_replication_pg_name` and `pure_replication_pod_name` should not be changed after volumes have been created in the Cinder backend, as this could have unexpected results in both replication and failover.

Automatic thin-provisioning/oversubscription ratio

This feature allows the driver to calculate the array oversubscription ratio as (total provisioned/actual used). By default this feature is enabled.

To disable this feature and honor the hard-coded configuration option `max_over_subscription_ratio` add the following option in the `cinder.conf` file:

```
[puredriver-1]
pure_automatic_max_oversubscription_ratio = False
```

Note

Arrays with very good data reduction rates (compression/data deduplication/thin provisioning) can get *very* large oversubscription rates applied.

Scheduling metrics

A large number of metrics are reported by the volume driver which can be useful in implementing more control over volume placement in multi-backend environments using the driver filter and weighter methods.

Metrics reported include, but are not limited to:

```
total_capacity_gb
free_capacity_gb
provisioned_capacity
total_volumes
total_snapshots
total_hosts
total_pgroups
writes_per_sec
reads_per_sec
input_per_sec
output_per_sec
usec_per_read_op
usec_per_read_op
queue_depth
replication_type
```

Note

All total metrics include non-OpenStack managed objects on the array.

In conjunction with QOS extra-specs, you can create very complex algorithms to manage volume placement. More detailed documentation on this is available in other external documentation.

Configuration Options

The following list all Pure driver specific configuration options that can be set in *cinder.conf*:

Table 65: Description of Pure configuration options

Configuration option = Default value	Description
pure_api_token = None	(String) REST API authorization token.
pure_automatic_ = True	(Boolean) Automatically determine an oversubscription ratio based on the current total data reduction values. If used this calculated value will override the max_over_subscription_ratio config option.
pure_eradicate_ = False	(Boolean) When enabled, all Pure volumes, snapshots, and protection groups will be eradicated at the time of deletion in Cinder. Data will NOT be recoverable after a delete with this set to True! When disabled, volumes and snapshots will go into pending eradication state and can be recovered.
pure_host_perso = None	(String(choices=[aix, esxi, hitachi-vsp, hpux, oracle-vm-server, solaris, vms, None])) Determines how the Purity system tunes the protocol used between the array and the initiator.
pure_iscsi_cidr = 0.0.0.0/0	(String) CIDR of FlashArray iSCSI targets hosts are allowed to connect to. Default will allow connection to any IPv4 address. This parameter now supports IPv6 subnets. Ignored when pure_iscsi_cidr_list is set.
pure_iscsi_cidr = None	(List of String) Comma-separated list of CIDR of FlashArray iSCSI targets hosts are allowed to connect to. It supports IPv4 and IPv6 subnets. This parameter supersedes pure_iscsi_cidr.
pure_nvme_cidr = 0.0.0.0/0	(String) CIDR of FlashArray NVMe targets hosts are allowed to connect to. Default will allow connection to any IPv4 address. This parameter now supports IPv6 subnets. Ignored when pure_nvme_cidr_list is set.
pure_nvme_cidr_ = None	(List of String) Comma-separated list of CIDR of FlashArray NVMe targets hosts are allowed to connect to. It supports IPv4 and IPv6 subnets. This parameter supersedes pure_nvme_cidr.
pure_nvme_trans = roce	(String(choices=[roce, tcp])) The NVMe transport layer to be used by the NVMe driver.
pure_replica_in = 3600	(Integer) Snapshot replication interval in seconds.
pure_replica_re = 7	(Integer) Retain snapshots per day on target for this time (in days.)
pure_replica_re = 3	(Integer) Retain how many snapshots for each day.
pure_replica_re = 14400	(Integer) Retain all snapshots on target for this time (in seconds.)
pure_replicatio = cinder-group	(String) Pure Protection Group name to use for async replication (will be created if it does not exist).
pure_replicatio = cinder-pod	(String) Pure Pod name to use for sync replication (will be created if it does not exist).
pure_trisync_en = False	(Boolean) When enabled and two replication devices are provided, one each of types sync and async, this will enable the ability to create a volume that is sync replicated to one array and async replicated to a separate array.
pure_trisync_pg = cinder-trisync	(String) Pure Protection Group name to use for trisync replication leg inside the sync replication pod (will be created if it does not exist).

Quobyte driver

The Quobyte volume driver enables storing Block Storage service volumes on a Quobyte storage back end. Block Storage service back ends are mapped to Quobyte volumes and individual Block Storage service volumes are stored as files on a Quobyte volume. Selection of the appropriate Quobyte volume is done by the aforementioned back end configuration that specifies the Quobyte volume explicitly.

Note

Note the dual use of the term `volume` in the context of Block Storage service volumes and in the context of Quobyte volumes.

For more information see the Quobyte support webpage.

Supported operations

The Quobyte volume driver supports the following volume operations:

- Create, delete, attach, and detach volumes
- Secure NAS operation (Starting with Mitaka release secure NAS operation is optional but still default)
- Create and delete a snapshot
- Create a volume from a snapshot
- Extend a volume
- Clone a volume
- Copy a volume to image
- Generic volume migration (no back end optimization)

Note

When running VM instances off Quobyte volumes, ensure that the Quobyte Compute service driver has been configured in your OpenStack cloud.

Configuration

To activate the Quobyte volume driver, configure the corresponding `volume_driver` parameter:

```
volume_driver = cinder.volume.drivers.quobyte.QuobyteDriver
```

The following table contains the configuration options supported by the Quobyte driver:

Table 66: Description of Quobyte USP configuration options

Configuration option = Default value	Description
quobyte_cli = None	(String) Path to a Quobyte Client configuration file.
quobyte_mou = $state_path mnt	(String) Base dir containing the mount point for the Quobyte volume.
quobyte_ove = False	(Boolean) Create new volumes from the volume_from_snapshot_cache by creating overlay files instead of full copies. This speeds up the creation of volumes from this cache. This feature requires the options quobyte_qcow2_volumes and quobyte_volume_from_snapshot_cache to be set to True. If one of these is set to False this option is ignored.
quobyte_qco = True	(Boolean) Create volumes as QCOW2 files rather than raw files.
quobyte_spa = True	(Boolean) Create volumes as sparse files which take no space. If set to False, volume is created as regular file.
quobyte_vol = False	(Boolean) Create a cache of volumes from merged snapshots to speed up creation of multiple volumes from a single snapshot.
quobyte_vol = None	(String) Quobyte URL to the Quobyte volume using e.g. a DNS SRV record (preferred) or a host list (alternatively) like quobyte://<DIR host1>, <DIR host2>/<volume name>

SandStone iSCSI Driver

SandStone USP volume can be used as a block storage resource in the OpenStack Block Storage driver that supports iSCSI protocols.

Before to go, you should have installed SandStoneUSP.

System requirements

Cluster	version
SandStone USP	3.2.3+

To use the SandStone driver, the following are required:

- Network connectivity between the OpenStack host and the SandStone USP management interfaces.
- HTTPS or HTTP must be enabled on the array.

When creating a volume from image, add the following configuration keys in the `[DEFAULT]` configuration group of the `/etc/cinder/cinder.conf` file:

Configuration example

The following table contains the configuration options supported by the SandStone driver.

```
[DEFAULT]
enabled_backends = sds-iscsi

[sds-iscsi]
volume_driver = cinder.volume.drivers.sandstone.sds_driver.SdsISCSIDriver
volume_backend_name = sds-iscsi
san_ip = 10.10.16.21
san_login = admin
san_password = admin
default_sandstone_target_ips = 10.10.16.21,10.10.16.22,10.10.16.23
chap_username = 123456789123
chap_password = 1234567891234
sandstone_pool = vms
initiator_assign_sandstone_target_ip = {"iqn.1993-08.org.debian:01:3a9cd5c484a
↪": "10.10.16.21"}
```

General parameters

Parameter	Description
volume_driver	Indicates the loaded driver
volume_backend_name	Indicates the name of the backend
san_ip	IP addresses of the management interfaces of SandStone USP
san_login	Storage system user name
san_password	Storage system password
default_sandstone _target_ips	Default IP address of the iSCSI target port that is provided for compute nodes
chap_username	CHAP authentication username
chap_password	CHAP authentication password
sandstone_pool	SandStone storage pool resource name
initiator_assign _sandstone_target_ip	Initiator assign target with assign ip

1. After modifying the `cinder.conf` file, restart the `cinder-volume` service.
2. Create and use volume types.

   Create and use sds-iscsi volume types

   ```
   $ openstack volume type create sandstone
   $ openstack volume type set --property volume_backend_name=sds-iscsi␣
   ↪sandstone
   ```

Seagate Array Fibre Channel and iSCSI drivers

The `STXFCDriver` and `STXISCSIDriver` Cinder drivers allow the Seagate Technology (STX) storage arrays to be used for Block Storage in OpenStack deployments.

System requirements

To use the Seagate drivers, the following are required:

- Seagate storage array with:
  - iSCSI or FC host interfaces
  - G28x firmware or later
- Network connectivity between the OpenStack host and the array management interfaces
- The HTTPS or HTTP protocol must be enabled on the array

Supported operations

- Create, delete, attach, and detach volumes.
- Create, list, and delete volume snapshots.
- Create a volume from a snapshot.
- Copy an image to a volume.
- Copy a volume to an image.
- Clone a volume.
- Extend a volume.
- Migrate a volume with back-end assistance.
- Retype a volume.
- Manage and unmanage a volume.

Configuring the array

1. Verify that the array can be managed via an HTTPS connection. HTTP can also be used if `driver_use_ssl` is set to (or defaults to) False in the `cinder.conf` file.

   Confirm that virtual pools A and B are present if you plan to use virtual pools for OpenStack storage.

   If you plan to use vdisks instead of virtual pools, create or identify one or more vdisks to be used for OpenStack storage; typically this will mean creating or setting aside one disk group for each of the A and B controllers.

2. Edit the `cinder.conf` file to define a storage back-end entry for each storage pool on the array that will be managed by OpenStack. Each entry consists of a unique section name, surrounded by square brackets, followed by options specified in a `key=value` format.

   - The `seagate_pool_name` value specifies the name of the storage pool or vdisk on the array.

- The `volume_backend_name` option value can be a unique value, if you wish to be able to assign volumes to a specific storage pool on the array, or a name that is shared among multiple storage pools to let the volume scheduler choose where new volumes are allocated.

3. The following `cinder.conf` options generally have identical values for each backend section on the array:

   - `volume_driver` specifies the Cinder driver name.
   - `san_ip` specifies the IP addresses or host names of the arrays management controllers.
   - `san_login` and `san_password` specify the username and password of an array user account with `manage` privileges
   - `driver_use_ssl` must be set to True to enable use of the HTTPS protocol.
   - `seagate_iscsi_ips` specifies the iSCSI IP addresses for the array if using the iSCSI transport protocol

   In the examples below, two back ends are defined, one for pool A and one for pool B, and a common `volume_backend_name` is used so that a single volume type definition can be used to allocate volumes from both pools.

   iSCSI example back-end entries

```
[pool-a]
seagate_pool_name = A
volume_backend_name = seagate-array
volume_driver = cinder.volume.drivers.stx.iscsi.STXISCSIDriver
san_ip = 10.1.2.3,10.1.2.4
san_login = manage
san_password = !manage
seagate_iscsi_ips = 10.2.3.4,10.2.3.5
driver_use_ssl = true

[pool-b]
seagate_backend_name = B
volume_backend_name = seagate-array
volume_driver = cinder.volume.drivers.stx.iscsi.STXISCSIDriver
san_ip = 10.1.2.3,10.1.2.4
san_login = manage
san_password = !manage
seagate_iscsi_ips = 10.2.3.4,10.2.3.5
driver_use_ssl = true
```

   Fibre Channel example back-end entries

```
[pool-a]
seagate_backend_name = A
volume_backend_name = seagate-array
volume_driver = cinder.volume.drivers.stx.fc.STXFCDriver
san_ip = 10.1.2.3,10.1.2.4
san_login = manage
san_password = !manage
driver_use_ssl = true
```

(continues on next page)

(continued from previous page)

```
[pool-b]
seagate_backend_name = B
volume_backend_name = seagate-array
volume_driver = cinder.volume.drivers.stx.fc.STXFCDriver
san_ip = 10.1.2.3,10.1.2.4
san_login = manage
san_password = !manage
driver_use_ssl = true
```

4. If any `volume_backend_name` value refers to a vdisk rather than a virtual pool, add an additional statement `seagate_backend_type = linear` to that back-end entry.

5. If HTTPS is enabled, you can enable certificate verification with the option `driver_ssl_cert_verify = True`. You may also use the `driver_ssl_cert_path` parameter to specify the path to a CA_BUNDLE file containing CAs other than those in the default list.

6. Modify the `[DEFAULT]` section of the `cinder.conf` file to add an `enabled_backends` parameter specifying the backend entries you added, and a `default_volume_type` parameter specifying the name of a volume type that you will create in the next step.

   Example of [DEFAULT] section changes

   ```
   [DEFAULT]
   enabled_backends = pool-a,pool-b
   default_volume_type = seagate
   ```

7. Create a new volume type for each distinct `volume_backend_name` value that you added in the `cinder.conf` file. The example below assumes that the same `volume_backend_name=seagate-array` option was specified in all of the entries, and specifies that the volume type `seagate` can be used to allocate volumes from any of them.

   Example of creating a volume type

   ```
   $ openstack volume type create seagate
   $ openstack volume type set --property volume_backend_name=seagate-array␣
   ↪seagate
   ```

8. After modifying the `cinder.conf` file, restart the `cinder-volume` service.

Driver-specific options

The following table contains the configuration options that are specific to the Seagate drivers.

Table 67: Description of Seagate configuration options

Configuration option = Default value	Description
`seagate_iscsi_ips` = `[]`	(List of String) List of comma-separated target iSCSI IP addresses.
`seagate_pool_name` = `A`	(String) Pool or vdisk name to use for volume creation.
`seagate_pool_type` = `virtual`	(String(choices=[linear, virtual])) linear (for vdisk) or virtual (for virtual pool).

SolidFire

The SolidFire Cluster is a high performance all SSD iSCSI storage device that provides massive scale out capability and extreme fault tolerance. A key feature of the SolidFire cluster is the ability to set and modify during operation specific QoS levels on a volume for volume basis. The SolidFire cluster offers this along with de-duplication, compression, and an architecture that takes full advantage of SSDs.

To configure the use of a SolidFire cluster with Block Storage, modify your `cinder.conf` file as follows:

```
volume_driver = cinder.volume.drivers.solidfire.SolidFireDriver
san_ip = 172.17.1.182         # the address of your MVIP
san_login = sfadmin           # your cluster admin login
san_password = sfpassword     # your cluster admin password
sf_account_prefix = ''        # prefix for tenant account creation on␣
↪solidfire cluster
```

Warning

Older versions of the SolidFire driver (prior to Icehouse) created a unique account prefixed with `$cinder-volume-service-hostname-$tenant-id` on the SolidFire cluster for each tenant. Unfortunately, this account formation resulted in issues for High Availability (HA) installations and installations where the `cinder-volume` service can move to a new node. The current default implementation does not experience this issue as no prefix is used. For installations created on a prior release, the OLD default behavior can be configured by using the keyword `hostname` in sf_account_prefix.

Note

The SolidFire driver creates names for volumes on the back end using the format UUID-<cinder-id>. This works well, but there is a possibility of a UUID collision for users running multiple clouds against the same cluster. In Mitaka the ability was added to eliminate the possibility of collisions by introducing the **sf_volume_prefix** configuration variable. On the SolidFire cluster each volume will be labeled with the prefix, providing the ability to configure unique volume names for each cloud. The default prefix is UUID-.

Changing the setting on an existing deployment will result in the existing volumes being inaccessible. To introduce this change to an existing deployment it is recommended to add the Cluster as if it were a second backend and disable new deployments to the current back end.

Table 68: Description of SolidFire configuration options

Configuration option = Default value	Description
sf_account_pref = None	(String) Create SolidFire accounts with this prefix. Any string can be used here, but the string hostname is special and will create a prefix using the cinder node hostname (previous default behavior). The default is NO prefix.
sf_allow_tenant = False	(Boolean) Allow tenants to specify QOS on create
sf_api_port = 443	(Port(min=0, max=65535)) SolidFire API port. Useful if the device api is behind a proxy on a different port.
sf_api_request_ = 30	(Integer(min=30)) Sets time in seconds to wait for an api request to complete.
sf_cluster_pair = 60	(Integer(min=3)) Sets time in seconds to wait for clusters to complete pairing.
sf_emulate_512 = True	(Boolean) Set 512 byte emulation on volume creation;
sf_enable_vag = False	(Boolean) Utilize volume access groups on a per-tenant basis.
sf_provisioning = maxProvisionedS	(String(choices=[maxProvisionedSpace, usedSpace])) Change how SolidFire reports used space and provisioning calculations. If this parameter is set to usedSpace, the driver will report correct values as expected by Cinder thin provisioning.
sf_svip = None	(String) Overrides default cluster SVIP with the one specified. This is required or deployments that have implemented the use of VLANs for iSCSI networks in their cloud.
sf_volume_clone = 600	(Integer(min=60)) Sets time in seconds to wait for a clone of a volume or snapshot to complete.
sf_volume_creat = 60	(Integer(min=30)) Sets time in seconds to wait for a create volume operation to complete.
sf_volume_pairi = 3600	(Integer(min=30)) Sets time in seconds to wait for a migrating volume to complete pairing and sync.
sf_volume_prefi = UUID-	(String) Create SolidFire volumes with this prefix. Volume names are of the form <sf_volume_prefix><cinder-volume-id>. The default is to use a prefix of UUID-.

Supported operations

- Create, delete, attach, and detach volumes.
- Create, list, and delete volume snapshots.
- Create a volume from a snapshot.
- Copy an image to a volume.
- Copy a volume to an image.
- Clone a volume.
- Extend a volume.
- Retype a volume.

- Manage and unmanage a volume.
- Consistency group snapshots.

QoS support for the SolidFire drivers includes the ability to set the following capabilities in the OpenStack Block Storage API `cinder.api.contrib.qos_specs_manage` qos specs extension module:

- **minIOPS** - The minimum number of IOPS guaranteed for this volume. Default = 100.
- **maxIOPS** - The maximum number of IOPS allowed for this volume. Default = 15,000.
- **burstIOPS** - The maximum number of IOPS allowed over a short period of time. Default = 15,000.
- **scaledIOPS** - The presence of this key is a flag indicating that the above IOPS should be scaled by the following scale values. It is recommended to set the value of scaledIOPS to True, but any value will work. The absence of this key implies false.
- **scaleMin** - The amount to scale the minIOPS by for every 1GB of additional volume size. The value must be an integer.
- **scaleMax** - The amount to scale the maxIOPS by for every 1GB of additional volume size. The value must be an integer.
- **scaleBurst** - The amount to scale the burstIOPS by for every 1GB of additional volume size. The value must be an integer.

The QoS keys above no longer require to be scoped but must be created and associated to a volume type. For information about how to set the key-value pairs and associate them with a volume type, see the volume qos section in the OpenStackClient command list.

> **Note**
>
> When using scaledIOPS, the scale values must be chosen such that the constraint minIOPS <= maxIOPS <= burstIOPS is always true. The driver will enforce this constraint.

Storage Performance Development Kit driver

Storage Performance Development Kit (SPDK) is a user space, polled-mode, asynchronous, lockless NVMe driver. It provides zero-copy, highly parallel access directly to an SSD from a user space application. SPDK provides NVMe-oF target that is capable of serving disks over the network or to other processes.

Preparation

SPDK NVMe-oF target installation

Follow instructions available on https://spdk.io/doc/nvmf.html to install and configure environment with SPDK NVMe-oF target application. Starting from Ussuri release SPDK release v19.10 or higher is required.

Storage pools configuration

SPDK Cinder driver requires storage pools to be configured upfront in SPDK NVMe-oF target application. SPDK driver uses Logical Volume Stores (LVS) as storage pools. Details on configuring LVS are available on https://spdk.io/doc/logical_volumes.html. After storage pools are configured remote access has to be enabled. Launch `scripts/rpc_http_proxy.py` script from SPDK directory to start an http server that will manage requests from volume driver.

Supported operations

- Create, delete, attach, and detach volumes.
- Create, list, and delete volume snapshots.
- Create a volume from a snapshot.
- Copy an image to a volume.
- Copy a volume to an image.
- Clone a volume.
- Extend a volume.
- Get volume statistics.

Configuration

Use the following options to configure for the SPDK NVMe-oF transport:

```
volume_driver = cinder.volume.drivers.spdk.SPDKDriver
target_protocol = nvmet_rdma          # SPDK driver supports only nvmet_rdma␣
↪target protocol
target_helper = spdk-nvmeof           # SPDK volume driver requires SPDK NVMe-
↪oF target driver
target_ip_address = 192.168.0.1       # NVMe-oF target IP address
target_port = 4260                    # NVMe-oF target port
target_prefix = nqn.2014-08.org.spdk  # NVMe-oF target nqn prefix
```

Table 69: Description of SPDK configuration options

Configuration option = Default value	Description
`spdk_max_queue_depth` = `64`	(Integer(min=1, max=128)) Queue depth for rdma transport.
`spdk_rpc_ip` = `None`	(String) The NVMe target remote configuration IP address.
`spdk_rpc_password` = `None`	(String) The NVMe target remote configuration password.
`spdk_rpc_port` = `8000`	(Port(min=0, max=65535)) The NVMe target remote configuration port.
`spdk_rpc_protocol` = `http`	(String(choices=[http, https])) Protocol to be used with SPDK RPC proxy
`spdk_rpc_username` = `None`	(String) The NVMe target remote configuration username.

StorPool volume driver

StorPool is distributed data storage software running on standard x86 servers. StorPool aggregates the performance and capacity of all drives into a shared pool of storage distributed among the servers. Within this storage pool the user creates thin-provisioned volumes that are exposed to the clients as block devices. StorPool consists of two parts wrapped in one package - a server and a client. The StorPool server allows a hypervisor to act as a storage node, while the StorPool client allows a hypervisor node to access the storage pool and act as a compute node. In OpenStack terms the StorPool solution allows each hypervisor node to be both a storage and a compute node simultaneously.

Prerequisites

- The controller and all the compute nodes must have access to the StorPool API service.
- All nodes where StorPool-backed volumes will be attached must have access to the StorPool data network and run the `storpool_block` service.
- If StorPool-backed Cinder volumes need to be created directly from Glance images, then the node running the `cinder-volume` service must also have access to the StorPool data network and run the `storpool_block` service.

Configuring the StorPool volume driver

A valid `/etc/storpool.conf` file is required; please contact the StorPool support team for assistance.

The StorPool Cinder volume driver has two configuration options that may be specified both in the global configuration (e.g. in a `cinder.conf` volume backend definition) and per volume type:

- `storpool_template`: specifies the StorPool template (replication, placement, etc. specifications defined once and used for multiple volumes and snapshots) to use for the Cinder volume type or, if specified globally, as a default value for Cinder volumes. There is no default value for this option, see `storpool_replication`.
- `storpool_replication`: if `storpool_template` is not set, the volume will be created with the specified chain replication and with the default placement constraints for the StorPool cluster. The default value for the chain replication is 3.

Using the StorPool volume driver

The most common use for the Cinder StorPool volume driver is probably attaching volumes to Nova instances. For this to work, the `nova-compute` service and the `os-brick` library must recognize the storpool volume attachment driver; please contact the StorPool support team for more information.

Currently there is no StorPool driver for Nova ephemeral volumes; to run Nova instances with a StorPool-backed volume as a root device, create a Cinder volume with the root filesystem image, make a snapshot, and let Nova create the instance with a root device as a new volume created from that snapshot.

Synology DSM volume driver

The `SynoISCSIDriver` volume driver allows Synology NAS to be used for Block Storage (cinder) in OpenStack deployments. Information on OpenStack Block Storage volumes is available in the DSM Storage Manager.

System requirements

The Synology driver has the following requirements:

- DSM version 6.0.2 or later.
- Your Synology NAS model must support advanced file LUN, iSCSI Target, and snapshot features. Refer to the Support List for applied models.

Note

The DSM driver is available in the OpenStack Newton release.

Supported operations

- Create, delete, clone, attach, and detach volumes.
- Create and delete volume snapshots.
- Create a volume from a snapshot.
- Copy an image to a volume.
- Copy a volume to an image.
- Extend a volume.
- Get volume statistics.

Driver configuration

Edit the `/etc/cinder/cinder.conf` file on your volume driver host.

Synology driver uses a volume in Synology NAS as the back end of Block Storage. Every time you create a new Block Storage volume, the system will create an advanced file LUN in your Synology volume to be used for this new Block Storage volume.

The following example shows how to use different Synology NAS servers as the back end. If you want to use all volumes on your Synology NAS, add another section with the volume number to differentiate between volumes within the same Synology NAS.

```
[default]
enabled_backends = ds1515pV1, ds1515pV2, rs3017xsV3, others

[ds1515pV1]
# configuration for volume 1 in DS1515+

[ds1515pV2]
# configuration for volume 2 in DS1515+

[rs3017xsV1]
# configuration for volume 1 in RS3017xs
```

Each section indicates the volume number and the way in which the connection is established. Below is an example of a basic configuration:

```
[Your_Section_Name]

# Required settings
volume_driver = cinder.volume.drivers.synology.synology_iscsi.SynoISCSIDriver
target_protocol = iscsi
target_ip_address = DS_IP
synology_admin_port = DS_PORT
```

(continues on next page)

(continued from previous page)

```
synology_username = DS_USER
synology_password = DS_PW
synology_pool_name = DS_VOLUME

# Optional settings
volume_backend_name = VOLUME_BACKEND_NAME
iscsi_secondary_ip_addresses = IP_ADDRESSES
driver_use_ssl = True
use_chap_auth = True
chap_username = CHAP_USER_NAME
chap_password = CHAP_PASSWORD
```

DS_PORT
This is the port for DSM management. The default value for DSM is 5000 (HTTP) and 5001 (HTTPS). To use HTTPS connections, you must set `driver_use_ssl = True`.

DS_IP
This is the IP address of your Synology NAS.

DS_USER
This is the account of any DSM administrator.

DS_PW
This is the password for `DS_USER`.

DS_VOLUME
This is the volume you want to use as the storage pool for the Block Storage service. The format is `volume[0-9]+`, and the number is the same as the volume number in DSM.

Note

If you set `driver_use_ssl` as `True`, `synology_admin_port` must be an HTTPS port.

Configuration options

The Synology DSM driver supports the following configuration options:

TOYOU NetStor Cinder driver

TOYOU NetStor series volume driver provides OpenStack Compute instances with access to TOYOU NetStor series storage systems.

TOYOU NetStor storage can be used with iSCSI or FC connection.

This documentation explains how to configure and connect the block storage nodes to TOYOU NetStor series storage.

Driver options

The following table contains the configuration options supported by the TOYOU NetStor iSCSI/FC driver.

Table 70: Description of TOYOU NetStor configuration options

Configuration option = Default value	Description
acs5000_copy_interval = 5	(Integer(min=3, max=100)) When volume copy task is going on,refresh volume status interval
acs5000_multiattach = False	(Boolean) Enable to allow volumes attaching to multiple hosts with no limit.
acs5000_volpool_name = [pool01]	(List of String) Comma separated list of storage system storage pools for volumes.

Supported operations

- Create, list, delete, attach (map), and detach (unmap) volumes.
- Create, list and delete volume snapshots.
- Create a volume from a snapshot.
- Copy an image to a volume.
- Copy a volume to an image.
- Clone a volume.
- Extend a volume.
- Migrate a volume.
- Manage/Unmanage volume.
- Revert to Snapshot.
- Multi-attach.
- Thin Provisioning.
- Extend Attached Volume.

Configure TOYOU NetStor iSCSI/FC backend

This section details the steps required to configure the TOYOU NetStor storage cinder driver.

1. In the `cinder.conf` configuration file under the `[DEFAULT]` section, set the enabled_backends parameter with the iSCSI or FC back-end group.

   - For Fibre Channel:

   ```
   [DEFAULT]
   enabled_backends = toyou-fc-1
   ```

   - For iSCSI:

   ```
   [DEFAULT]
   enabled_backends = toyou-iscsi-1
   ```

2. Add a backend group section for the backend group specified in the enabled_backends parameter.

3. In the newly created backend group section, set the following configuration options:

   - For Fibre Channel:

```
[toyou-fc-1]
# The TOYOU NetStor driver path
volume_driver = cinder.volume.drivers.toyou.acs5000.acs5000_fc.
↪Acs5000FCDriver
# Management IP of TOYOU NetStor storage array
san_ip = 10.0.0.10
# Management username of TOYOU NetStor storage array
san_login = cliuser
# Management password of TOYOU NetStor storage array
san_password = clipassword
# The Pool used to allocated volumes
acs5000_volpool_name = pool01
# Backend name
volume_backend_name = toyou-fc
```

   - For iSCSI:

```
[toyou-iscsi-1]
# The TOYOU NetStor driver path
volume_driver = cinder.volume.drivers.toyou.acs5000.acs5000_iscsi.
↪Acs5000ISCSIDriver
# Management IP of TOYOU NetStor storage array
san_ip = 10.0.0.10
# Management username of TOYOU NetStor storage array
san_login = cliuser
# Management password of TOYOU NetStor storage array
san_password = clipassword
# The Pool used to allocated volumes
acs5000_volpool_name = pool01
# Backend name
volume_backend_name = toyou-iscsi
```

TOYOU NetStor TYDS Cinder driver

TOYOU NetStor TYDS series volume driver provides OpenStack Compute instances with access to TOYOU NetStor TYDS series storage systems.

TOYOU NetStor TYDS storage can be used with iSCSI connection.

This documentation explains how to configure and connect the block storage nodes to TOYOU NetStor TYDS series storage.

Driver options

The following table contains the configuration options supported by the TOYOU NetStor TYDS iSCSI driver.

Table 71: Description of TOYOU NetStor TYDS configuration options

Configuration option = Default value	Description
`tyds_clone_progress_interval = 3`	(Integer) Interval (in seconds) for retrieving clone progress.
`tyds_copy_progress_interval = 3`	(Integer) Interval (in seconds) for retrieving copy progress.
`tyds_http_port = 80`	(Port(min=0, max=65535)) The port that connects to the http api.
`tyds_pools = [pool01]`	(List of String) The pool name where volumes are stored.
`tyds_stripe_size = 4M`	(String) Volume stripe size.

Supported operations

- Create Volume.
- Delete Volume.
- Attach Volume.
- Detach Volume.
- Extend Volume
- Create Snapshot.
- Delete Snapshot.
- Create Volume from Snapshot.
- Create Volume from Volume (clone).
- Create Image from Volume.
- Volume Migration (host assisted).

Configure TOYOU NetStor TOYOU TYDS iSCSI backend

This section details the steps required to configure the TOYOU NetStor TYDS storage cinder driver.

1. In the `cinder.conf` configuration file under the `[DEFAULT]` section, set the enabled_backends parameter with the iSCSI back-end group.

```
[DEFAULT]
enabled_backends = toyou-tyds-iscsi-1
```

2. Add a backend group section for the backend group specified in the enabled_backends parameter.
3. In the newly created backend group section, set the following configuration options:

```
[toyou-tyds-iscsi-1]
# The TOYOU NetStor TYDS driver path
volume_driver = cinder.volume.drivers.toyou.tyds.tyds.TYDSDriver
# Management http ip of TOYOU NetStor TYDS storage
```

(continues on next page)

(continued from previous page)

```
san_ip = 10.0.0.10
# Management http username of TOYOU NetStor TYDS storage
san_login = superuser
# Management http password of TOYOU NetStor TYDS storage
san_password = Toyou@123
# The Pool used to allocated volumes
tyds_pools = pool01
# Backend name
volume_backend_name = toyou-tyds-iscsi-1
```

Veritas ACCESS iSCSI driver

Veritas Access is a software-defined scale-out network-attached storage (NAS) solution for unstructured data that works on commodity hardware and takes advantage of placing data on premise or in the cloud based on intelligent policies. Through Veritas Access iSCSI Driver, OpenStack Block Storage can use Veritas Access backend as a block storage resource. The driver enables you to create iSCSI volumes that an OpenStack Block Storage server can allocate to any virtual machine running on a compute host.

Requirements

The Veritas ACCESS iSCSI Driver, version `1.0.0` and later, supports Veritas ACCESS release `7.4` and later.

Supported operations

- Create and delete volumes.
- Create and delete snapshots.
- Create volume from snapshot.
- Extend a volume.
- Attach and detach volumes.
- Clone volumes.

Configuration

1. Enable RESTful service on the Veritas Access Backend.
2. Create Veritas Access iSCSI target, add store and portal IP to it.

   You can create target and add portal IP, store to it as follows:

   ```
   Target> iscsi target create iqn.2018-02.com.veritas:target02
   Target> iscsi target store add target_fs iqn.2018-02.com.veritas:target02
   Target> iscsi target portal add iqn.2018-02.com.veritas:target02 10.10.10.
   ↪1
   ...
   ```

   You can add authentication to target as follows:

```
Target> iscsi target auth incominguser add iqn.2018-02.com.
↪veritas:target02 user1
...
```

3. Ensure that the Veritas Access iSCSI target service is online. If the Veritas Access iSCSI target service is not online, enable the service by using the CLI or REST API.

```
Target> iscsi service start
Target> iscsi service status
...
```

Define the following required properties in the `cinder.conf` file:

```
volume_driver = cinder.volume.drivers.veritas_access.veritas_iscsi.
↪ACCESSIscsiDriver
san_ip = va_console_ip
san_api_port = 14161
san_login = master
san_password = password
target_port = 3260
vrts_lun_sparse = True
vrts_target_config = /etc/cinder/vrts_target.xml
```

4. Define Veritas Access Target details in `/etc/cinder/vrts_target.xml`:

```
<?xml version="1.0" ?>
<VRTS>
     <VrtsTargets>
          <Target>
                  <Name>iqn.2018-02.com.veritas:target02</Name>
                  <PortalIP>10.10.10.1</PortalIP>
                  <Authentication>0</Authentication>
          </Target>
     </VrtsTargets>
</VRTS>
...
```

VMware VMDK driver

Use the VMware VMDK driver to enable management of the OpenStack Block Storage volumes on vCenter-managed data stores. Volumes are backed by VMDK files on data stores that use any VMware-compatible storage technology such as NFS, iSCSI, FiberChannel, and vSAN.

> **Note**
>
> The VMware VMDK driver requires vCenter version 5.1 at minimum.

Functional context

The VMware VMDK driver connects to vCenter, through which it can dynamically access all the data stores visible from the ESX hosts in the managed cluster.

When you create a volume, the VMDK driver creates a VMDK file on demand. The VMDK file creation completes only when the volume is subsequently attached to an instance. The reason for this requirement is that data stores visible to the instance determine where to place the volume. Before the service creates the VMDK file, attach a volume to the target instance.

The running vSphere VM is automatically reconfigured to attach the VMDK file as an extra disk. Once attached, you can log in to the running vSphere VM to rescan and discover this extra disk.

With the update to ESX version 6.0, the VMDK driver now supports NFS version 4.1.

Configuration

The recommended volume driver for OpenStack Block Storage is the VMware vCenter VMDK driver. When you configure the driver, you must match it with the appropriate OpenStack Compute driver from VMware and both drivers must point to the same server.

In the `nova.conf` file, use this option to define the Compute driver:

```
compute_driver = vmwareapi.VMwareVCDriver
```

In the `cinder.conf` file, use this option to define the volume driver:

```
volume_driver = cinder.volume.drivers.vmware.vmdk.VMwareVcVmdkDriver
```

The following table lists various options that the drivers support for the OpenStack Block Storage configuration (`cinder.conf`):

Table 72: Description of VMware configuration options

Configuration option = Default value	Description
[DEFAULT]	
`vmware_adapter_t` = `lsiLogic`	(String) Default adapter type to be used for attaching volumes.
`vmware_api_retry` = `10`	(Integer) Number of times VMware vCenter server API must be retried upon connection related issues.
`vmware_ca_file` = `None`	(String) CA bundle file to use in verifying the vCenter server certificate.
`vmware_cluster_n` = `None`	(Multi-valued) Name of a vCenter compute cluster where volumes should be created.
`vmware_connectio` = `10`	(Integer) Maximum number of connections in http connection pool.
`vmware_host_ip` = `None`	(String) IP address for connecting to VMware vCenter server.
`vmware_host_pass` = `None`	(String) Password for authenticating with VMware vCenter server.
`vmware_host_port` = `443`	(Port number) Port number for connecting to VMware vCenter server.
`vmware_host_user` = `None`	(String) Username for authenticating with VMware vCenter server.
`vmware_host_vers` = `None`	(String) Optional string specifying the VMware vCenter server version. The driver attempts to retrieve the version from VMware vCenter server. Set this configuration only if you want to override the vCenter server version.
`vmware_image_tra` = `7200`	(Integer) Timeout in seconds for VMDK volume transfer between Cinder and Glance.
`vmware_insecure` = `False`	(Boolean) If true, the vCenter server certificate is not verified. If false, then the default CA truststore is used for verification. This option is ignored if vmware_ca_file is set.
`vmware_max_objec` = `100`	(Integer) Max number of objects to be retrieved per batch. Query results will be obtained in batches from the server and not in one shot. Server may still limit the count to something less than the configured value.
`vmware_task_poll` = `2.0`	(Floating point) The interval (in seconds) for polling remote tasks invoked on VMware vCenter server.
`vmware_tmp_dir` = `/tmp`	(String) Directory where virtual disks are stored during volume backup and restore.
`vmware_volume_fo` = `Volumes`	(String) Name of the vCenter inventory folder that will contain Cinder volumes. This folder will be created under OpenStack/<project_folder>, where project_folder is of format Project (<volume_project_id>).
`vmware_wsdl_loca` = `None`	(String) Optional VIM service WSDL Location e.g http://<server>/vimService.wsdl. Optional over-ride to default location for bug work-arounds.

VMDK disk type

The VMware VMDK drivers support the creation of VMDK disk file types `thin`, `lazyZeroedThick` (sometimes called thick or flat), or `eagerZeroedThick`.

A thin virtual disk is allocated and zeroed on demand as the space is used. Unused space on a Thin disk is available to other users.

A lazy zeroed thick virtual disk will have all space allocated at disk creation. This reserves the entire disk space, so it is not available to other users at any time.

An eager zeroed thick virtual disk is similar to a lazy zeroed thick disk, in that the entire disk is allocated at creation. However, in this type, any previous data will be wiped clean on the disk before the write. This can mean that the disk will take longer to create, but can also prevent issues with stale data on physical media.

Use the `vmware:vmdk_type` extra spec key with the appropriate value to specify the VMDK disk file type. This table shows the mapping between the extra spec entry and the VMDK disk file type:

Table 73: Extra spec entry to VMDK disk file type mapping

Disk file type	Extra spec key	Extra spec value
thin	`vmware:vmdk_type`	`thin`
lazyZeroedThick	`vmware:vmdk_type`	`thick`
eagerZeroedThick	`vmware:vmdk_type`	`eagerZeroedThick`

If you do not specify a `vmdk_type` extra spec entry, the disk file type will default to `thin`.

The following example shows how to create a `lazyZeroedThick` VMDK volume by using the appropriate `vmdk_type`:

```
$ openstack volume type create THICK_VOLUME
$ openstack volume type set --property vmware:vmdk_type=thick THICK_VOLUME
$ openstack volume create --size 1 --type THICK_VOLUME VOLUME1
```

Clone type

With the VMware VMDK drivers, you can create a volume from another source volume or a snapshot point. The VMware vCenter VMDK driver supports the `full` and `linked/fast` clone types. Use the `vmware:clone_type` extra spec key to specify the clone type. The following table captures the mapping for clone types:

Table 74: Extra spec entry to clone type mapping

Clone type	Extra spec key	Extra spec value
full	`vmware:clone_type`	`full`
linked/fast	`vmware:clone_type`	`linked`

If you do not specify the clone type, the default is `full`.

The following example shows linked cloning from a source volume, which is created from an image:

```
$ openstack volume type create FAST_CLONE
$ openstack volume type set --property vmware:clone_type=linked FAST_CLONE
$ openstack volume create --size 1 --type FAST_CLONE --image MYIMAGE SOURCE_
↪VOL
$ openstack volume create --size 1 --source SOURCE_VOL DEST_VOL
```

Adapter type

The VMware vCenter VMDK driver supports the adapter types `LSI Logic Parallel`, `BusLogic Parallel`, `LSI Logic SAS`, `VMware Paravirtual` and `IDE` for volumes. Use the `vmware:adapter_type` extra spec key to specify the adapter type. The following table captures the mapping for adapter types:

Table 75: Extra spec entry to adapter type mapping

Adapter type	Extra spec key	Extra spec value
BusLogic Parallel	`vmware:adapter_type`	`busLogic`
IDE	`vmware:adapter_type`	`ide`
LSI Logic Parallel	`vmware:adapter_type`	`lsiLogic`
LSI Logic SAS	`vmware:adapter_type`	`lsiLogicsas`
VMware Paravirtual	`vmware:adapter_type`	`paraVirtual`

If you do not specify the adapter type, the default is the value specified by the config option `vmware_adapter_type`.

Use vCenter storage policies to specify back-end data stores

This section describes how to configure back-end data stores using storage policies. In vCenter 5.5 and greater, you can create one or more storage policies and expose them as a Block Storage volume-type to a vmdk volume. The storage policies are exposed to the vmdk driver through the extra spec property with the `vmware:storage_profile` key.

For example, assume a storage policy in vCenter named `gold_policy.` and a Block Storage volume type named `vol1` with the extra spec key `vmware:storage_profile` set to the value `gold_policy`. Any Block Storage volume creation that uses the `vol1` volume type places the volume only in data stores that match the `gold_policy` storage policy.

The Block Storage back-end configuration for vSphere data stores is automatically determined based on the vCenter configuration. If you configure a connection to connect to vCenter version 5.5 or later in the `cinder.conf` file, the use of storage policies to configure back-end data stores is automatically supported.

Note

You must configure any data stores that you configure for the Block Storage service for the Compute service.

To configure back-end data stores by using storage policies

1. In vCenter, tag the data stores to be used for the back end.

OpenStack also supports policies that are created by using vendor-specific capabilities; for example vSAN-specific storage policies.

> **Note**
>
> The tag value serves as the policy. For details, see *Storage policy-based configuration in vCenter*.

2. Set the extra spec key `vmware:storage_profile` in the desired Block Storage volume types to the policy name that you created in the previous step.
3. Optionally, for the `vmware_host_version` parameter, enter the version number of your vSphere platform. For example, `5.5`.

   This setting overrides the default location for the corresponding WSDL file. Among other scenarios, you can use this setting to prevent WSDL error messages during the development phase or to work with a newer version of vCenter.
4. Complete the other vCenter configuration parameters as appropriate.

> **Note**
>
> Any volume that is created without an associated policy (that is to say, without an associated volume type that specifies `vmware:storage_profile` extra spec), there is no policy-based placement for that volume.

Supported operations

The VMware vCenter VMDK driver supports these operations:

- Create, delete, attach, and detach volumes.

  > **Note**
  >
  > When a volume is attached to an instance, a reconfigure operation is performed on the instance to add the volumes VMDK to it. The user must manually rescan and mount the device from within the guest operating system.

- Create, list, and delete volume snapshots.

  > **Note**
  >
  > Allowed only if volume is not attached to an instance.

- Create a volume from a snapshot.

  > **Note**
  >
  > The vmdk UUID in vCenter will not be set to the volume UUID if the vCenter version is 6.0 or above and the extra spec key `vmware:clone_type` in the destination volume type is set to

`linked`.

- Copy an image to a volume.

> **Note**
>
> Only images in `vmdk` disk format with `bare` container format are supported. The `vmware_disktype` property of the image can be `preallocated`, `sparse`, `streamOptimized` or `thin`.

- Copy a volume to an image.

> **Note**
>
> - Allowed only if the volume is not attached to an instance.
> - This operation creates a `streamOptimized` disk image.

- Clone a volume.

> **Note**
>
> - Supported only if the source volume is not attached to an instance.
> - The vmdk UUID in vCenter will not be set to the volume UUID if the vCenter version is 6.0 or above and the extra spec key `vmware:clone_type` in the destination volume type is set to `linked`.

- Backup a volume.

> **Note**
>
> This operation creates a backup of the volume in `streamOptimized` disk format.

- Restore backup to new or existing volume.

> **Note**
>
> Supported only if the existing volume doesnt contain snapshots.

- Change the type of a volume.

> **Note**
>
> This operation is supported only if the volume state is `available`.

- Extend a volume.

Storage policy-based configuration in vCenter

You can configure Storage Policy-Based Management (SPBM) profiles for vCenter data stores supporting the Compute, Image service, and Block Storage components of an OpenStack implementation.

In a vSphere OpenStack deployment, SPBM enables you to delegate several data stores for storage, which reduces the risk of running out of storage space. The policy logic selects the data store based on accessibility and available storage space.

Prerequisites

- Determine the data stores to be used by the SPBM policy.
- Determine the tag that identifies the data stores in the OpenStack component configuration.
- Create separate policies or sets of data stores for separate OpenStack components.

Create storage policies in vCenter

1. In vCenter, create the tag that identifies the data stores:
   1. From the *Home* screen, click *Tags*.
   2. Specify a name for the tag.
   3. Specify a tag category. For example, `spbm-cinder`.
2. Apply the tag to the data stores to be used by the SPBM policy.

   Note

   For details about creating tags in vSphere, see the vSphere documentation.

3. In vCenter, create a tag-based storage policy that uses one or more tags to identify a set of data stores.

   Note

   For details about creating storage policies in vSphere, see the vSphere documentation.

Data store selection

If storage policy is enabled, the driver initially selects all the data stores that match the associated storage policy.

If two or more data stores match the storage policy, the driver chooses a data store that is connected to the maximum number of hosts.

In case of ties, the driver chooses the data store with lowest space utilization, where space utilization is defined by the `(1-freespace/totalspace)` meters.

These actions reduce the number of volume migrations while attaching the volume to instances.

The volume must be migrated if the ESX host for the instance cannot access the data store that contains the volume.

Virtuozzo Storage driver

The Virtuozzo Storage driver is a fault-tolerant distributed storage system that is optimized for virtualization workloads. Set the following in your `cinder.conf` file, and use the following options to configure it.

```
volume_driver = cinder.volume.drivers.vzstorage.VZStorageDriver
```

Table 76: Description of Virtuozzo Storage configuration options

Configuration option = Default value	Description
`vzstorage_default_volume` = `raw`	(String) Default format that will be used when creating volumes if no volume format is specified.
`vzstorage_mount_options` = `None`	(List of String) Mount options passed to the vzstorage client. See section of the pstorage-mount man page for details.
`vzstorage_mount_point_ba` = `$state_path/mnt`	(String) Base dir containing mount points for vzstorage shares.
`vzstorage_shares_config` = `/etc/cinder/vzstorage_shares`	(String) File with the list of available vzstorage shares.
`vzstorage_sparsed_volume` = `True`	(Boolean) Create volumes as sparsed files which take no space rather than regular files when using raw format, in which case volume creation takes lot of time.
`vzstorage_used_ratio` = `0.95`	(Float) Percent of ACTUAL usage of the underlying volume before no new volumes can be allocated to the volume destination.

Windows iSCSI volume driver

Windows Server offers an integrated iSCSI Target service that can be used with OpenStack Block Storage in your stack.

Being entirely a software solution, consider it in particular for mid-sized networks where the costs of a SAN might be excessive.

The Windows iSCSI Block Storage driver works with OpenStack Compute on any hypervisor.

This driver creates volumes backed by fixed-type VHD images on Windows Server 2012 and dynamic-type VHDX on Windows Server 2012 R2 and onwards, stored locally on a user-specified path. The system uses those images as iSCSI disks and exports them through iSCSI targets. Each volume has its own iSCSI target.

The `cinder-volume` service as well as the required Python components will be installed directly onto the Windows node.

Prerequisites

The Windows iSCSI volume driver depends on the `wintarget` Windows service. This will require the `iSCSI Target Server` Windows feature to be installed.

Note

The Cinder MSI will automatically enable this feature, if available (some minimal Windows versions do not provide it).

You may check the availability of this feature by running the following: `Get-WindowsFeature FS-iSCSITarget-Server`

The Windows Server installation requires at least 16 GB of disk space. The volumes hosted by this node will need extra space.

Configuring cinder-volume

Below is a configuration sample for using the Windows iSCSI Driver. Append those options to your already existing `cinder.conf` file, described at *Install and configure a storage node*.

```
[DEFAULT]
enabled_backends = winiscsi

[winiscsi]
volume_driver = cinder.volume.drivers.windows.iscsi.WindowsISCSIDriver
windows_iscsi_lun_path = C:\iSCSIVirtualDisks
volume_backend_name = winiscsi

# The following config options are optional
#
# use_chap_auth = true
# target_port = 3260
# target_ip_addres = <IP_USED_FOR_ISCSI_TRAFFIC>
# iscsi_secondary_ip_addresses = <SECONDARY_ISCSI_IPS>
# reserved_percentage = 5
```

The `windows_iscsi_lun_path` config option specifies the directory in which VHD backed volumes will be stored.

Windows SMB volume driver

Description

The Windows SMB volume driver leverages pre-existing SMB shares, used to store volumes as virtual disk images.

The main reasons to use the Windows SMB driver are:

- ease of management and use
- great integration with other Microsoft technologies (e.g. Hyper-V Failover Cluster)
- suitable for a various range of deployment types and sizes

The `cinder-volume` service as well as the required Python components will be installed directly onto designated Windows nodes (preferably the ones exposing the shares).

Common deployment scenarios

The SMB driver is designed to support a variety of scenarios, such as:

- Scale-Out File Servers (SoFS), providing highly available SMB shares.
- standalone Windows or Samba shares
- any other SMB 3.0 capable device

By using SoFS shares, the virtual disk images are stored on Cluster Shared Volumes (CSV).

A common practice involves deploying CSVs on top of SAN backed LUNs (exposed to all the nodes of the cluster through iSCSI or Fibre Channel). In absence of a SAN, Storage Spaces/Storage Spaces Direct (S2D) may be used for the underlying storage.

Note

S2D is commonly used in hyper-converged deployments.

Features

VHD and VHDX are the currently supported image formats and may be consumed by Hyper-V and KVM compute nodes. By default, dynamic (thinly provisioned) images will be used, unless configured otherwise.

The driver accepts one or more shares that will be reported to the Cinder scheduler as storage pools. This can provide means of tiering, allowing specific shares (pools) to be requested through volume types.

```
openstack volume type set $volume_type --property pool_name=$pool_name
```

Frontend QoS specs may be associated with the volume types and enforced on the consumer side (e.g. Hyper-V).

```
openstack volume qos create $rule_name --property consumer=front-end --
↪property total_bytes_sec=20971520
openstack volume qos associate $rule_name $volume_type_id
openstack volume create $volume_name --type $volume_type_id --size $size
```

The Cinder Backup Service can be run on Windows. This driver stores the volumes using vhdx images stored on SMB shares which can be attached in order to retrieve the volume data and send it to the backup service.

Prerequisites:

- All physical disks must be in byte mode
- rb+ must be used when writing backups to disk

Clustering support

Active-Active Cinder clustering is currently experimental and should not be used in production. This implies having multiple Cinder Volume services handling the same share simultaneously.

On the other hand, Active-Passive clustering can easily be achieved, configuring the Cinder Volume service as clustered using Microsoft Failover Cluster.

By using SoFS, you can provide high availability of the shares used by Cinder. This can be used in conjunction with the Nova Hyper-V cluster driver, which allows clustering virtual machines. This ensures that when a compute node is compromised, the virtual machines are transparently migrated to a healthy node, preserving volume connectivity.

Note

The Windows SMB driver is the only Cinder driver that may be used along with the Nova Hyper-V cluster driver. The reason is that during an unexpected failover, the volumes need to be available on the destination compute node side.

Prerequisites

Before setting up the SMB driver, you will need to create and configure one or more SMB shares that will be used for storing virtual disk images.

Note

The driver does not manage share permissions. You will have to make sure that Cinder as well as share consumers (e.g. Nova, Hyper-V) have access.

Note that Hyper-V VMs are run using a built-in user group: `NT VIRTUAL MACHINE\Virtual Machines`.

The easiest way to provide share access is by using Active Directory accounts. You may grant share access to the users running OpenStack services, as well as the compute nodes (and optionally storage nodes), using per computer account access rules. One of the main advantages is that by doing so, you dont need to pass share credentials to Cinder (and implicitly volume consumers).

By granting access to a computer account, youre basically granting access to the LocalSystem account of that node, and thus to the VMs running on that host.

Note

By default, OpenStack services deployed using the MSIs are run as LocalSystem.

Once youve granted share access to a specific account, dont forget to also configure file system level permissions on the directory exported by the share.

Configuring cinder-volume

Below is a configuration sample for using the Windows SMB Driver. Append those options to your already existing `cinder.conf` file, described at *Install and configure a storage node*.

```
[DEFAULT]
enabled_backends = winsmb

[winsmb]
volume_backend_name = myWindowsSMBBackend
volume_driver = cinder.volume.drivers.windows.smbfs.WindowsSmbfsDriver
```

(continues on next page)

(continued from previous page)

```
smbfs_mount_point_base = C:\OpenStack\mnt\
smbfs_shares_config = C:\Program Files\Cloudbase Solutions\OpenStack\etc\
↪cinder\smbfs_shares_list

# The following config options are optional
#
# image_volume_cache_enabled = true
# image_volume_cache_max_size_gb = 100
# image_volume_cache_max_count = 10
#
# nas_volume_prov_type = thin
# smbfs_default_volume_format = vhdx
# max_over_subscription_ratio = 1.5
# reserved_percentage = 5
# smbfs_pool_mappings = //addr/share:pool_name,//addr/share2:pool_name2
```

The `smbfs_mount_point_base` config option allows you to specify where the shares will be *mounted*. This directory will contain symlinks pointing to the shares used by Cinder. Each symlink name will be a hash of the actual share path.

Configuring the list of available shares

In addition to `cinder.conf`, you will need to have another config file, providing a list of shares that will be used by Cinder for storing disk images. In the above sample, this file is referenced by the `smbfs_shares_config` option.

The share list config file must contain one share per line, optionally including mount options. You may also add comments, using a # at the beginning of the line.

Bellow is a sample of the share list config file:

```
# Cinder Volume shares
//sofs-cluster/share
//10.0.0.10/volumes -o username=user,password=mypassword
```

Keep in mind that Linux hosts can also consume those volumes. For this reason, the mount options resemble the ones used by mount.cifs (in fact, those will actually be passed to mount.cifs by the Nova Linux nodes).

In case of Windows nodes, only the share location, username and password will be used when mounting the shares. The share address must use slashes instead of backslashes (as opposed to what Windows admins may expect) because of the above mentioned reason.

Depending on the configured share access rules, you may skip including share credentials in the config file, as described in the *Prerequisites* section.

Configuring Nova credentials

The SMB volume driver relies on the `nova assisted volume snapshots` feature when snapshotting in-use volumes, as do other similar drivers using shared filesystems.

By default, the Nova policy requires admin rights for this operation. You may provide Cinder specific credentials to be used when requesting Nova assisted volume snapshots, as shown bellow:

```
[nova]
region_name=RegionOne
auth_strategy=keystone
auth_type=password
auth_url=http://keystone_host/identity
project_name=service
username=nova
password=password
project_domain_name=Default
user_domain_name=Default
```

Configuring storage pools

Each share is reported to the Cinder scheduler as a storage pool.

By default, the share name will be the name of the pool. If needed, you may provide pool name mappings, specifying a custom pool name for each share, as shown bellow:

```
smbfs_pool_mappings = //addr/share:pool0
```

In the above sample, the `//addr/share` share will be reported as `pool0`.

YADRO Cinder Driver

YADRO Cinder driver provides iSCSI and FC support for TATLIN.UNIFIED storages.

Supported Functions

Basic Functions

- Create Volume
- Delete Volume
- Attach Volume
- Detach Volume
- Extend Volume
- Create Volume from Volume (clone)
- Create Image from Volume
- Volume Migration (host assisted)

Additional Functions

- Extend an Attached Volume
- Thin Provisioning
- Manage/Unmanage Volume
- Image Cache
- Multiattach

- High Availability

Configuration

Set up TATLIN.UNIFIED storage

You need to specify settings as described below for storage systems. For details about each setting, see the users guide of the storage system.

1. User account

   Create a storage account belonging to the admin user group.

2. Pool

   Create a storage pool that is used by the driver.

3. Ports

   Setup Ethernet or FC ports you want to export volumes to.

4. Hosts

   Create storage hosts and set ports of the initiators. One host must correspond to one initiator.

5. Host Group

   Create storage host group and add hosts created on the previous step to the host group.

6. CHAP Authentication

   Set up CHAP credentials for iSCSI storage hosts (if CHAP is used).

Set up YADRO Cinder Driver

Add the following configuration to /etc/cinder/cinder.conf:

```
[iscsi-1]
volume_driver=cinder.volume.drivers.yadro.tatlin_iscsi.TatlinISCSIVolumeDriver
san_ip=<management_ip>
san_login=<login>
san_password=<password>
tat_api_retry_count=<count>
api_port=<management_port>
pool_name=<cinder_volumes_pool>
export_ports=<port1>,<port2>
host_group=<name>
max_resource_count=<count>
auth_method=<CHAP|NONE>
chap_username=<chap_username>
chap_password=<chap_password>
```

or

```
[fc-1]
volume_driver=cinder.volume.drivers.yadro.tatlin_fc.TatlinFCVolumeDriver
san_ip=<management_ip>
```

(continues on next page)

(continued from previous page)

```
san_login=<login>
san_password=<password>
tat_api_retry_count=<count>
api_port=<management_port>
pool_name=<cinder_volumes_pool>
export_ports=<port1>,<port2>
host_group=<name>
max_resource_count=<count>
```

volume_driver
Volume driver name.

san_ip
TATLIN.UNIFIED management IP address or FQDN.

san_login
TATLIN.UNIFIED user name.

san_password
TATLIN.UNIFIED user password.

tat_api_retry_count
Number of repeated requests to TATLIN.UNIFIED.

api_port
TATLIN.UNIFIED management port. Default: 443.

pool_name
TATLIN.UNIFIED name of pool for Cinder Volumes.

export_ports
Comma-separated data ports for volumes to be exported to.

host_group
TATLIN.UNIFIED host group name.

max_resource_count
Limit on the number of resources for TATLIN.UNIFIED. Default: 150

auth_method (only iSCSI)

Authentication method:

- `CHAP` use CHAP authentication (default)

chap_username, chap_password (if auth_method=CHAP)
CHAP credentials to validate the initiator.

Zadara Storage VPSA volume driver

Zadara Storage, Virtual Private Storage Array (VPSA) is the first software defined, Enterprise-Storage-as-a-Service. It is an elastic and private block and file storage system which, provides enterprise-grade data protection and data management storage services.

The `ZadaraVPSAISCSIDriver` volume driver allows the Zadara Storage VPSA to be used as a volume back end storage in OpenStack deployments.

System requirements

To use Zadara Storage VPSA Volume Driver you will require:

- Zadara Storage VPSA version 15.07 and above
- iSCSI or iSER host interfaces

Supported operations

- Create, delete, attach, and detach volumes
- Create, list, and delete volume snapshots
- Create a volume from a snapshot
- Copy an image to a volume
- Copy a volume to an image
- Clone a volume
- Extend a volume
- Migrate a volume with back end assistance
- Manage and unmanage a volume
- Manage and unmanage volume snapshots
- Multiattach a volume

Configuration

1. Create a VPSA pool(s) or make sure you have an existing pool(s) that will be used for volume services. The VPSA pool(s) will be identified by its ID (pool-xxxxxxxx). For further details, see the VPSAs user guide.
2. Adjust the `cinder.conf` configuration file to define the volume driver name along with a storage back end entry for each VPSA pool that will be managed by the block storage service. Each back end entry requires a unique section name, surrounded by square brackets (or parentheses), followed by options in `key=value` format.

Note

Restart cinder-volume service after modifying `cinder.conf`.

Sample minimum back end configuration

```
[DEFAULT]
enabled_backends = vpsa

[vpsa]
zadara_vpsa_host = 172.31.250.10
zadara_vpsa_port = 80
zadara_user = vpsauser
zadara_password = mysecretpassword
```

(continues on next page)

(continued from previous page)

```
zadara_use_iser = false
zadara_vpsa_poolname = pool-00000001
volume_driver = cinder.volume.drivers.zadara.zadara.ZadaraVPSAISCSIDriver
volume_backend_name = vpsa
```

Driver-specific options

This section contains the configuration options that are specific to the Zadara Storage VPSA driver.

Table 77: Description of Zadara configuration options

Configuration option = Default value	Description
zadara_access_key = None	(String) VPSA access key
zadara_default_sn = False	(Boolean) VPSA - Attach snapshot policy for volumes. If the option is neither configured nor provided as metadata, the VPSA will inherit the default value.
zadara_gen3_vol_c = False	(Boolean) VPSA - Enable compression for volumes. If the option is neither configured nor provided as metadata, the VPSA will inherit the default value.
zadara_gen3_vol_d = False	(Boolean) VPSA - Enable deduplication for volumes. If the option is neither configured nor provided as metadata, the VPSA will inherit the default value.
zadara_ssl_cert_v = True	(Boolean) If set to True the http client will validate the SSL certificate of the VPSA endpoint.
zadara_use_iser = True	(Boolean) VPSA - Use ISER instead of iSCSI
zadara_vol_encryp = False	(Boolean) VPSA - Default encryption policy for volumes. If the option is neither configured nor provided as metadata, the VPSA will inherit the default value.
zadara_vol_name_t = OS_%s	(String) VPSA - Default template for VPSA volume names
zadara_vpsa_host = None	(HostAddress) VPSA - Management Host name or IP address
zadara_vpsa_pooln = None	(String) VPSA - Storage Pool assigned for volumes
zadara_vpsa_port = None	(Port(min=0, max=65535)) VPSA - Port number
zadara_vpsa_use_s = False	(Boolean) VPSA - Use SSL connection

Note

By design, all volumes created within the VPSA are thin provisioned.

Backup drivers

Ceph backup driver

The Ceph backup driver backs up volumes of any type to a Ceph back-end store. The driver can also detect whether the volume to be backed up is a Ceph RBD volume, and if so, it tries to perform incremental and differential backups.

For source Ceph RBD volumes, you can perform backups within the same Ceph pool (not recommended). You can also perform backups between different Ceph pools and between different Ceph clusters.

At the time of writing, differential backup support in Ceph/librbd was quite new. This driver attempts a differential backup in the first instance. If the differential backup fails, the driver falls back to full backup/copy.

If incremental backups are used, multiple backups of the same volume are stored as snapshots so that minimal space is consumed in the backup store. It takes far less time to restore a volume than to take a full copy.

By default, all incremental backups are held on the source volume storage, which can take up much disk space on the usually more expensive primary storage compared to backup storage. Enabling the option `backup_ceph_max_snapshots` can save disk space on the source volume storage by only keeping a limited number of snapshots per backup volume. After every successful creation of a new incremental backup, the Ceph backup driver will then ensure that excess snapshots of the corresponding backup volume are deleted so that only the `backup_ceph_max_snapshots` most recent snapshots are kept on the primary storage. However, this can cause incremental backups to automatically become full backups instead if a user manually deleted at least `backup_ceph_max_snapshots` incremental backups. In that case the next snapshot, being a full backup, will require more disk space on the backup storage and will take longer to complete than an incremental backup would have.

Thus, the option allows to configure a tradeoff between required space on the source volume storage and required space on the backup storage as well as a longer backup process under the above conditions.

Note

Block Storage enables you to:

- Restore to a new volume, which is the default and recommended action.
- Restore to the original volume from which the backup was taken. The restore action takes a full copy because this is the safest action.

To enable the Ceph backup driver, include the following option in the `cinder.conf` file:

```
backup_driver = cinder.backup.drivers.ceph.CephBackupDriver
```

The following configuration options are available for the Ceph backup driver.

Table 78: Description of Ceph backup driver configuration options

Configuration option = Default value	Description
backup_ce = 134217728	(Integer) The chunk size, in bytes, that a backup is broken into before transfer to the Ceph object store.
backup_ce = /etc/ceph/ceph.conf	(String) Ceph configuration file to use.
backup_ce = False	(Boolean) If True, apply JOURNALING and EXCLUSIVE_LOCK feature bits to the backup RBD objects to allow mirroring
backup_ce = 0	(Integer) Number of the most recent snapshots to keep. 0 indicates to keep an unlimited number of snapshots. Configuring this option can save disk space by only keeping a limited number of snapshots on the source volume storage. However, if a user deletes all incremental backups which still have snapshots on the source storage, the next incremental backup will automatically become a full backup as no common snapshot exists anymore.
backup_ce = backups	(String) The Ceph pool where volume backups are stored.
backup_ce = 0	(Integer) RBD stripe count to use when creating a backup image.
backup_ce = 0	(Integer) RBD stripe unit to use when creating a backup image.
backup_ce = cinder	(String) The Ceph user to connect with. Default here is to use the same user as for Cinder volumes. If not using cephx this should be set to None.
restore_c = True	(Boolean) If True, always discard excess bytes when restoring volumes i.e. pad with zeroes.

This example shows the default options for the Ceph backup driver.

```
backup_ceph_conf=/etc/ceph/ceph.conf
backup_ceph_user = cinder-backup
backup_ceph_chunk_size = 134217728
backup_ceph_pool = backups
backup_ceph_stripe_unit = 0
backup_ceph_stripe_count = 0
backup_ceph_max_snapshots = 0
```

GlusterFS backup driver

The GlusterFS backup driver backs up volumes of any type to GlusterFS.

To enable the GlusterFS backup driver, include the following option in the `cinder.conf` file:

```
backup_driver = cinder.backup.drivers.glusterfs.GlusterfsBackupDriver
```

The following configuration options are available for the GlusterFS backup driver.

Table 79: Description of GlusterFS backup driver configuration options

Configuration option = Default value	Description
`glusterfs_backup_mount_point` = `$state_path/backup_mount`	(String) Base dir containing mount point for gluster share.
`glusterfs_backup_share` = `None`	(String) GlusterFS share in <hostname|ipv4addr|ipv6addr>:<gluster_vol_name> format. Eg: 1.2.3.4:backup_vol

NFS backup driver

The backup driver for the NFS back end backs up volumes of any type to an NFS exported backup repository.

To enable the NFS backup driver, include the following option in the `[DEFAULT]` section of the `cinder.conf` file:

```
backup_driver = cinder.backup.drivers.nfs.NFSBackupDriver
```

The following configuration options are available for the NFS back-end backup driver.

Table 80: Description of NFS backup driver configuration options

Configuration option = Default value	Description
backup_conta = None	(String) Custom directory to use for backups.
backup_enabl = True	(Boolean) Enable or Disable the timer to send the periodic progress notifications to Ceilometer when backing up the volume to the backend storage. The default value is True to enable the timer.
backup_file_ = 1999994880	(Integer) The maximum size in bytes of the files used to hold backups. If the volume being backed up exceeds this size, then it will be backed up into multiple files. backup_file_size also determines the buffer size used to build backup files, so should be scaled according to available RAM and number of workers. backup_file_size must be a multiple of backup_sha_block_size_bytes.
backup_mount = 3	(Integer(min=1)) The number of attempts to mount NFS shares before raising an error.
backup_mount = None	(String) Mount options passed to the NFS client. See NFS man page for details.
backup_mount = $state_path/backup_mount	(String) Base dir containing mount point for NFS share.
backup_posix = $state_path/backup	(String) Path specifying where to store backups.
backup_sha_b = 32768	(Integer) The size in bytes that changes are tracked for incremental backups. backup_file_size has to be multiple of backup_sha_block_size_bytes.
backup_share = None	(String) NFS share in hostname:path, ipv4addr:path, or [ipv6addr]:path format.

POSIX file systems backup driver

The POSIX file systems backup driver backs up volumes of any type to POSIX file systems.

To enable the POSIX file systems backup driver, include the following option in the `cinder.conf` file:

```
backup_driver = cinder.backup.drivers.posix.PosixBackupDriver
```

The following configuration options are available for the POSIX file systems backup driver.

Table 81: Description of POSIX backup driver configuration options

Configuration option = Default value	Description
`backup_con` = `None`	(String) Custom directory to use for backups.
`backup_ena` = `True`	(Boolean) Enable or Disable the timer to send the periodic progress notifications to Ceilometer when backing up the volume to the backend storage. The default value is True to enable the timer.
`backup_fil` = `1999994880`	(Integer) The maximum size in bytes of the files used to hold backups. If the volume being backed up exceeds this size, then it will be backed up into multiple files. backup_file_size also determines the buffer size used to build backup files, so should be scaled according to available RAM and number of workers. backup_file_size must be a multiple of backup_sha_block_size_bytes.
`backup_pos` = `$state_path/backup`	(String) Path specifying where to store backups.
`backup_sha` = `32768`	(Integer) The size in bytes that changes are tracked for incremental backups. backup_file_size has to be multiple of backup_sha_block_size_bytes.

Swift backup driver

The backup driver for the swift back end performs a volume backup to an object storage system.

To enable the swift backup driver, include the following option in the `cinder.conf` file:

```
backup_driver = cinder.backup.drivers.swift.SwiftBackupDriver
```

The following configuration options are available for the Swift back-end backup driver.

Table 82: Description of Swift backup driver configuration options

Configuration option = Default value	Description
backup_swift_auth = per_user	(String(choices=[per_user, single_user])) Swift authentication mechanism (per_user or single_user).
backup_swift_auth_ins = False	(Boolean) Bypass verification of server certificate when making SSL connection to Swift.
backup_swift_auth_url = None	(URI) The URL of the Keystone endpoint
backup_swift_auth_ver = 1	(String) Swift authentication version. Specify 1 for auth 1.0, or 2 for auth 2.0 or 3 for auth 3.0
backup_swift_block_si = 32768	(Integer) The size in bytes that changes are tracked for incremental backups. backup_swift_object_size has to be multiple of backup_swift_block_size.
backup_swift_ca_cert_ = None	(String) Location of the CA certificate file to use for swift client requests.
backup_swift_containe = volumebackups	(String) The default Swift container to use
backup_swift_create_s = None	(String) The storage policy to use when creating the Swift container. If the container already exists the storage policy cannot be enforced
backup_swift_enable_p = True	(Boolean) Enable or Disable the timer to send the periodic progress notifications to Ceilometer when backing up the volume to the Swift backend storage. The default value is True to enable the timer.
backup_swift_key = None	(String) Swift key for authentication
backup_swift_object_s = 52428800	(Integer) The size in bytes of Swift backup objects
backup_swift_project = None	(String) Swift project/account name. Required when connecting to an auth 3.0 system
backup_swift_project_ = None	(String) Swift project domain name. Required when connecting to an auth 3.0 system
backup_swift_retry_at = 3	(Integer) The number of retries to make for Swift operations
backup_swift_retry_ba = 2	(Integer) The backoff time in seconds between Swift retries
backup_swift_service_ = False	(Boolean) Send a X-Service-Token header with service auth credentials. If enabled you also must set the service_user group and enable send_service_user_token.
backup_swift_tenant = None	(String) Swift tenant/account name. Required when connecting to an auth 2.0 system
backup_swift_url = None	(URI) The URL of the Swift endpoint
backup_swift_user = None	(String) Swift user name
backup_swift_user_dom = None	(String) Swift user domain name. Required when connecting to an auth 3.0 system
keystone_catalog_info = identity:Identity Service:publicURL	(String) Info to match when looking for keystone in the service catalog. Format is: separated values of the form: <service_type>:<service_name>:<endpoint_type> - Only used if backup_swift_auth_url is unset
swift_catalog_info = object-store:swift:pu	(String) Info to match when looking for swift in the service catalog. Format is: separated values of the form: <service_type>:<service_name>:<endpoint_type> - Only used if backup_swift_url is unset

To enable the swift backup driver for 1.0, 2.0, or 3.0 authentication version, specify `1`, `2`, or `3` correspondingly. For example:

```
backup_swift_auth_version = 2
```

In addition, the 2.0 authentication system requires the definition of the `backup_swift_tenant` setting:

```
backup_swift_tenant = <None>
```

This example shows the default options for the Swift back-end backup driver.

```
backup_swift_url = http://localhost:8080/v1/AUTH_
backup_swift_auth_url = http://localhost:5000/v3
backup_swift_auth = per_user
backup_swift_auth_version = 1
backup_swift_user = <None>
backup_swift_user_domain = <None>
backup_swift_key = <None>
backup_swift_container = volumebackups
backup_swift_object_size = 52428800
backup_swift_project = <None>
backup_swift_project_domain = <None>
backup_swift_retry_attempts = 3
backup_swift_retry_backoff = 2
backup_compression_algorithm = zlib
```

Google Cloud Storage backup driver

The Google Cloud Storage (GCS) backup driver backs up volumes of any type to Google Cloud Storage.

To enable the GCS backup driver, include the following option in the `cinder.conf` file:

```
backup_driver = cinder.backup.drivers.gcs.GoogleBackupDriver
```

The following configuration options are available for the GCS backup driver.

Table 83: Description of GCS backup driver configuration options

Configuration option = Default value	Description
backup_gcs_block = 32768	(Integer) The size in bytes that changes are tracked for incremental backups. backup_gcs_object_size has to be multiple of backup_gcs_block_size.
backup_gcs_bucke = None	(String) The GCS bucket to use.
backup_gcs_bucke = US	(String) Location of GCS bucket.
backup_gcs_crede = None	(String) Absolute path of GCS service account credential file.
backup_gcs_enabl = True	(Boolean) Enable or Disable the timer to send the periodic progress notifications to Ceilometer when backing up the volume to the GCS backend storage. The default value is True to enable the timer.
backup_gcs_num_r = 3	(Integer) Number of times to retry.
backup_gcs_objec = 52428800	(Integer) The size in bytes of GCS backup objects.
backup_gcs_proje = None	(String) Owner project id for GCS bucket.
backup_gcs_proxy = None	(URI) URL for http proxy access.
backup_gcs_reade = 2097152	(Integer) GCS object will be downloaded in chunks of bytes.
backup_gcs_retry = [429]	(List of String) List of GCS error codes.
backup_gcs_stora = NEARLINE	(String) Storage class of GCS bucket.
backup_gcs_user_ = gcscinder	(String) Http user-agent string for gcs api.
backup_gcs_write = 2097152	(Integer) GCS object will be uploaded in chunks of bytes. Pass in a value of -1 if the file is to be uploaded as a single chunk.

S3 Storage backup driver

The S3 backup driver backs up volumes to any type of Amazon S3 and S3 compatible object storages.

To enable the S3 backup driver, include the following option in the `cinder.conf` file:

```
backup_driver = cinder.backup.drivers.s3.S3BackupDriver
```

The following configuration options are available for the S3 backup driver.

Table 84: Description of S3 backup driver configuration options

Configuration option = Default value	Description
backup_s3_block_ = 32768	(Integer) The size in bytes that changes are tracked for incremental backups. backup_s3_object_size has to be multiple of backup_s3_block_size.
backup_s3_ca_cer = None	(String) path/to/cert/bundle.pem - A filename of the CA cert bundle to use.
backup_s3_enable = True	(Boolean) Enable or Disable the timer to send the periodic progress notifications to Ceilometer when backing up the volume to the S3 backend storage. The default value is True to enable the timer.
backup_s3_endpoi = None	(String) The url where the S3 server is listening.
backup_s3_http_p = <>	(String) Address or host for the http proxy server.
backup_s3_https_ = <>	(String) Address or host for the https proxy server.
backup_s3_max_po = 10	(Integer) The maximum number of connections to keep in a connection pool.
backup_s3_md5_va = True	(Boolean) Enable or Disable md5 validation in the s3 backend.
backup_s3_object = 52428800	(Integer) The size in bytes of S3 backup objects
backup_s3_retry_ = 4	(Integer) An integer representing the maximum number of retry attempts that will be made on a single request.
backup_s3_retry_ = legacy	(String) A string representing the type of retry mode. e.g: legacy, standard, adaptive
backup_s3_sse_cu = None	(String) The SSECustomerAlgorithm. backup_s3_sse_customer_key must be set at the same time to enable SSE.
backup_s3_sse_cu = None	(String) The SSECustomerKey. backup_s3_sse_customer_algorithm must be set at the same time to enable SSE.
backup_s3_store_ = None	(String) The S3 query token access key.
backup_s3_store_ = volumebackups	(String) The S3 bucket to be used to store the Cinder backup data.
backup_s3_store_ = None	(String) The S3 query token secret key.
backup_s3_timeou = 60	(Float) The time in seconds till a timeout exception is thrown.
backup_s3_verify = True	(Boolean) Enable or Disable ssl verify.

This section describes how to configure the cinder-backup service and its drivers.

The volume drivers are included with the Block Storage repository. To set a backup driver, use the `backup_driver` flag. By default there is no backup driver enabled.

Block Storage schedulers

Block Storage service uses the `cinder-scheduler` service to determine how to dispatch block storage requests.

For more information, see:

Cinder Scheduler Filters

AvailabilityZoneFilter

Filters Backends by availability zone.

CapabilitiesFilter

BackendFilter to work with resource (instance & volume) type records.

CapacityFilter

Capacity filters based on volume backends capacity utilization.

DifferentBackendFilter

Schedule volume on a different back-end from a set of volumes.

DriverFilter

DriverFilter filters backend based on a filter function and metrics.

DriverFilter filters based on volume backends provided filter function and metrics.

InstanceLocalityFilter

Schedule volume on the same host as a given instance.

This filter enables selection of a storage back-end located on the host where the instances hypervisor is running. This provides data locality: the instance and the volume are located on the same physical machine.

In order to work:

- The Extended Server Attributes extension needs to be active in Nova (this is by default), so that the OS-EXT-SRV-ATTR:host property is returned when requesting instance info.
- Either an account with privileged rights for Nova must be configured in Cinder configuration (configure a keystone authentication plugin in the [nova] section), or the user making the call needs to have sufficient rights (see extended_server_attributes in Nova policy).

JsonFilter

Backend filter for simple JSON-based grammar for selecting backends.

If you want to choose one of your backend, make a query hint, for example:

cinder create hint query=[=, $backend_id, rbd:vol@ceph#cloud]

RetryFilter

Filter out previously attempted hosts

A host passes this filter if it has not already been attempted for scheduling. The scheduler needs to add previously attempted hosts to the retry key of filter_properties in order for this to work correctly. For example:

```
{
 'retry': {
           'backends': ['backend1', 'backend2'],
           'num_attempts': 3,
          }
}
```

SameBackendFilter

Schedule volume on the same back-end as another volume.

Cinder Scheduler Weights

AllocatedCapacityWeigher

Allocated Capacity Weigher weighs hosts by their allocated capacity.

The default behavior is to place new volume to the host allocated the least space. This weigher is intended to simulate the behavior of SimpleScheduler. If you prefer to place volumes to host allocated the most space, you can set the `allocated_capacity_weight_multiplier` option to a positive number and the weighing has the opposite effect of the default.

CapacityWeigher

Capacity Weigher weighs hosts by their virtual or actual free capacity.

For thin provisioning, weigh hosts by their virtual free capacity calculated by the total capacity multiplied by the max over subscription ratio and subtracting the provisioned capacity; Otherwise, weigh hosts by their actual free capacity, taking into account the reserved space.

The default is to spread volumes across all hosts evenly. If you prefer stacking, you can set the `capacity_weight_multiplier` option to a negative number and the weighing has the opposite effect of the default.

ChanceWeigher

Chance Weigher assigns random weights to hosts.

Used to spread volumes randomly across a list of equally suitable hosts.

GoodnessWeigher

Goodness Weigher. Assign weights based on a hosts goodness function.

Goodness rating is the following:

```
  0 -- host is a poor choice
  .
  .
 50 -- host is a good choice
  .
  .
100 -- host is a perfect choice
```

VolumeNumberWeigher

Weigher that weighs hosts by volume number in backends.

The default is to spread volumes across all hosts evenly. If you prefer stacking, you can set the `volume_number_multiplier` option to a positive number and the weighing has the opposite effect of the default.

Log files used by Block Storage

The corresponding log file of each Block Storage service is stored in the `/var/log/cinder/` directory of the host on which each service runs.

Table 85: **Log files used by Block Storage services**

Log file	Service/interface (for CentOS, Fedora, openSUSE, Red Hat Enterprise Linux, and SUSE Linux Enterprise)	Service/interface (for Ubuntu and Debian)
api.log	openstack-cinder-api	cinder-api
cinder-manage.log	cinder-manage	cinder-manage
scheduler.log	openstack-cinder-scheduler	cinder-scheduler
volume.log	openstack-cinder-volume	cinder-volume

Policy Personas and Permissions

Beginning with the Xena release, the Block Storage service API v3 takes advantage of the default authentication and authorization apparatus supplied by the Keystone project to give operators a rich set of default policies to control how users interact with the Block Storage service API.

This document describes Cinders part in an effort across OpenStack services to provide a consistent and useful default RBAC configuration. (This effort is referred to as secure RBAC for short.)

Vocabulary Note

We need to clarify some terms well be using below.

Project
: This is a grouping of users into a unit that can own cloud resources. (This is what used to be called a tenant, but you should never call it that.) Users, projects, and their associations are created in Keystone.

Service
: This is an OpenStack component that users interact with through an API it provides. For example, Cinder is the OpenStack code name for the service that provides the Block Storage API version 3. Cinder is also known as the OpenStack Block Storage service.

The point of making this distinction is that theres another use of the term project that is relevant to the discussion, but that were **not** going to use. Each OpenStack service is produced and maintained by a project team. *We will not be using the term project in that sense in this document. Well always use the term service.* (If you are new to OpenStack, this wont be a problem. But if youre discussing this content with someone whos been around OpenStack for a while, youll want to be clear about this so that youre not talking past each other.)

The Cinder Personas

This is easiest to explain if we introduce the five personas Cinder recognizes. In the list below, a system refers to the deployed system (that is, Cinder and all its services), and a project refers to a container or namespace for resources.

- In order to consume resources, a user must be assigned to a project by being given a role (for example, member) in that project. Thats done in Keystone; its not a Cinder concern.

  See Default Roles in the Keystone documentation for more information.

Table 86: The Five Personas

who	what
projec reader	Has access to the API for read-only requests that affect only project-specific resources (that is, cannot create, update, or delete resources within a project)
projec memb	A normal user in a project.
projec admin	All the normal stuff plus some minor administrative abilities in a particular project, for example, able to set the default volume type for a project. (The administrative abilities are minor in the sense that they have no impact on the Cinder system, they only allow the project-admin to make system-safe changes isolated to that project.)
systen reader	Has read only access to the API; like the project-reader, but can read any project recognized by cinder.
systen admin	Has the highest level of authorization on the system and can perform any action in Cinder. In most deployments, only the operator, deployer, or other highly trusted person will be assigned this persona. This is a Cinder super-user who can do *everything*, both with respect to the Cinder system and all individual projects.

Note

The Keystone project provides the ability to describe additional personas, but Cinder does not currently recognize them. In particular:

- Cinder does not recognize the `domain` scope at all. So even if you successfully request a domain-scoped token from the Identity service, you wont be able to use it with Cinder. Instead, request a project-scoped token for the particular project in your domain that you want to act upon.
- Cinder does not recognize a system-member persona, that is, a user with the `member` role on a `system`. The default Cinder policy configuration treats such a user as identical to the *system-reader* persona described above.

More information about roles and scope is available in the Keystone Administrator Guides.

Note

Privacy Expectations

Cinders model of resources (volumes, backups, snapshots, etc.) is that they are owned by the *project*. Thus, they are shared by all users who have a role assignment on that project, no matter what persona that user has been assigned.

For example, if Alice and Bob are in Project P, and Alice has persona project-member while Bob has persona project-reader, if Alice creates volume V in Project P, Bob can see volume V in the volume-list response, and Bob can read all the volume metadata on volume V that Alice can readeven volume metadata that Alice may have added to the volume. The key point here is that even though Alice created volume V, *its not her volume*. The volume is owned by Project P and is available to all users who have authorization on that project via role assignments in keystone. What a user can do with volume V depends on whether that user has an admin, member, or reader role in project P.

With respect to Project P, the personas with system scope (system-admin and system-reader) have access to the project in the sense that a cinder system-admin can do anything in Project P that the project-admin can do plus some additional powers. A cinder system-reader has read-only access to everything in Project P that the system-admin can access.

The above describe the default policy configuration for Cinder. It is possible to modify policies to obtain different behavior, but that is beyond the scope of this document.

Implementation Schedule

For reasons that will become clear in this section, the secure RBAC effort is being implemented in Cinder in two phases. In Xena, there are three personas.

Table 87: The 3 Xena Personas

who	Keystone technical info
project-reader	`reader` role on a `project`, resulting in project-scope
project-member	`member` role on a `project`, resulting in project-scope
system-admin	`admin` role on a `project`, but recognized by Cinder as having permission to act on the cinder *system*

Note that you *cannot* create a project-admin persona on your own simply by assigning the `admin` role to a user. Such assignment results in that user becoming a system-admin.

In the Yoga release, we plan to implement the full set of Cinder personas:

Table 88: The 5 Yoga Personas

who	Keystone technical info
project-reader	`reader` role on a `project`, resulting in project-scope
project-member	`member` role on a `project`, resulting in project-scope
project-admin	`admin` role on a `project`, resulting in project-scope
system-reader	`reader` role on a `system`, resulting in system-scope
system-admin	`admin` role on a `system`, resulting in system-scope

Note that although the underlying technical information changes for the system-admin, the range of actions performable by that persona does not change.

Cinder Permissions Matrix

Now that you know who the personas are, heres what they can do with respect to the policies that are recognized by Cinder. Keep in mind that only three of the personas (project-reader, project-member, and system-admin) are implemented in the Xena release.

NOTE: the columns in () will be deleted; they are here for comparison as the matrix is validated by human beings.

Table 89: Attachments (Microversion 3.27)

functionality	API call	policy name	(old rule)	project-reader	project-member	project-admin	system-reader	system-admin	(old owner)	(old admin)
Create attachment	`POST / attach`	volume:atta	empty	no	yes	yes	no	yes	yes	yes
Update attachment	`PUT / attach {attac`	volume:atta	rule:adm	no	yes	yes	no	yes	yes	yes
Delete attachment	`DELETE / attach {attac`	volume:atta	rule:adm	no	yes	yes	no	yes	yes	yes
Mark a volume attachment process as completed (in-use)	Microve 3.44 `POST / attach {attac action` (os-complet	volume:atta	rule:adm	no	yes	yes	no	yes	yes	yes
Allow multi-attach of bootable volumes	This is a secondary check on `POST / attach` which is governed by another policy	volume:mu	rule:adm	no	yes	yes	no	yes	yes	yes

Table 90: User Messages (Microversion 3.3)

functionality	API call	policy name	(old rule)	project reader	project-membe	projec admin	system reader	system admin	(old owne	(old admin)
List messages	`GET /messages`	message:ge	rule:admi	yes	yes	yes	yes	yes	yes	yes
Show message	`GET /messages/{message_id}`	message:ge	rule:admi	yes	yes	yes	yes	yes	yes	yes
Delete message	`DELETE /messages/{message_id}`	message:de	rule:admi	no	yes	yes	no	yes	yes	yes

Table 91: Clusters (Microversion 3.7)

functionality	API call	policy name	(old rule)	project-reader	project-membe	project-admin	system reader	system admin	(old owner)	(old admin)
List clusters	`GET /cluste` `GET /cluste detail`	clusters:get_	rule:adn	no	no	no	no	yes	no	yes
Show cluster	`GET /cluste {clust`	clusters:get	rule:adn	no	no	no	no	yes	no	yes
Update cluster	`PUT /cluste {clust`	clusters:upd	rule:adn	no	no	no	no	yes	no	yes

Table 92: Workers (Microversion 3.24)

functionality	API call	policy name	(old rule)	project reader	project-membe	project admin	system reader	system admin	(old owner	(old admin)
Clean up workers	`POST /workers/cleanup`	workers:clea	rule:adı	no	no	no	no	yes	no	yes

Table 93: Snapshots

functionality	API call	policy name	(old rule)	project-reader	project-membe	project-admin	system-reader	system-admin	(old owner)	(old admin)
List snapshots	`GET /snapsh` `GET /snapsh detail`	volume:get	rule:adn	yes	yes	yes	yes	yes	yes	yes
List or show snapshots with extended attributes	`GET /snapsh {snaps` `GET /snapsh detail`	volume_ex	rule:adn	yes	yes	yes	yes	yes	yes	yes
Create snapshot	`POST / snapsh`	volume:cre	rule:adn	no	yes	yes	no	yes	yes	yes
Show snapshot	`GET / snapsh {snaps`	volume:get	rule:adn	yes	yes	yes	yes	yes	yes	yes
Update snapshot	`PUT / snapsh {snaps`	volume:upc	rule:adn	no	yes	yes	no	yes	yes	yes
Delete snapshot	`DELETE / snapsh {snaps`	volume:del	rule:adn	no	yes	yes	no	yes	yes	yes
Reset status of a snapshot.	`POST / snapsh {snaps action` (os-reset_st	volume_ex	rule:adn	no	no	no	no	yes	no	yes
Update status (and optionally progress of snapshot	`POST / snapsh {snaps action` (os-update_	snapshot_ext	empty	no	yes	yes	no	yes	yes	yes
Force delete	`POST /`	volume_ex	rule:adn	no	no	no	no	yes	no	yes

Table 94: Snapshot Metadata

functionality	API call	policy name	(old rule)	project-reader	project-membe	project-admin	system-reader	system-admin	(old owner)	(old admin)
Show snapshots metadata or one specified metadata with a given key	`GET / snapsh {snaps metada` `GET / snapsh {snaps metada {key}`	volume:get	rule:adn	yes	yes	yes	yes	yes	yes	yes
Update snapshots metadata or one specified metadata with a given key	`PUT / snapsh {snaps metada` `PUT / snapsh {snaps metada {key}`	volume:upc	rule:adn	no	yes	yes	no	yes	yes	yes
Delete snapshots specified metadata with a given key	`DELETE / snapsh {snaps metada {key}`	volume:del	rule:adn	no	yes	yes	no	yes	yes	yes

Table 95: Backups

func-tion-ality	API call	policy name	(old rule)	project-reader	project-membe	project-admin	system reader	system admin	(old owner)	(old ad-min)
List back-ups	GET / backup GET / backup detail	backup:	rule:adn	yes	yes	yes	yes	yes	yes	yes
In-clude project at-tributes in the list back-ups, show backup re-sponses	Microve 3.18 Adds os-bac to the fol-low-ing re-sponses GET / backup detail GET / backup {backu The ability to make these API calls is gov-erned by other poli-cies.	backup:	rule:adn	no	no	no	no	yes	no	yes
Create backup	POST / backup	backup:	empty	no	yes	yes	no	yes	yes	yes
Show backup	GET / backup {backu	backup:	rule:adn	yes	yes	yes	yes	yes	yes	yes
Up-date backup	Microve 3.9 PUT /	backup:	rule:adn	no	yes	yes	no	yes	yes	yes

Table 96: Groups (Microversion 3.13)

functionality	API call	policy name	(old rule)	project-reader	project-member	project-admin	system-reader	system-admin	(old owner)	(old admin)
List groups	`GET /groups` `GET /groups/detail`	group:g	rule:adm	yes	yes	yes	yes	yes	yes	yes
Create group, create group from src	`POST /groups` Microversion 3.14: `POST /groups/action` (create-from-src)	group:c	empty	no	yes	yes	no	yes	yes	yes
Show group	`GET /groups/{group`	group:g	rule:adm	yes	yes	yes	yes	yes	yes	yes
Update group	`PUT /groups/{group`	group:u	rule:adm	no	yes	yes	no	yes	yes	yes
Include project attributes in the list groups, show group responses	Microve 3.58 Adds `projec` to the following responses: `GET /groups/detail` `GET /groups/{group` The ability to	group:g	rule:adm	no	no	no	no	yes	no	yes

Table 97: Group Types (Microversion 3.11)

functionality	API call	policy name	(old rule)	project-reader	project-member	project-admin	system-reader	system-admin	(old owner)	(old admin)
DEPRE Create, update or delete a group type	(NOTE: new policies split POST, PUT, DELET `POST /group_` `PUT /group_{group` `DELETE /group_{group`	group:g	rule:adn	no	no	no	no	yes	no	yes
NEW Create a group type	`POST /group_`	group:g	(new policy)	no	no	no	no	yes	n/a	n/a
NEW Update a group type	`PUT /group_{group`	group:g	(new policy)	no	no	no	no	yes	n/a	n/a
NEW Delete a group type	`DELETE /group_{group`	group:g	(new policy)	no	no	no	no	yes	n/a	n/a
Show group type with	Adds `group_`	group:ac	rule:adn	no	no	no	no	yes	no	yes

Table 98: Group Snapshots (Microversion 3.14)

functionality	API call	policy name	(old rule)	project-reader	project-membe	project-admin	system-reader	system-admin	(old owner)	(old admin)
List group snapshots	GET /group_ GET /group_detail	group:g	rule:adn	yes	yes	yes	yes	yes	yes	yes
Create group snapshot	POST /group_	group:cı	empty	no	yes	yes	no	yes	yes	yes
Show group snapshot	GET /group_{group	group:g	rule:adn	yes	yes	yes	yes	yes	yes	yes
Delete group snapshot	DELETE /group_{group	group:d	rule:adn	no	yes	yes	no	yes	yes	yes
Update group snapshot	PUT /group_{group Note: even though the policy is defined, this call is not implemented in the Block Storage API.	group:u	rule:adn	no	yes	yes	no	yes	yes	yes
Reset status of group snapshot	Microve 3.19 POST /group_{group action	group:re	rule:adn	no	no	no	no	yes	no	yes

Table 99: Group Actions

functionality	API call	policy name	(old rule)	project-reader	project-membe	project-admin	system-reader	system-admin	(old owner)	(old admin)
Delete group	POST / groups {group action (delete)	group:d	rule:adn	no	yes	yes	no	yes	yes	yes
Reset status of group	Microve 3.20 POST / groups {group action (reset_statu	group:re	rule:adn	no	no	no	no	yes	no	yes
Enable replication	Microve 3.38 POST / groups {group action (enable_rep	group:er	rule:adn	no	yes	yes	no	yes	yes	yes
Disable replication	Microve 3.38 POST / groups {group action (disable_rep	group:d	rule:adn	no	yes	yes	no	yes	yes	yes
Fail over replication	Microve 3.38 POST / groups {group action	group:fa	rule:adn	no	yes	yes	no	yes	yes	yes

Table 100: QOS specs

functionality	API call	policy name	(old rule)	project-reader	project-membe	project-admin	system-reader	system-admin	(old owner)	(old admin)
List qos specs or list all associations	GET /qos-sp GET /qos-sp{qos_i associ	volume_ex	rule:adn	no	no	no	no	yes	no	yes
Show qos specs	GET /qos-sp{qos_i	volume_ex	rule:adn	no	no	no	no	yes	no	yes
Create qos specs	POST /qos-sp	volume_ex	rule:adn	no	no	no	no	yes	no	yes
Update qos specs: update key/valu in the qos-spec or update the volume-types associated with the qos-spec	PUT /qos-sp{qos_i GET /qos-sp{qos_i associ vol_ty GET /qos-sp{qos_i disass vol_ty GET /qos-sp{qos_i disass (yes, these GETs are really updates)	volume_ex	rule:adn	no	no	no	no	yes	no	yes
Delete a qos-spec, or remove a list of keys from th	DELETE /qos-sp{qos_i PUT /qos-sp{qos_i	volume_ex	rule:adn	no	no	no	no	yes	no	yes

Table 101: Quotas

functionality	API call	policy name	(old rule)	project-reader	project-membe	project-admin	system-reader	system-admin	(old owner)	(old admin)
DEPRE Show or update project quota class	(NOTE: new policies split GET and PUT) `GET /os-quo {proje` `PUT /os-quo {proje`	volume_ex	rule:adn	no	no	no	no	yes	no	yes
NEW Show project quota class	`GET /os-quo {proje`	volume_ex	(new policy)	no	no	no	no	yes	n/a	n/a
NEW Update project quota class	`PUT /os-quo {proje`	volume_ex	(new policy)	no	no	no	no	yes	n/a	n/a
Show project quota (including usage and default)	`GET /os-quo {proje` `GET /os-quo {proje defaul` `GET /os-quo {proje usage=`	volume_ex	rule:adn	yes	yes	yes	yes	yes	yes	yes
Update project quota	`PUT /os-quo {proje`	volume_ex	rule:adn	no	no	no	no	yes	no	yes

Table 102: Capabilities

functionality	API call	policy name	(old rule)	project reader	project member	project admin	system reader	system admin	(old owner)	(old admin)
Show backend capabilities	`GET /capabilitie {host_name}`	volume_extens	rule:ac	no	no	no	no	yes	no	yes

Table 103: Services

functionality	API call	policy name	(old rule)	project-reader	project-membe	project-admin	system-reader	system-admin	(old owner)	(old admin)
List all services	`GET /os-ser`	volume_ex	rule:adn	no	no	no	no	yes	no	yes
Update service	`PUT /os-ser enable` `PUT /os-ser disabl` `PUT /os-ser disabl` `PUT /os-ser freeze` `PUT /os-ser thaw` `PUT /os-ser failov` `PUT /os-ser failov` (microversion 3.26) `PUT /os-ser set-lo` `PUT /os-ser get-lo`	volume_ex	rule:adn	no	no	no	no	yes	no	yes
Freeze a backend host. Secondary check; must also satisfy volume_ex to	`PUT /os-ser freeze`	volume:fre	rule:adn	no	no	no	no	yes	no	yes

Table 104: Volume Types

functionality	API call	policy name	(old rule)	project-reader	project-membe	project-admin	system reader	system admin	(old owner)	(old admin)
DEPRE Create, update and delete volume type (new policies for create/upda	`POST /types` `PUT /types/{type_` `DELETE /types`	volume_ex	rule:adn	no	no	no	no	yes	no	yes
NEW Create a volume type	`POST /types`	volume_ex	(new policy)	no	no	no	no	yes	no	yes
NEW Update a volume type	`PUT /types/{type_`	volume_ex	(new policy)	no	no	no	no	yes	no	yes
NEW Delete a volume type	`DELETE /types/{type_`	volume_ex	(new policy)	no	no	no	no	yes	no	yes
Show a specific volume type	`GET /types/{type_`	volume_ex	empty	yes	yes	yes	yes	yes	yes	yes
List vol-	`GET /types`	volume_ex	empty	yes	yes	yes	yes	yes	yes	yes

Table 105: Volume Actions

functionality	API call	policy name	(old rule)	project-reader	project-membe	project-admin	system-reader	system-admin	(old owner)	(old admin)
Extend a volume	POST /volume {volum action (os-extend)	volume:ext	rule:adn	no	yes	yes	no	yes	yes	yes
Extend an attached volume	Microve 3.42 POST /volume {volum action (os-extend)	volume:ext	rule:adn	no	yes	yes	no	yes	yes	yes
Revert a volume to a snapshot	Microve 3.40 POST /volume {volum action (revert)	volume:rev	rule:adn	no	yes	yes	no	yes	yes	yes
Reset status of a volume	POST /volume {volum action (os-reset_st	volume_ex	rule:adn	no	no	no	no	yes	no	yes
Retype a volume	POST /volume {volum action (os-retype)	volume:ret	rule:adn	no	yes	yes	no	yes	yes	yes
Update a volumes read-only	POST /volume {volum action (os-	volume:upc	rule:adn	no	yes	yes	no	yes	yes	yes

Table 106: Volume Transfers

functionality	API call	policy name	(old rule)	project-reader	project-membe	project-admin	system reader	system admin	(old owner)	(old admin)
List volume transfer	GET / os-vol GET / os-vol detail GET / volume GET / volume detail	volume:get	rule:adn	yes	yes	yes	yes	yes	yes	yes
Create a volume transfer	POST / os-vol POST / volume	volume:cre	rule:adn	no	yes	yes	no	yes	yes	yes
Show one specified volume transfer	GET / os-vol {trans GET / volume {trans	volume:get	rule:adn	yes	yes	yes	yes	yes	yes	yes
Accept a volume transfer	POST / os-vol {trans accept POST / volume {trans accept	volume:acc	empty	no	yes	yes	no	yes	yes	yes
Delete volume transfer	DELETE / os-vol {trans DELETE	volume:del	rule:adn	no	yes	yes	no	yes	yes	yes

Table 107: Volume Metadata

functionality	API call	policy name	(old rule)	project-reader	project-membe	project-admin	system-reader	system-admin	(old owner)	(old admin)
Show volumes metadata or one specified metadata with a given key.	`GET /volume{volum metada` `GET /volume{volum metada{key}` `POST /volume{volum action` (os-show_ir	volume:get	rule:adn	yes	yes	yes	yes	yes	yes	yes
Create volume metadata	`POST /volume{volum metada`	volume:cre	rule:adn	no	yes	yes	no	yes	yes	yes
Update volumes metadata or one specified metadata with a given key	`PUT /volume{volum metada` `PUT /volume{volum metada{key}`	volume:upd	rule:adn	no	yes	yes	no	yes	yes	yes
Delete volumes specified metadata with a given key	`DELETE /volume{volum metada{key}`	volume:del	rule:adn	no	yes	yes	no	yes	yes	yes
DEPRE	(NOTE:	volume_ex	rule:adn	no	yes	yes	no	yes	yes	yes

Table 108: Volume Type Extra-Specs

functionality	API call	policy name	(old rule)	project-reader	project-membe	project-admin	system reader	system admin	(old owner)	(old admin)
List type extra specs	`GET / types/ {type_ extra_`	volume_ex	empty	yes	yes	yes	yes	yes	yes	yes
Create type extra specs	`POST / types/ {type_ extra_`	volume_ex	rule:adn	no	no	no	no	yes	no	yes
Show one specified type extra specs	`GET / types/ {type_ extra_ {extra`	volume_ex	empty	yes	yes	yes	yes	yes	yes	yes
Update type extra specs	`PUT / types/ {type_ extra_ {extra`	volume_ex	rule:adn	no	no	no	no	yes	no	yes
Delete type extra specs	`DELETE / types/ {type_ extra_ {extra`	volume_ex	rule:adn	no	no	no	no	yes	no	yes
Include extra_spec fields that may reveal sensitive information about the deployment that should not be exposed to end	`GET / types` `GET / types/ {type_` `GET / types/ {type_ extra_` `GET / types/ {type_ extra_ {extra` The ability to make these API calls is gov-	volume_ex	rule:adn	no	no	no	no	yes	no	yes

Table 109: Volumes

functionality	API call	policy name	(old rule)	project-reader	project-membe	project-admin	system reader	system admin	(old owner)	(old admin)
Create volume	POST / volume	volume:cre	empty	no	yes	yes	no	yes	yes	yes
Create volume from image	POST / volume	volume:cre	empty	no	yes	yes	no	yes	yes	yes
Show volume	GET / volume {volum	volume:get	rule:adn	yes	yes	yes	yes	yes	yes	yes
List volumes or get summary of volumes	GET / volume GET / volume detail GET / volume summar	volume:get	rule:adn	yes	yes	yes	yes	yes	yes	yes
Update volume or update a volumes bootable status	PUT / volume POST / volume {volum action (os-set_boo	volume:upo	rule:adn	no	yes	yes	no	yes	yes	yes
Delete volume	DELETE / volume {volum	volume:del	rule:adn	no	yes	yes	no	yes	yes	yes
Force Delete a volume (Microversion 3.23)	DELETE / volume {volum force=	volume:for	rule:adn	no	no	no	no	yes	no	yes
List or show volume with host	Adds os-vol to the fol-	volume_ex	rule:adn	no	no	no	no	yes	no	yes

Table 110: Default Volume Types (Microversion 3.62)

functionality	API call	policy name	(old rule)	project-reader	project-membe	project-admin	system reader	system admin	(old owner)	(old admin)
Set or update default volume type for a project	`PUT /defaul`	volume_ex	rule:syst	no	no	yes	no	yes	no	yes
Get default type for a project	`GET /defaul` `{proje` (Note: a project-* persona can always determine their effective default-type by making the `GET /v3/{proje types/defaul` call, which is governed by the volume_ex policy.)	volume_ex	rule:syst	no	no	yes	no	yes	no	yes
Get all de- fault	`GET /defaul`	volume_ex	role:adm and sys	no	no	no	no	yes	no	yes

Policy configuration

Configuration

The following is an overview of all available policies in Cinder. For information on how to write a custom policy file to modify these policies, see *policy.yaml* in the Cinder configuration documentation.

cinder

admin_or_owner

Default
`is_admin:True or (role:admin and is_admin_project:True) or project_id:%(project_id)s`

DEPRECATED: This rule will be removed in the Yoga release. Default rule for most non-Admin APIs.

system_or_domain_or_project_admin

Default
`(role:admin and system_scope:all) or (role:admin and domain_id:%(domain_id)s) or (role:admin and project_id:%(project_id)s)`

DEPRECATED: This rule will be removed in the Yoga release. Default rule for admins of cloud, domain or a project.

context_is_admin

Default
`role:admin`

Decides what is required for the is_admin:True check to succeed.

admin_api

Default
`is_admin:True or (role:admin and is_admin_project:True)`

Default rule for most Admin APIs.

xena_system_admin_or_project_reader

Default
`(role:admin) or (role:reader and project_id:%(project_id)s)`

NOTE: this purely role-based rule recognizes only project scope

xena_system_admin_or_project_member

Default
`(role:admin) or (role:member and project_id:%(project_id)s)`

NOTE: this purely role-based rule recognizes only project scope

volume:attachment_create

Default
`rule:xena_system_admin_or_project_member`

Operations

- **POST** `/attachments`

Create attachment.

`volume:attachment_update`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **PUT** `/attachments/{attachment_id}`

Update attachment.

`volume:attachment_delete`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **DELETE** `/attachments/{attachment_id}`

Delete attachment.

`volume:attachment_complete`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **POST** `/attachments/{attachment_id}/action (os-complete)`

Mark a volume attachment process as completed (in-use)

`volume:multiattach_bootable_volume`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **POST** `/attachments`

Allow multiattach of bootable volumes.

`message:get_all`

Default
`rule:xena_system_admin_or_project_reader`

Operations

- **GET** `/messages`

List messages.

`message:get`

Default
`rule:xena_system_admin_or_project_reader`

Operations

- **GET** `/messages/{message_id}`

Show message.

`message:delete`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **DELETE** `/messages/{message_id}`

Delete message.

`clusters:get_all`

Default
`rule:admin_api`

Operations

- **GET** `/clusters`
- **GET** `/clusters/detail`

List clusters.

`clusters:get`

Default
`rule:admin_api`

Operations

- **GET** `/clusters/{cluster_id}`

Show cluster.

`clusters:update`

Default
`rule:admin_api`

Operations

- **PUT** `/clusters/{cluster_id}`

Update cluster.

`workers:cleanup`

Default
`rule:admin_api`

Operations

- **POST** `/workers/cleanup`

Clean up workers.

`volume:get_snapshot_metadata`

Default
`rule:xena_system_admin_or_project_reader`

Operations

- **GET** `/snapshots/{snapshot_id}/metadata`
- **GET** `/snapshots/{snapshot_id}/metadata/{key}`

Show snapshots metadata or one specified metadata with a given key.

`volume:update_snapshot_metadata`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **POST** `/snapshots/{snapshot_id}/metadata`
- **PUT** `/snapshots/{snapshot_id}/metadata/{key}`

Update snapshots metadata or one specified metadata with a given key.

`volume:delete_snapshot_metadata`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **DELETE** `/snapshots/{snapshot_id}/metadata/{key}`

Delete snapshots specified metadata with a given key.

`volume:get_all_snapshots`

Default
`rule:xena_system_admin_or_project_reader`

Operations

- **GET** `/snapshots`
- **GET** `/snapshots/detail`

List snapshots.

`volume_extension:extended_snapshot_attributes`

Default
`rule:xena_system_admin_or_project_reader`

Operations

- **GET** `/snapshots/{snapshot_id}`
- **GET** `/snapshots/detail`

List or show snapshots with extended attributes.

`volume:create_snapshot`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **POST** `/snapshots`

Create snapshot.

`volume:get_snapshot`

Default
`rule:xena_system_admin_or_project_reader`

Operations

- **GET** `/snapshots/{snapshot_id}`

Show snapshot.

`volume:update_snapshot`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **PUT** `/snapshots/{snapshot_id}`

Update snapshot.

`volume:delete_snapshot`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **DELETE** `/snapshots/{snapshot_id}`

Delete snapshot.

`volume_extension:snapshot_admin_actions:reset_status`

Default
`rule:admin_api`

Operations

- **POST** `/snapshots/{snapshot_id}/action (os-reset_status)`

Reset status of a snapshot.

`snapshot_extension:snapshot_actions:update_snapshot_status`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **POST** `/snapshots/{snapshot_id}/action (update_snapshot_status)`

Update database fields of snapshot.

`volume_extension:snapshot_admin_actions:force_delete`

Default
`rule:admin_api`

Operations

- **POST** `/snapshots/{snapshot_id}/action (os-force_delete)`

Force delete a snapshot.

`snapshot_extension:list_manageable`

Default
`rule:admin_api`

Operations

- **GET** `/manageable_snapshots`
- **GET** `/manageable_snapshots/detail`

List (in detail) of snapshots which are available to manage.

`snapshot_extension:snapshot_manage`

Default
`rule:admin_api`

Operations

- **POST** `/manageable_snapshots`

Manage an existing snapshot.

`snapshot_extension:snapshot_unmanage`

Default
`rule:admin_api`

Operations

- **POST** `/snapshots/{snapshot_id}/action (os-unmanage)`

Stop managing a snapshot.

`backup:get_all`

Default
`rule:xena_system_admin_or_project_reader`

Operations

- **GET** `/backups`
- **GET** `/backups/detail`

List backups.

`backup:backup_project_attribute`

Default
`rule:admin_api`

Operations

- **GET** `/backups/{backup_id}`
- **GET** `/backups/detail`

List backups or show backup with project attributes.

`backup:create`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **POST** `/backups`

Create backup.

`backup:get`

Default
`rule:xena_system_admin_or_project_reader`

Operations

- **GET** `/backups/{backup_id}`

Show backup.

`backup:update`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **PUT** `/backups/{backup_id}`

Update backup.

`backup:delete`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **DELETE** `/backups/{backup_id}`

Delete backup.

`backup:restore`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **POST** `/backups/{backup_id}/restore`

Restore backup.

`backup:backup-import`

Default
`rule:admin_api`

Operations

- **POST** `/backups/{backup_id}/import_record`

Import backup.

`backup:export-import`

Default
`rule:admin_api`

Operations

- **POST** `/backups/{backup_id}/export_record`

Export backup.

`volume_extension:backup_admin_actions:reset_status`

Default
`rule:admin_api`

Operations

- **POST** `/backups/{backup_id}/action (os-reset_status)`

Reset status of a backup.

`volume_extension:backup_admin_actions:force_delete`

Default
`rule:admin_api`

Operations

- **POST** `/backups/{backup_id}/action (os-force_delete)`

Force delete a backup.

`group:get_all`

Default
`rule:xena_system_admin_or_project_reader`

Operations

- **GET** `/groups`
- **GET** `/groups/detail`

List groups.

`group:create`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **POST** `/groups`

Create group.

`group:get`

Default
`rule:xena_system_admin_or_project_reader`

Operations

- **GET** `/groups/{group_id}`

Show group.

`group:update`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **PUT** `/groups/{group_id}`

Update group.

`group:group_project_attribute`

Default
`rule:admin_api`

Operations

- **GET** `/groups/{group_id}`
- **GET** `/groups/detail`

List groups or show group with project attributes.

`group:group_types:create`

Default
`rule:admin_api`

Operations

- **POST** `/group_types/`

Create a group type.

`group:group_types:update`

Default
`rule:admin_api`

Operations

- **PUT** `/group_types/{group_type_id}`

Update a group type.

`group:group_types:delete`

Default
`rule:admin_api`

Operations

- **DELETE** `/group_types/{group_type_id}`

Delete a group type.

`group:access_group_types_specs`

Default
`rule:admin_api`

Operations

- **GET** `/group_types/{group_type_id}`

Show group type with type specs attributes.

`group:group_types_specs:get`

Default
`rule:admin_api`

Operations

- **GET** `/group_types/{group_type_id}/group_specs/{g_spec_id}`

Show a group type spec.

`group:group_types_specs:get_all`

Default
`rule:admin_api`

Operations

- **GET** `/group_types/{group_type_id}/group_specs`

List group type specs.

`group:group_types_specs:create`

Default
`rule:admin_api`

Operations

- **POST** `/group_types/{group_type_id}/group_specs`

Create a group type spec.

`group:group_types_specs:update`

Default
`rule:admin_api`

Operations

- **PUT** `/group_types/{group_type_id}/group_specs/{g_spec_id}`

Update a group type spec.

`group:group_types_specs:delete`

Default
`rule:admin_api`

Operations

- **DELETE** `/group_types/{group_type_id}/group_specs/{g_spec_id}`

Delete a group type spec.

`group:get_all_group_snapshots`

Default
`rule:xena_system_admin_or_project_reader`

Operations

- **GET** `/group_snapshots`

- **GET** `/group_snapshots/detail`

List group snapshots.

`group:create_group_snapshot`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **POST** `/group_snapshots`

Create group snapshot.

`group:get_group_snapshot`

Default
`rule:xena_system_admin_or_project_reader`

Operations

- **GET** `/group_snapshots/{group_snapshot_id}`

Show group snapshot.

`group:delete_group_snapshot`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **DELETE** `/group_snapshots/{group_snapshot_id}`

Delete group snapshot.

`group:update_group_snapshot`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **PUT** `/group_snapshots/{group_snapshot_id}`

Update group snapshot.

`group:group_snapshot_project_attribute`

Default
`rule:admin_api`

Operations

- **GET** `/group_snapshots/{group_snapshot_id}`
- **GET** `/group_snapshots/detail`

List group snapshots or show group snapshot with project attributes.

`group:reset_group_snapshot_status`

Default
`rule:admin_api`

Operations

- **POST** `/group_snapshots/{g_snapshot_id}/action (reset_status)`

Reset status of group snapshot.

`group:delete`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **POST** `/groups/{group_id}/action (delete)`

Delete group.

`group:reset_status`

Default
`rule:admin_api`

Operations

- **POST** `/groups/{group_id}/action (reset_status)`

Reset status of group.

`group:enable_replication`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **POST** `/groups/{group_id}/action (enable_replication)`

Enable replication.

`group:disable_replication`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **POST** `/groups/{group_id}/action (disable_replication)`

Disable replication.

`group:failover_replication`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **POST** `/groups/{group_id}/action (failover_replication)`

Fail over replication.

`group:list_replication_targets`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **POST** `/groups/{group_id}/action (list_replication_targets)`

List failover replication.

`volume_extension:qos_specs_manage:get_all`

Default
`rule:admin_api`

Operations

- **GET** `/qos-specs`
- **GET** `/qos-specs/{qos_id}/associations`

List qos specs or list all associations.

`volume_extension:qos_specs_manage:get`

Default
`rule:admin_api`

Operations

- **GET** `/qos-specs/{qos_id}`

Show qos specs.

`volume_extension:qos_specs_manage:create`

Default
`rule:admin_api`

Operations

- **POST** `/qos-specs`

Create qos specs.

`volume_extension:qos_specs_manage:update`

Default
`rule:admin_api`

Operations

- **PUT** `/qos-specs/{qos_id}`
- **GET** `/qos-specs/{qos_id}/disassociate_all`
- **GET** `/qos-specs/{qos_id}/associate`
- **GET** `/qos-specs/{qos_id}/disassociate`

Update qos specs (including updating association).

`volume_extension:qos_specs_manage:delete`

Default
`rule:admin_api`

Operations

- **DELETE** `/qos-specs/{qos_id}`

- **PUT** `/qos-specs/{qos_id}/delete_keys`

delete qos specs or unset one specified qos key.

`volume_extension:quota_classes:get`

Default
`rule:admin_api`

Operations

- **GET** `/os-quota-class-sets/{project_id}`

Show project quota class.

`volume_extension:quota_classes:update`

Default
`rule:admin_api`

Operations

- **PUT** `/os-quota-class-sets/{project_id}`

Update project quota class.

`volume_extension:quotas:show`

Default
`rule:xena_system_admin_or_project_reader`

Operations

- **GET** `/os-quota-sets/{project_id}`
- **GET** `/os-quota-sets/{project_id}/default`
- **GET** `/os-quota-sets/{project_id}?usage=True`

Show project quota (including usage and default).

`volume_extension:quotas:update`

Default
`rule:admin_api`

Operations

- **PUT** `/os-quota-sets/{project_id}`

Update project quota.

`volume_extension:quotas:delete`

Default
`rule:admin_api`

Operations

- **DELETE** `/os-quota-sets/{project_id}`

Delete project quota.

`volume_extension:capabilities`

Default
`rule:admin_api`

Operations

- **GET** `/capabilities/{host_name}`

Show backend capabilities.

`volume_extension:services:index`

Default
`rule:admin_api`

Operations

- **GET** `/os-services`

List all services.

`volume_extension:services:update`

Default
`rule:admin_api`

Operations

- **PUT** `/os-services/{action}`

Update service, including failover_host, thaw, freeze, disable, enable, set-log and get-log actions.

`volume:freeze_host`

Default
`rule:admin_api`

Operations

- **PUT** `/os-services/freeze`

Freeze a backend host.

`volume:thaw_host`

Default
`rule:admin_api`

Operations

- **PUT** `/os-services/thaw`

Thaw a backend host.

`volume:failover_host`

Default
`rule:admin_api`

Operations

- **PUT** `/os-services/failover_host`

Failover a backend host.

`scheduler_extension:scheduler_stats:get_pools`

Default
`rule:admin_api`

Operations

- **GET** `/scheduler-stats/get_pools`

List all backend pools.

`volume_extension:hosts`

Default
`rule:admin_api`

Operations

- **GET** `/os-hosts`
- **PUT** `/os-hosts/{host_name}`
- **GET** `/os-hosts/{host_id}`

List, update or show hosts for a project.

`limits_extension:used_limits`

Default
`rule:xena_system_admin_or_project_reader`

Operations

- **GET** `/limits`

Show limits with used limit attributes.

`volume_extension:list_manageable`

Default
`rule:admin_api`

Operations

- **GET** `/manageable_volumes`
- **GET** `/manageable_volumes/detail`

List (in detail) of volumes which are available to manage.

`volume_extension:volume_manage`

Default
`rule:admin_api`

Operations

- **POST** `/manageable_volumes`

Manage existing volumes.

`volume_extension:volume_unmanage`

Default
`rule:admin_api`

Operations

- **POST** `/volumes/{volume_id}/action (os-unmanage)`

Stop managing a volume.

`volume_extension:type_create`

Default
`rule:admin_api`

Operations

- **POST** `/types`

Create volume type.

`volume_extension:type_update`

Default
`rule:admin_api`

Operations

- **PUT** `/types`

Update volume type.

`volume_extension:type_delete`

Default
`rule:admin_api`

Operations

- **DELETE** `/types`

Delete volume type.

`volume_extension:type_get`

Default
`rule:xena_system_admin_or_project_reader`

Operations

- **GET** `/types/{type_id}`

Get one specific volume type.

`volume_extension:type_get_all`

Default
`rule:xena_system_admin_or_project_reader`

Operations

- **GET** `/types/`

List volume types.

`volume_extension:access_types_extra_specs`

Default
`rule:xena_system_admin_or_project_reader`

Operations

- **GET** /types/{type_id}
- **GET** /types

Include the volume types extra_specs attribute in the volume type list or show requests. The ability to make these calls is governed by other policies.

`volume_extension:access_types_qos_specs_id`

Default
rule:admin_api

Operations

- **GET** /types/{type_id}
- **GET** /types

Include the volume types QoS specifications ID attribute in the volume type list or show requests. The ability to make these calls is governed by other policies.

`volume_extension:volume_type_encryption`

Default
rule:admin_api

DEPRECATED: This rule will be removed in the Yoga release.

`volume_extension:volume_type_encryption:create`

Default
rule:admin_api

Operations

- **POST** /types/{type_id}/encryption

Create volume type encryption.

`volume_extension:volume_type_encryption:get`

Default
rule:admin_api

Operations

- **GET** /types/{type_id}/encryption
- **GET** /types/{type_id}/encryption/{key}

Show a volume types encryption type, show an encryption specs item.

`volume_extension:volume_type_encryption:update`

Default
rule:admin_api

Operations

- **PUT** /types/{type_id}/encryption/{encryption_id}

Update volume type encryption.

`volume_extension:volume_type_encryption:delete`

Default
`rule:admin_api`

Operations

- **DELETE** `/types/{type_id}/encryption/{encryption_id}`

Delete volume type encryption.

`volume_extension:volume_type_access`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **GET** `/types`
- **GET** `/types/{type_id}`
- **POST** `/types`

Adds the boolean field os-volume-type-access:is_public to the responses for these API calls. The ability to make these calls is governed by other policies.

`volume_extension:volume_type_access:addProjectAccess`

Default
`rule:admin_api`

Operations

- **POST** `/types/{type_id}/action (addProjectAccess)`

Add volume type access for project.

`volume_extension:volume_type_access:removeProjectAccess`

Default
`rule:admin_api`

Operations

- **POST** `/types/{type_id}/action (removeProjectAccess)`

Remove volume type access for project.

`volume_extension:volume_type_access:get_all_for_type`

Default
`rule:admin_api`

Operations

- **GET** `/types/{type_id}/os-volume-type-access`

List private volume type access detail, that is, list the projects that have access to this volume type.

`volume:extend`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **POST** `/volumes/{volume_id}/action (os-extend)`

Extend a volume.

`volume:extend_attached_volume`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **POST** `/volumes/{volume_id}/action (os-extend)`

Extend a attached volume.

`volume_extension:volume_admin_actions:extend_volume_completion`

Default
`rule:admin_api`

Operations

- **POST** `/volumes/{volume_id}/action (os-extend_volume_completion)`

Complete a volume extend operation.

`volume:revert_to_snapshot`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **POST** `/volumes/{volume_id}/action (revert)`

Revert a volume to a snapshot.

`volume_extension:volume_admin_actions:reset_status`

Default
`rule:admin_api`

Operations

- **POST** `/volumes/{volume_id}/action (os-reset_status)`

Reset status of a volume.

`volume:retype`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **POST** `/volumes/{volume_id}/action (os-retype)`

Retype a volume.

`volume:update_readonly_flag`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **POST** `/volumes/{volume_id}/action (os-update_readonly_flag)`

Update a volumes readonly flag.

`volume_extension:volume_admin_actions:force_delete`

Default
`rule:admin_api`

Operations

- **POST** `/volumes/{volume_id}/action (os-force_delete)`

Force delete a volume.

`volume_extension:volume_actions:upload_public`

Default
`rule:admin_api`

Operations

- **POST** `/volumes/{volume_id}/action (os-volume_upload_image)`

Upload a volume to image with public visibility.

`volume_extension:volume_actions:upload_image`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **POST** `/volumes/{volume_id}/action (os-volume_upload_image)`

Upload a volume to image.

`volume_extension:volume_admin_actions:force_detach`

Default
`rule:admin_api`

Operations

- **POST** `/volumes/{volume_id}/action (os-force_detach)`

Force detach a volume.

`volume_extension:volume_admin_actions:migrate_volume`

Default
`rule:admin_api`

Operations

- **POST** `/volumes/{volume_id}/action (os-migrate_volume)`

migrate a volume to a specified host.

`volume_extension:volume_admin_actions:migrate_volume_completion`

Default
`rule:admin_api`

Operations

- **POST** `/volumes/{volume_id}/action (os-migrate_volume_completion)`

Complete a volume migration.

`volume_extension:volume_actions:initialize_connection`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **POST** `/volumes/{volume_id}/action (os-initialize_connection)`

Initialize volume attachment.

`volume_extension:volume_actions:terminate_connection`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **POST** `/volumes/{volume_id}/action (os-terminate_connection)`

Terminate volume attachment.

`volume_extension:volume_actions:roll_detaching`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **POST** `/volumes/{volume_id}/action (os-roll_detaching)`

Roll back volume status to in-use.

`volume_extension:volume_actions:reserve`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **POST** `/volumes/{volume_id}/action (os-reserve)`

Mark volume as reserved.

`volume_extension:volume_actions:unreserve`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **POST** `/volumes/{volume_id}/action (os-unreserve)`

Unmark volume as reserved.

`volume_extension:volume_actions:begin_detaching`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **POST** `/volumes/{volume_id}/action (os-begin_detaching)`

Begin detach volumes.

`volume_extension:volume_actions:attach`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **POST** `/volumes/{volume_id}/action (os-attach)`

Add attachment metadata.

`volume_extension:volume_actions:detach`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **POST** `/volumes/{volume_id}/action (os-detach)`

Clear attachment metadata.

`volume:reimage`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **POST** `/volumes/{volume_id}/action (os-reimage)`

Reimage a volume in available or error status.

`volume:reimage_reserved`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **POST** `/volumes/{volume_id}/action (os-reimage)`

Reimage a volume in reserved status.

`volume:get_all_transfers`

Default
`rule:xena_system_admin_or_project_reader`

Operations

- **GET** `/os-volume-transfer`
- **GET** `/os-volume-transfer/detail`
- **GET** `/volume_transfers`
- **GET** `/volume-transfers/detail`

List volume transfer.

`volume:create_transfer`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **POST** `/os-volume-transfer`
- **POST** `/volume_transfers`

Create a volume transfer.

`volume:get_transfer`

Default
`rule:xena_system_admin_or_project_reader`

Operations

- **GET** `/os-volume-transfer/{transfer_id}`
- **GET** `/volume-transfers/{transfer_id}`

Show one specified volume transfer.

`volume:accept_transfer`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **POST** `/os-volume-transfer/{transfer_id}/accept`
- **POST** `/volume-transfers/{transfer_id}/accept`

Accept a volume transfer.

`volume:delete_transfer`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **DELETE** `/os-volume-transfer/{transfer_id}`
- **DELETE** `/volume-transfers/{transfer_id}`

Delete volume transfer.

`volume:get_volume_metadata`

Default
`rule:xena_system_admin_or_project_reader`

Operations

- **GET** `/volumes/{volume_id}/metadata`
- **GET** `/volumes/{volume_id}/metadata/{key}`
- **POST** `/volumes/{volume_id}/action (os-show_image_metadata)`

Show volumes metadata or one specified metadata with a given key.

`volume:create_volume_metadata`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **POST** `/volumes/{volume_id}/metadata`

Create volume metadata.

`volume:update_volume_metadata`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **PUT** `/volumes/{volume_id}/metadata`
- **PUT** `/volumes/{volume_id}/metadata/{key}`

Replace a volumes metadata dictionary or update a single metadatum with a given key.

`volume:delete_volume_metadata`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **DELETE** `/volumes/{volume_id}/metadata/{key}`

Delete a volumes metadatum with the given key.

`volume_extension:volume_image_metadata:show`

Default
`rule:xena_system_admin_or_project_reader`

Operations

- **GET** `/volumes/detail`
- **GET** `/volumes/{volume_id}`

Include a volumes image metadata in volume detail responses. The ability to make these calls is governed by other policies.

`volume_extension:volume_image_metadata:set`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **POST** `/volumes/{volume_id}/action (os-set_image_metadata)`

Set image metadata for a volume

`volume_extension:volume_image_metadata:remove`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **POST** `/volumes/{volume_id}/action (os-unset_image_metadata)`

Remove specific image metadata from a volume

`volume:update_volume_admin_metadata`

Default
`rule:admin_api`

Operations

- **POST** `/volumes/{volume_id}/action (os-update_readonly_flag)`
- **POST** `/volumes/{volume_id}/action (os-attach)`

Update volume admin metadata. This permission is required to complete these API calls, though the ability to make these calls is governed by other policies.

`volume_extension:types_extra_specs:index`

Default
`rule:xena_system_admin_or_project_reader`

Operations

- **GET** `/types/{type_id}/extra_specs`

List type extra specs.

`volume_extension:types_extra_specs:create`

Default
`rule:admin_api`

Operations

- **POST** `/types/{type_id}/extra_specs`

Create type extra specs.

`volume_extension:types_extra_specs:show`

Default
`rule:xena_system_admin_or_project_reader`

Operations

- **GET** `/types/{type_id}/extra_specs/{extra_spec_key}`

Show one specified type extra specs.

`volume_extension:types_extra_specs:read_sensitive`

Default
`rule:admin_api`

Operations

- **GET** `/types`
- **GET** `/types/{type_id}`
- **GET** `/types/{type_id}/extra_specs`
- **GET** `/types/{type_id}/extra_specs/{extra_spec_key}`

Include extra_specs fields that may reveal sensitive information about the deployment that should not be exposed to end users in various volume-type responses that show extra_specs. The ability to make these calls is governed by other policies.

`volume_extension:types_extra_specs:update`

Default
`rule:admin_api`

Operations

- **PUT** `/types/{type_id}/extra_specs/{extra_spec_key}`

Update type extra specs.

`volume_extension:types_extra_specs:delete`

Default
`rule:admin_api`

Operations

- **DELETE** `/types/{type_id}/extra_specs/{extra_spec_key}`

Delete type extra specs.

`volume:create`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **POST** `/volumes`

Create volume.

`volume:create_from_image`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **POST** `/volumes`

Create volume from image.

`volume:get`

Default
`rule:xena_system_admin_or_project_reader`

Operations

- **GET** `/volumes/{volume_id}`

Show volume.

`volume:get_all`

Default
`rule:xena_system_admin_or_project_reader`

Operations

- **GET** /volumes
- **GET** /volumes/detail
- **GET** /volumes/summary

List volumes or get summary of volumes.

volume:update

Default
rule:xena_system_admin_or_project_member

Operations

- **PUT** /volumes
- **POST** /volumes/{volume_id}/action (os-set_bootable)

Update volume or update a volumes bootable status.

volume:delete

Default
rule:xena_system_admin_or_project_member

Operations

- **DELETE** /volumes/{volume_id}

Delete volume.

volume:force_delete

Default
rule:admin_api

Operations

- **DELETE** /volumes/{volume_id}

Force Delete a volume.

volume_extension:volume_host_attribute

Default
rule:admin_api

Operations

- **GET** /volumes/{volume_id}
- **GET** /volumes/detail

List or show volume with host attribute.

volume_extension:volume_tenant_attribute

Default
rule:xena_system_admin_or_project_reader

Operations

- **GET** /volumes/{volume_id}
- **GET** /volumes/detail

List or show volume with tenant attribute.

`volume_extension:volume_mig_status_attribute`

Default
`rule:admin_api`

Operations

- **GET** `/volumes/{volume_id}`
- **GET** `/volumes/detail`

List or show volume with migration status attribute.

`volume_extension:volume_encryption_metadata`

Default
`rule:xena_system_admin_or_project_reader`

Operations

- **GET** `/volumes/{volume_id}/encryption`
- **GET** `/volumes/{volume_id}/encryption/{encryption_key}`

Show volumes encryption metadata.

`volume:multiattach`

Default
`rule:xena_system_admin_or_project_member`

Operations

- **POST** `/volumes`

Create multiattach capable volume.

`volume_extension:default_set_or_update`

Default
`rule:admin_api`

Operations

- **PUT** `/default-types`

Set or update default volume type.

`volume_extension:default_get`

Default
`rule:admin_api`

Operations

- **GET** `/default-types/{project-id}`

Get default types.

`volume_extension:default_get_all`

Default
`rule:admin_api`

Operations

- **GET** `/default-types/`

Get all default types. WARNING: Changing this might open up too much information regarding cloud deployment.

`volume_extension:default_unset`

Default
`rule:admin_api`

Operations

- **DELETE** `/default-types/{project-id}`

Unset default type.

Policy configuration HowTo

You can use Cinder policies to control how your users and administrators interact with the Block Storage Service. In this HowTo, well discuss the user model Cinder employs and how it can be modified by adjusting policies.

- Like most OpenStack services, Cinder uses the OpenStack `oslo.policy` library as a base for its policy-related code. For a discussion of rules and roles, other vocabulary, and general information about OpenStack policies and the policy configuration file, see Administering Applications that use oslo.policy.
- See *Policy configuration* for the list of policy targets recognized by Cinder.
- Since the Queens release, the default way to run Cinder is without a policy file. This is because sensible default values are defined in the code. To run Cinder with a custom policy configuration, however, youll need to write your changes into a policy file.
- Instructions for generating a sample `policy.yaml` file directly from the Cinder source code can be found in the file `README-policy.generate.md` in the `etc/cinder` directory in the Cinder source code repository (or its github mirror).
- OpenStack has deprecated the use of a JSON policy file since the Wallaby release (Cinder 18.0.0). If you are still using the JSON format, there is a oslopolicy-convert-json-to-yaml tool that will migrate your existing JSON-formatted policy file to YAML in a backward-compatible way.

Vocabulary Note

We need to clarify some terms well be using below.

Project
This is an administrative grouping of users into a unit that can own cloud resources. (This is what used to be called a tenant.)

Service
This is an OpenStack component that users interact with through an API it provides. For example, Cinder is the OpenStack code name for the service that provides the Block Storage API versions 2 and 3. Cinder is also known as the OpenStack Block Storage Service.

The point of making this distinction is that theres another use of the term project that is relevant to the discussion, but that were **not** going to use. Each OpenStack service is produced and maintained by a project team. *We will not be using the term project in that sense in this document. Well always use the* *term service*. (If you are new to OpenStack, this wont be a problem. But if youre discussing this content with someone whos been around OpenStack for a while, youll want to be clear about this so that youre not talking past each other.)

The User Model

The Cinder code is written with the expectation that there are two kinds of users.

End users
These are users who consume resources and (possibly) pay the bills. End users are restricted to acting within a specific project and cannot perform operations on resources that are not owned by the project(s) they are in.

Administrative users (admins)
These are users who keep the lights on. They have the ability to view all resources controlled by Cinder and can perform most operations on them. They also have access to other operations (for example, setting quotas) that cannot be performed by end users.

Additionally, admins can view resource properties that cannot be seen by end users (for example, the migration status of a volume). The technical term to describe this is that when a volume-show call is made in an *administrative context* it will contain additional properties than when the call is *not* made in an administrative context. Similarly, when a volume-list call is made in an administrative context, the response may include volumes that are not owned by the project of the person making the call; this never happens when a call is *not* made in an administrative context.

Policies

Broadly speaking, an operator can accomplish two things with policies:

1. The policy file can define the criteria for what users are granted the privilege to act in an administrative context.
2. The policy file can specify for specific *actions* (or *policy targets*), which users can perform those actions.

In general, while an operator can define *who* can make calls in an administrative context, an operator cannot affect *what* can be done in an administrative context (because thats already been decided when the code was implemented). For example, the boundaries between projects are strictly enforced in Cinder, and only an admin can view resources across projects. There is no way to grant a user the ability to see into another project (at least not by policy configurationthis could be done by using the Identity Service to add the user to the other project, but note that at that point, the user is no longer *not* a member of the project owning the now visible resources.)

Pre-Defined Policy Rules

The default Cinder policy file contains three rules that are used as the basis of policy file configuration.

context_is_admin
This defines the administrative context in Cinder. Youll notice that its defined once at the beginning of the sample policy file and isnt referred to anywhere else in that file. To understand what this does, its helpful to know something about the API implementation.

A users API request must be accompanied by an authentication token from the Identity Service. (If you are using client software, for example, the python-cinderclient or python-openstack client, the token is being requested for you under the hood.) The Block Storage API confirms that the token is unexpired and obtains other information about the requestor, for example, what roles the

Identity Service recognizes the user to have. Cinder uses this information to create an internal context object that will be passed around the code as various functions and services are called to satisfy the users request.

When the request context object is created, Cinder uses the context_is_admin rule to decide whether this context object will be recognized as providing an administrative context. It does this by setting the is_admin property to True on the context object. Cinder code later in the call chain simply checks whether the is_admin property is true on the context object to determine whether the call is taking place in an administrative context. Similarly, policies will refer to is_admin:True (either directly or indirectly) to require an administrative context.

All of this is a long-winded way to say that in a Cinder policy file, youll only see context_is_admin at the top; after that, youll see is_admin:True whenever you want to refer to an administrative context.

admin_or_owner
This is the default rule for most non-admin API calls. As the name indicates, it allows an administrator or an owner to make the call.

admin_api
This is the default rule for API calls that only administrators should be allowed to make.

> **Note**
>
> For some API calls, there are checks way down in the code to ensure that a call is being made in an administrative context before the request is allowed to succeed. Thus it is not always the case that simply changing a policy target whose value is rule:admin_api to rule:admin_or_owner (or rule:admin_api or role:some-special-role) will give a non-admin user the ability to successfully make the call. Unfortunately, you cant tell which calls these are without experimenting with a policy file (or looking at the source code). A good rule of thumb, however, is that API calls governed by policies marked as rule:admin_api in the default policy configuration fall into this category.

Example: Configuring a Read-Only Administrator

A fairly common configuration request is to create a special category of administrator who has only an *observer* (look but dont touch) function. The idea is that for security and stability reasons, its a good idea to allow all users, including administrators, the least amount of privileges they need to successfully perform their job. Someone whose job is to audit information about Cinder (for example, to see what the current quota settings are) doesnt need the ability to change these settings. In this section, well discuss one way to configure the Cinder policy file to accomplish this.

> **Note**
>
> To keep the discussion focused, this example assumes that youre working from the default policy file. Hopefully the general strategy will be clear enough to be applied to clouds already using non-default configurations. Additionally, there are other logically equivalent ways to configure the policy file to introduce a read-only administrator; this is not by any means the only way to do it.

Given the job requirements, the observer administrator (who well refer to as the observer-admin for short) needs to operate in the administrative context. Thus, well have to adjust the context_is_admin definition in the policy file to include such a person. Note that this will make such a person a **full administrator** if we make no other changes to the policy file. Thus the strategy well use is to first make the observer-admin a full administrator, and then block the observer-admins access to those API calls that arent read-only.

Warning

Metaphorically, what we are doing is opening the floodgates and then plugging up the holes one by one. That sounds alarming, and it should. We cannot emphasize strongly enough that any policy file changes should be **well-contained** (that is, you know exactly who has the new role or roles) and **tested** (you should have some kind of tests in place to determine that your changes have only the effects you intend).

This is probably as good a place as any to remind you that the suggestions that follow are provided without warranty of any kind, either expressed or implied. Like the OpenStack source code, they are covered by the Apache License, version 2.0. In particular, we direct your attention to sections 7-9.

Step 0: Testing

We mention testing first (even though you havent made any changes yet) because if we wait to mention it until after weve made the configuration changes, you might get the impression that its the last thing to do (or the least important). It will make your life much easier if you come up with a plan for how you will test these changes before you start modifiying the policy configuration.

We advise setting up automated tests because the Block Storage API has a lot of API calls and youll want to test each of them against an admin user, an observer-admin user, and a regular end user. Further, if you anticipate that you may require finer-grained access than outlined in this example (for example, you would like a creator role that can create and read, but not delete), your configuration will be all the more complex and hence require more extensive testing that you wont want to do by hand.

Step 1: Create a new role

In the Identity Service, create a new role. Its a good idea to make this a new, never before assigned role so that you can easily track who its been assigned to. As you recall from the discussion above, this person will have **full administrative powers** for any functions that are missed when we do the block up the holes stage.

For this example, well use a role named `cinder:reader-admin`. There is nothing special about this role name; you may use any name that makes sense to the administrators who will be assigning the role and configuring the policies. (The cinder: part is to remind you that this role applies to the Block Storage Service, the reader part is from the role name that OpenStack has converged upon for this type of observer role, and the -admin part is to remind you that whoever has this role will be able to observe admin-type stuff.)

Note

Beginning with the Rocky release, the Identity Service (Keystone) creates three roles when the service is initiated: `member`, `reader`, and `admin`. By default, the `reader` role is not assigned to any users. Work is underway during the Stein cycle so that the Identity API will recognize users with the `reader` role as having read-only access to the Identity API. See the Keystone spec Basic Default Roles for more information.

We mention this so that you are aware that if you use a role named `reader` when doing the policy configuration described in this document, at some point users assigned the `reader` role may have read-only access to services other than the Block Storage Service. The desirability of this outcome depends upon your particular use case.

Step 2: Open the floodgates

If your installation doesnt have an `/etc/cinder/policy.yaml` file, you can generate one from the source code (see the introductory section of this document).

Note

The default file is *completely commented out*. For any of the changes you make below to be effective, dont forget to *uncomment* the line in which they occur.

To extend the administrative context to include the new role, change:

```
"context_is_admin": "role:admin"
```

to:

```
"context_is_admin": "role:admin or role:cinder:reader-admin"
```

Step 3: Plug the holes in the Admin API

Now we make adjustments to the policy configuration so that the observer-admin will in fact have only read-only access to Cinder resources.

3A: New Policy Rule

First, we create a new policy rule for Admin API access that specifically excludes the new role. Find the line in the policy file that has `"admin_api"` on the left hand side. Immediately after it, introduce a new rule:

```
"strict_admin_api": "not role:cinder:reader-admin and rule:admin_api"
```

3B: Plugging Holes

Now, plug up the holes weve opened in the Admin API by using this new rule. Find each of the lines in the remainder of the policy file that look like:

```
"target": "rule:admin_api"
```

and for each line, decide whether the observer-admin needs access to this action or not. For example, the target `"volume_extension:services:index"` specifies a read-only action, so its appropriate for the observer-admin to perform. Well leave that one in its default configuration of:

```
"volume_extension:services:index": "rule:admin_api"
```

On the other hand, if the target is something that allows modification, we most likely dont want to allow the observer-admin to perform it. For such actions we need to use the strict form of the admin rule. For example, consider the action "volume_extension:quotas:delete". To exclude the observer-admin from performing it, change the default setting of:

```
"volume_extension:quotas:delete": "rule:admin_api"
```

to:

```
"volume_extension:quotas:delete": "rule:strict_admin_api"
```

Do this on a case-by-case basis for the other policy targets that by default are governed by the rule:admin_api.

3C: Other Changes

Youve probably figured this out already, but there may be some other changes that are implied by, but not explicitly mentioned in, the above instructions. For example, youll find the following policies in the sample file:

```
"volume_extension:volume_type_encryption": "rule:admin_api"
"volume_extension:volume_type_encryption:create": "rule:volume_
↪extension:volume_type_encryption"
"volume_extension:volume_type_encryption:get": "rule:volume_extension:volume_
↪type_encryption"
"volume_extension:volume_type_encryption:update": "rule:volume_
↪extension:volume_type_encryption"
"volume_extension:volume_type_encryption:delete": "rule:volume_
↪extension:volume_type_encryption"
```

The first policy covers all of create/read/update/delete (and is deprecated for removal during the Stein development cycle). However, if you set it to "rule:strict_admin_api", the observer-admin wont be able to read the volume type encryption. So it should be left at "rule:admin_api" and the create/update/delete policies should be changed to "rule:strict_admin_api". Additionally, in preparation for the deprecated policy targets removal, its a good idea to change the value of the get policy to "rule:admin_api".

Step 4: Plug the holes in the Regular API

As stated earlier, a user with the role cinder:reader-admin is elevated to full administrative powers. That implies that such a user can perform administrative functions on end-user resources. Hence, we have another set of holes to plug up.

4A: New Policy Rule

As we did for the Admin API, well create a strict version of the admin_or_owner rule so we can specifically exclude the observer-admin from executing that action. Find the line in the policy file where "admin_or_owner" appears on the left hand side. It probably looks something like this:

```
"admin_or_owner": "is_admin:True or (role:admin and is_admin_project:True) or␣
↪project_id:%(project_id)s"
```

Immediately following it, introduce a new rule:

```
"strict_admin_or_owner": "(not role:cinder:reader-admin and (is_admin:True or␣
↪(role:admin and is_admin_project:True))) or project_id:%(project_id)s"
```

Note

To understand what this change does, note that the admin_or_owner rule definition has the general structure:

```
<admin-stuff> or <project-stuff>
```

To construct the strict version, we need to make sure that the `not cinder:reader-admin` part applies only the left-hand side (the <admin-stuff>). The easiest way to do that is to structure the new rule as follows:

```
(not role:cinder:reader-admin and (<admin-stuff>)) or <project-stuff>
```

Note

If you dont need a user with the role `cinder:reader-admin` to manage resources in their own project, you could simplify this rule to:

```
"strict_admin_or_owner": "not role:cinder:reader-admin and rule:admin_or_
↪owner"
```

4B: Plugging Holes

Find each line in the policy file that looks like:

```
"target": "rule:admin_or_owner"
```

and decide whether it represents an action that the observer-admin needs to perform. For those actions you *dont* want the observer-admin to do, change the policy to:

```
"target": "rule:strict_admin_or_owner"
```

4C: Unrestricted Policies

There are some policies in the default file that look like this:

```
"target": ""
```

These are called *unrestricted policies* because the requirements are empty, and hence can be satisfied by any authenticated user. (Recall from the earlier discussion of *The User Model*, however, that this does *not* mean that any user can see any other users resources.)

Unrestricted policies may be found on GET calls that dont have a particular resource to refer to (for example, the call to get all volumes) or a POST call that creates a completely new resource (for example, the call to create a volume). You dont see them much in the Cinder policy file because the code implementing the Block Storage API v2 and v3 always make sure theres a target object containing at least the `project_id` and `user_id` that can be used in evaluating whether the policy should allow the action or not.

Thus, obvious read-only targets (for example, `volume_extension:type_get`) can be left unrestricted. Policy targets that are not read only (for example, `volume:accept_transfer`), can be changed to `rule:strict_admin_or_owner`.

Step 5: Testing

We emphasized above that because of the nature of this change, it is extremely important to test it carefully. One thing to watch out for: because were using a clause like `not role:cinder:reader-admin`, a typographical error in the role name will cause problems. (For example, if you enter it into the file as `not role:cinder_reader-admin`, it wont exclude the user were worried about, who has the role `cinder:reader-admin`.)

As mentioned earlier, we advise setting up automated tests so that you can prevent regressions if you have to modify your policy files at some point.

Fibre Channel Zone Manager

The Fibre Channel Zone Manager allows FC SAN Zone/Access control management in conjunction with Fibre Channel block storage. The configuration of Fibre Channel Zone Manager and various zone drivers are described in this section.

Configure Block Storage to use Fibre Channel Zone Manager

If Block Storage is configured to use a Fibre Channel volume driver that supports Zone Manager, update `cinder.conf` to add the following configuration options to enable Fibre Channel Zone Manager.

Make the following changes in the `/etc/cinder/cinder.conf` file under a `[fc-zone-manager]` section.

Table 111: Description of zoning configuration options

Configuration option = Default value	Description
`enable_unsupported_driver` = `False`	(Boolean) Set this to True when you want to allow an unsupported zone manager driver to start. Drivers that havent maintained a working CI system and testing are marked as unsupported until CI is working again. This also marks a driver as deprecated and may be removed in the next release.
`fc_fabric_names` = `None`	(String) Comma separated list of Fibre Channel fabric names. This list of names is used to retrieve other SAN credentials for connecting to each SAN fabric
`fc_san_lookup_service` = `cinder.zonemanager.drivers.brocade.brcd_fc_san_lookup_servic BrcdFCSanLookupService`	(String) FC SAN Lookup Service
`zone_driver` = `cinder.zonemanager.drivers.brocade.brcd_fc_zone_driver.BrcdFCZoneDriver`	(String) FC Zone Driver responsible for zone management
`zoning_policy` = `initiator-target`	(String) Zoning policy configured by user; valid values include initiator-target or initiator

To use different Fibre Channel Zone Drivers, use the parameters described in this section.

Note

When multi backend configuration is used, provide the `zoning_mode` configuration option as part of the volume driver configuration where `volume_driver` option is specified.

Note

Default value of `zoning_mode` is `None` and this needs to be changed to `fabric` to allow fabric zoning.

Note

`zoning_policy` can be configured as `initiator-target` or `initiator`

Brocade Fibre Channel Zone Driver

Brocade Fibre Channel Zone Driver performs zoning operations through HTTP, HTTPS, or SSH.

Warning

The Brocade Fibre Channel Zone Driver is being supported by the Cinder community on a best-effort basis. While it is tested with the first Release Candidate of each release, be aware that it is not continually tested by a third-party CI system. The driver was deprecated and marked as unsupported in the Ussuri release, and is subject to immediate removal if the maintenance burden exceeds the communitys capacity.

Set the following options in the `cinder.conf` configuration file under the `[fc-zone-manager]` section.

Table 112: Description of Brocade zoning manager configuration options

Configuration option = Default value	Description
`brcd_sb_connector` = HTTP	(String) South bound connector for zoning operation

Configure SAN fabric parameters under a section matching the name used in `fc_fabric_names` as described in the example below:

Table 113: Description of Brocade zoning fabrics configuration options

Configuration option = Default value	Description
`fc_fabric_address` = <>	(String) Management IP of fabric.
`fc_fabric_password` = <>	(String) Password for user.
`fc_fabric_port` = 22	(Port(min=0, max=65535)) Connecting port
`fc_fabric_ssh_cert_path` = <>	(String) Local SSH certificate Path.
`fc_fabric_user` = <>	(String) Fabric user ID.
`fc_southbound_protocol` = `REST_HTTP`	(String(choices=[SSH, HTTP, HTTPS, REST_HTTP, REST_HTTPS])) South bound connector for the fabric.
`fc_virtual_fabric_id` = `None`	(String) Virtual Fabric ID.
`zone_activate` = `True`	(Boolean) Overridden zoning activation state.
`zone_name_prefix` = `openstack`	(String) Overridden zone name prefix.
`zoning_policy` = `initiator-target`	(String) Overridden zoning policy.

Note

Define a fabric group for each fabric using the fabric names used in `fc_fabric_names` configuration option as group name.

Note

To define a fabric group for a switch which has Virtual Fabrics enabled, include the `fc_virtual_fabric_id` configuration option and `fc_southbound_protocol` configuration option set to `HTTP`, `HTTPS`, `REST_HTTP` or `REST_HTTPS` in the fabric group. Zoning on VF enabled fabric using `SSH` southbound protocol is not supported.

Note

On switches running Fabric OS v8.2.1 or greater, the use of the REST interface is recommended for southbound communication. Set the `fc_southbound_protocol` configuration option to `REST_HTTP` or `REST_HTTPS` in the fabric group.

System requirements

Brocade Fibre Channel Zone Driver requires firmware version FOS v6.4 or higher.

As a best practice for zone management, use a user account with `zoneadmin` role. Users with `admin` role (including the default `admin` user account) are limited to a maximum of two concurrent SSH sessions.

For information about how to manage Brocade Fibre Channel switches, see the Brocade Fabric OS user documentation.

Cisco Fibre Channel Zone Driver

Cisco Fibre Channel Zone Driver automates the zoning operations through SSH. Configure Cisco Zone Driver, Cisco Southbound connector, FC SAN lookup service and Fabric name.

Set the following options in the `cinder.conf` configuration file.

```
[fc-zone-manager]
zone_driver = cinder.zonemanager.drivers.cisco.cisco_fc_zone_driver.
↪CiscoFCZoneDriver
fc_san_lookup_service = cinder.zonemanager.drivers.cisco.cisco_fc_san_lookup_
↪service.CiscoFCSanLookupService
fc_fabric_names = CISCO_FABRIC_EXAMPLE
cisco_sb_connector = cinder.zonemanager.drivers.cisco.cisco_fc_zone_client_
↪cli.CiscoFCZoneClientCLI
```

Table 114: Description of Cisco zoning manager configuration options

Configuration option = Default value	Description
`cisco_sb_connector` = `cinder.zonemanager.drivers.cisco.cisco_fc_zone_client_cli.CiscoFCZoneClientCLI`	(String) Southbound connector for zoning operation

Configure SAN fabric parameters under a section matching the name used in `fc_fabric_names` as described in the example below:

Table 115: Description of Cisco zoning fabrics configuration options

Configuration option = Default value	Description
`cisco_fc_fabric_address` = `<>`	(String) Management IP of fabric
`cisco_fc_fabric_password` = `<>`	(String) Password for user
`cisco_fc_fabric_port` = `22`	(Port(min=0, max=65535)) Connecting port
`cisco_fc_fabric_user` = `<>`	(String) Fabric user ID
`cisco_zone_activate` = `True`	(Boolean) overridden zoning activation state
`cisco_zone_name_prefix` = `None`	(String) overridden zone name prefix
`cisco_zoning_policy` = `initiator-target`	(String) overridden zoning policy
`cisco_zoning_vsan` = `None`	(String) VSAN of the Fabric

Note

Define a fabric group for each fabric using the fabric names used in `fc_fabric_names` configuration option as group name.

The Cisco Fibre Channel Zone Driver supports basic and enhanced zoning modes.The zoning VSAN must exist with an active zone set name which is same as the `fc_fabric_names` option.

System requirements

Cisco MDS 9000 Family Switches.

Cisco MDS NX-OS Release 6.2(9) or later.

For information about how to manage Cisco Fibre Channel switches, see the Cisco MDS 9000 user documentation.

Volume encryption supported by the key manager

We recommend the Key management service (barbican) for storing encryption keys used by the OpenStack volume encryption feature. It can be enabled by updating `cinder.conf` and `nova.conf`.

Initial configuration

Configuration changes need to be made to any nodes running the `cinder-api` or `nova-compute` server.

Steps to update `cinder-api` servers:

1. Edit the `/etc/cinder/cinder.conf` file to use Key management service as follows:
   - Look for the `[key_manager]` section.
   - Enter a new line directly below `[key_manager]` with the following:

   ```
   backend = barbican
   ```

2. Restart `cinder-api`, `cinder-volume` and `cinder-backup`.

Update `nova-compute` servers:

1. Install the `python-barbicanclient` Python package.
2. Set up the Key Manager service by editing `/etc/nova/nova.conf`:

   ```
   [key_manager]
   backend = barbican
   ```

   Note

   Use a # prefix to comment out the line in this section that begins with fixed_key.

3. Restart `nova-compute`.

Key management access control

Special privileges can be assigned on behalf of an end user to allow them to manage their own encryption keys, which are required when creating the encrypted volumes. The Barbican Default Policy for access control specifies that only users with an `admin` or `creator` role can create keys. The policy is very flexible and can be modified.

To assign the `creator` role, the admin must know the user ID, project ID, and creator role ID. See Assign a role for more information. An admin can list existing roles and associated IDs using the `openstack role list` command. If the creator role does not exist, the admin can create the role.

Create an encrypted volume type

Block Storage volume type assignment provides scheduling to a specific back-end, and can be used to specify actionable information for a back-end storage device.

This example creates a volume type called LUKS and provides configuration information for the storage system to encrypt or decrypt the volume.

1. Source your admin credentials:

```
$ . admin-openrc.sh
```

2. Create the volume type, marking the volume type as encrypted and providing the necessary details. Use `--encryption-control-location` to specify where encryption is performed: `front-end` (default) or `back-end`.

```
$ openstack volume type create --encryption-provider luks \
  --encryption-cipher aes-xts-plain64 --encryption-key-size 256 --
↪encryption-control-location front-end LUKS

  +-------------+----------------------------------------------------------
↪-------+
  | Field       | Value                                                    ␣
↪       |
  +-------------+----------------------------------------------------------
↪-------+
  | description | None                                                     ␣
↪       |
  | encryption  | cipher='aes-xts-plain64', control_location='front-end',␣
↪       |
  |             | encryption_id='8584c43f-1666-43d1-a348-45cfcef72898',    ␣
↪       |
  |             | key_size='256',                                          ␣
↪       |
  |             | provider='luks'                                          ␣
↪       |
  | id          | b9a8cff5-2f60-40d1-8562-d33f3bf18312                     ␣
↪       |
  | is_public   | True                                                     ␣
↪       |
  | name        | LUKS                                                     ␣
↪       |
  +-------------+----------------------------------------------------------
↪-------+
```

The OpenStack dashboard (horizon) supports creating the encrypted volume type as of the Kilo release. For instructions, see Create an encrypted volume type.

Create an encrypted volume

Use the OpenStack dashboard (horizon), or **openstack volume create** command to create volumes just as you normally would. For an encrypted volume, pass the `--type LUKS` flag, which specifies that the volume type will be LUKS (Linux Unified Key Setup). If that argument is left out, the default volume type, `unencrypted`, is used.

1. Source your admin credentials:

```
$ . admin-openrc.sh
```

2. Create an unencrypted 1GB test volume:

```
$ openstack volume create --size 1 'unencrypted volume'
```

3. Create an encrypted 1GB test volume:

```
$ openstack volume create --size 1 --type LUKS 'encrypted volume'
```

Notice the encrypted parameter; it will show `True` or `False`. The option `volume_type` is also shown for easy review.

Non-admin users need the `creator` role to store secrets in Barbican and to create encrypted volumes. As an administrator, you can give a user the creator role in the following way:

```
$ openstack role add --project PROJECT --user USER creator
```

For details, see the Barbican Access Control page.

Testing volume encryption

This is a simple test scenario to help validate your encryption. It assumes an LVM based Block Storage server.

Perform these steps after completing the volume encryption setup and creating the volume-type for LUKS as described in the preceding sections.

1. Create a VM:

```
$ openstack server create --image cirros-0.3.1-x86_64-disk --flavor m1.
↪tiny TESTVM
```

2. Create two volumes, one encrypted and one not encrypted then attach them to your VM:

```
$ openstack volume create --size 1 'unencrypted volume'
$ openstack volume create --size 1 --type LUKS 'encrypted volume'
$ openstack volume list
$ openstack server add volume --device /dev/vdb TESTVM 'unencrypted volume
↪'
$ openstack server add volume --device /dev/vdc TESTVM 'encrypted volume'
```

Note

The `--device` option to specify the mountpoint for the attached volume may not be where the block device is actually attached in the guest VM, it is used here for illustration purposes.

3. On the VM, send some text to the newly attached volumes and synchronize them:

```
# echo "Hello, world (unencrypted /dev/vdb)" >> /dev/vdb
# echo "Hello, world (encrypted /dev/vdc)" >> /dev/vdc
# sync && sleep 2
# sync && sleep 2
```

4. On the system hosting cinder volume services, synchronize to flush the I/O cache then test to see if your strings can be found:

```
# sync && sleep 2
# sync && sleep 2
# strings /dev/stack-volumes/volume-* | grep "Hello"
Hello, world (unencrypted /dev/vdb)
```

In the above example you see that the search returns the string written to the unencrypted volume, but not the encrypted one.

Known Issues

Retyping an unencrypted volume to the same size encrypted volume will most likely fail. Even though the volume is the same size as the source volume, the encrypted volume needs to store additional encryption information overhead. This results in the new volume not being large enough to hold all data.

Additional options

These options can also be set in the `cinder.conf` file.

Table 116: Description of API configuration options

Configuration option = Default value	Description
api_rate_limit = True	(Boolean) Enables or disables rate limit of the API.
compute_api_class = cinder.compute.nova.API	(String) The full class name of the compute API class to use
group_api_class = cinder.group.api.API	(String) The full class name of the group API class
osapi_volume_ext_list = []	(List of String) Specify list of extensions to load when using osapi_volume_extension option with cinder.api.contrib.select_extensions
osapi_volume_extension = [cinder.api.contrib.standard_extensions]	(String) osapi volume extension to load
tcp_keepalive = True	(Boolean) Sets the value of TCP_KEEPALIVE (True/False) for each server socket.
tcp_keepalive_count = None	(Integer) Sets the value of TCP_KEEPCNT for each server socket. Not supported on OS X.
tcp_keepalive_interval = None	(Integer) Sets the value of TCP_KEEPINTVL in seconds for each server socket. Not supported on OS X.
volume_api_class = cinder.volume.api.API	(String) The full class name of the volume API class to use

Table 117: Description of [oslo_middleware] configuration options

Configuration option = Default value	Description
enable_proxy_headers_ = False	(Boolean) Whether the application is behind a proxy or not. This determines if the middleware should parse the headers or not.
max_request_body_size = 114688	(Integer) The maximum body size for each request, in bytes.

Table 118: Description of authorization configuration options

Configuration option = Default value	Description
auth_strategy = keystone	(String(choices=[noauth, noauth_include_project_id, keystone])) The strategy to use for auth. Supports noauth, noauth_include_project_id or keystone.

Table 119: Description of Volume Manager configuration options

Configuration option = Default value	Description
backend_nativ = 20	(Integer(min=20)) Size of the native threads pool for the backend. Increase for backends that heavily rely on this, like the RBD driver.
backend_stats = 60	(Integer(min=3)) Time in seconds between requests for usage statistics from the backend. Be aware that generating usage statistics is expensive for some backends, so setting this value too low may adversely affect performance.
extra_capabil = {}	(String) User defined capabilities, a JSON formatted string specifying key/value pairs. The key/value pairs can be used by the CapabilitiesFilter to select between backends when requests specify volume types. For example, specifying a service level or the geographical location of a backend, then creating a volume type to allow the user to select by these different properties.
init_host_max = 0	(Integer) Max number of volumes and snapshots to be retrieved per batch during volume manager host initialization. Query results will be obtained in batches from the database and not in one shot to avoid extreme memory usage. Set 0 to turn off this functionality.
migration_cre = 300	(Integer) Timeout for creating the volume to migrate to when performing volume migration (seconds)
reinit_driver = 3	(Integer) Maximum times to reintialize the driver if volume initialization fails. The interval of retry is exponentially backoff, and will be 1s, 2s, 4s etc.
suppress_requ = False	(Boolean) Suppress requests library SSL certificate warnings.
volume_driver = cinder.volume.drivers.lvm.LVMVolumeDriv	(String) Driver to use for volume creation
volume_servic = False	(Boolean) Offload pending volume delete during volume service startup
zoning_mode = None	(String) FC Zoning mode configured, only fabric is supported now.

Table 120: Description of Volume Scheduler configuration options

Configuration option = Default value	Description
allocated_capacity_weight_multiplier = -1.0	(Float) Multiplier used for weighing allocated capacity. Positive numbers mean to stack vs spread.
capacity_weight_multiplier = 1.0	(Float) Multiplier used for weighing free capacity. Negative numbers mean to stack vs spread.
scheduler_default_filters = [AvailabilityZoneFilter, CapacityFilter, CapabilitiesFilter]	(List of String) Which filter class names to use for filtering hosts when not specified in the request.
scheduler_default_weighers = [CapacityWeigher]	(List of String) Which weigher class names to use for weighing hosts.
scheduler_driver = cinder.scheduler.filter_scheduler.FilterScheduler	(String) Default scheduler driver to use
scheduler_driver_init_wait_time = 60	(Integer(min=1)) Maximum time in seconds to wait for the driver to report as ready
scheduler_host_manager = cinder.scheduler.host_manager.HostManager	(String) The scheduler host manager class to use
scheduler_max_attempts = 3	(Integer) Maximum number of attempts to schedule a volume
scheduler_weight_handler = cinder.scheduler.weights.OrderedHostWeightHandler	(String) Which handler to use for selecting the host/pool after weighing
volume_number_multiplier = -1.0	(Float) Multiplier used for weighing volume number. Negative numbers mean to spread vs stack.

Table 121: Description of backup configuration options

Configuration option = Default value	Description
`backup_api_class = cinder.backup.api.API`	(String) The full class name of the volume backup API class
`backup_compression_algor: = zlib`	(String(choices=[none, off, no, zlib, gzip, bz2, bzip2, zstd])) Compression algorithm for backups (none to disable)
`backup_driver = cinder.backup.drivers.swift.SwiftBackupDriver`	(String) Driver to use for backups.
`backup_driver_init_check_ = 60`	(Integer(min=5)) Time in seconds between checks to see if the backup driver has been successfully initialized, any time the driver is restarted.
`backup_driver_stats_poll: = 60`	(Integer(min=10)) Time in seconds between checks of the backup driver status. If does not report as working, it is restarted.
`backup_manager = cinder.backup.manager.BackupManager`	(String) Full class name for the Manager for volume backup
`backup_metadata_version = 2`	(Integer) Backup metadata version to be used when backing up volume metadata. If this number is bumped, make sure the service doing the restore supports the new version.
`backup_name_template = backup-%s`	(String) Template string to be used to generate backup names
`backup_native_threads_poo = 60`	(Integer(min=20)) Size of the native threads pool for the backups. Most backup drivers rely heavily on this, it can be decreased for specific drivers that dont.
`backup_object_number_per_ = 10`	(Integer) The number of chunks or objects, for which one Ceilometer notification will be sent
`backup_service_inithost_( = True`	(Boolean) Offload pending backup delete during backup service startup. If false, the backup service will remain down until all pending backups are deleted.
`backup_timer_interval = 120`	(Integer) Interval, in seconds, between two progress notifications reporting the backup status
`backup_use_same_host = False`	(Boolean) Backup services use same backend.

Table 122: Description of [nova] configuration options

Configuration option = Default value	Description
`auth_section` = `None`	(<class str>) Config Section from which to load plugin specific options
`auth_type` = `None`	(<class str>) Authentication type to load
`cafile` = `None`	(String) PEM encoded Certificate Authority to use when verifying HTTPs connections.
`certfile` = `None`	(String) PEM encoded client certificate cert file
`collect-timing` = `False`	(Boolean) Collect per-API call timing information.
`insecure` = `False`	(Boolean) Verify HTTPS connections.
`interface` = `public`	(String(choices=[public, admin, internal])) Type of the nova endpoint to use. This endpoint will be looked up in the keystone catalog and should be one of public, internal or admin.
`keyfile` = `None`	(String) PEM encoded client certificate key file
`region_name` = `None`	(String) Name of nova region to use. Useful if keystone manages more than one region.
`split-loggers` = `False`	(Boolean) Log requests to multiple loggers.
`timeout` = `None`	(Integer) Timeout value for http requests
`token_auth_url` = `None`	(String) The authentication URL for the nova connection when using the current users token

Table 123: Description of images configuration options

Configuration option = Default value	Description
allowed_direc = []	(List of String) A list of url schemes that can be downloaded directly via the direct_url. Currently supported schemes: [file, cinder].
enforce_multi = False	(Boolean) If this is set to True, attachment of volumes for image transfer will be aborted when multipathd is not running. Otherwise, it will fallback to single path. This parameter needs to be configured for each backend section or in [backend_defaults] section as a common configuration for all backends.
glance_api_ir = False	(Boolean) Allow to perform insecure SSL (https) requests to glance (https will be used but cert validation will not be performed).
glance_api_se = None	(List of String) A list of the URLs of glance API servers available to cinder ([http[s]://][hostname|ip]:port). If protocol is not specified it defaults to http.
glance_api_ss = False	(Boolean) Enables or disables negotiation of SSL layer compression. In some cases disabling compression can improve data throughput, such as when high network bandwidth is available and you use compressed image formats like qcow2.
glance_ca_cer = None	(String) Location of ca certificates file to use for glance client requests.
glance_catalo = image:glance:	(String) Info to match when looking for glance in the service catalog. Format is: separated values of the form: <service_type>:<service_name>:<endpoint_type> - Only used if glance_api_servers are not provided.
glance_certfi = None	(String) Location of certificate file to use for glance client requests.
glance_core_p = [checksum, container_for disk_format, image_name, image_id, min_disk, min_ram, name, size]	(List of String) Default core properties of image
glance_keyfil = None	(String) Location of certificate key file to use for glance client requests.
glance_num_re = 3	(Integer(min=0)) Number retries when downloading an image from glance
glance_reques = None	(Integer) http/https timeout value for glance operations. If no value (None) is supplied here, the glanceclient default value is used.
image_compres = True	(Boolean) When possible, compress images uploaded to the image service
image_convers = 1	(Integer) Address space limit in gigabytes to convert the image
image_convers = 60	(Integer) CPU time limit in seconds to convert the image
image_convers = $state_path/ conversion	(String) Directory used for temporary storage during image conversion
image_convers = False	(Boolean) Disallow image conversion when creating a volume from an image and when uploading a volume as an image. Image conversion consumes a large amount of system resources and can cause performance problems on the cinder-volume node. When set True, this option disables image conversion.
image_upload_	(Boolean) If set to True, upload-to-image in raw format will create a cloned vol-

Table 124: Description of NAS configuration options

Configuration option = Default value	Description
nas_host = <>	(String) IP address or Hostname of NAS system.
nas_logir = admin	(String) User name to connect to NAS system.
nas_mount = None	(String) Options used to mount the storage backend file system where Cinder volumes are stored.
nas_passw = <>	(String) Password to connect to NAS system.
nas_priva = <>	(String) Filename of private key to use for SSH authentication.
nas_secur = auto	(String) Allow network-attached storage systems to operate in a secure environment where root level access is not permitted. If set to False, access is as the root user and insecure. If set to True, access is not as root. If set to auto, a check is done to determine if this is a new installation: True is used if so, otherwise False. Default is auto.
nas_secur = auto	(String) Set more secure file permissions on network-attached storage volume files to restrict broad other/world access. If set to False, volumes are created with open permissions. If set to True, volumes are created with permissions for the cinder user and group (660). If set to auto, a check is done to determine if this is a new installation: True is used if so, otherwise False. Default is auto.
nas_share = <>	(String) Path to the share to use for storing Cinder volumes. For example: /srv/export1 for an NFS server export available at 10.0.5.10:/srv/export1 .
nas_ssh_p = 22	(Port(min=0, max=65535)) SSH port to use to connect to NAS system.
nas_volum = thin	(String(choices=[thin, thick])) Provisioning type that will be used when creating volumes.

Configuration option = Default value	Description
backend_availability_zone = None	(String) Availability zone for this volume backend.
chap_password = <>	(String) Password for specified CHAP account nam
chap_username = <>	(String) CHAP user name.
chiscsi_conf = /etc/chelsio-iscsi/chiscsi.conf	(String) Chiscsi (CXT) global defaults configuratio
driver_client_cert = None	(String) The path to the client certificate for verifica
driver_client_cert_key = None	(String) The path to the client certificate key for ver
driver_data_namespace = None	(String) Namespace for driver private data values to
driver_ssl_cert_path = None	(String) Can be used to specify a non default path to
driver_ssl_cert_verify = False	(Boolean) If set to True the http client will validate
driver_use_ssl = False	(Boolean) Tell driver to use SSL for connection to b
enable_unsupported_driver = False	(Boolean) Set this to True when you want to allow a
filter_function = None	(String) String representation for an equation that w
goodness_function = None	(String) String representation for an equation that w

Configuration option = Default value	Description
iscsi_target_flags = <>	(String) Sets the target-specific flags for the iSCSI t
iscsi_write_cache = on	(String(choices=[on, off])) Sets the behavior of the
max_over_subscription_ratio = 20.0	(String(regex=^(auto|\d*\.\d+|\d+)$)) Representatio
num_shell_tries = 3	(Integer) Number of times to attempt to run flakey s
num_volume_device_scan_tries = 3	(Integer) The maximum number of times to rescan t
replication_device = None	(Dict of String) Multi opt of dictionaries to represe
report_discard_supported = False	(Boolean) Report to clients of Cinder that the backe
reserved_percentage = 0	(Integer(min=0, max=100)) The percentage of back
storage_protocol = iSCSI	(String(choices=[iSCSI, FC])) Protocol for transfer
target_helper = tgtadm	(String(choices=[tgtadm, lioadm, scstadmin, iscsict
target_ip_address = $my_ip	(String) The IP address that the iSCSI/NVMEoF da
target_port = 3260	(Port(min=0, max=65535)) The port that the iSCSI/
target_prefix = iqn.2010-10.org.openstack:	(String) Prefix for iSCSI/NVMEoF volumes
target_protocol = iscsi	(String(choices=[iscsi, iser, nvmet_rdma, nvmet_tc
target_secondary_ip_addresses = []	(List of String) The list of secondary IP addresses o
trace_flags = None	(List of String) List of options that control which tr
use_chap_auth = False	(Boolean) Option to enable/disable CHAP authentic
volume_backend_name = None	(String) The backend name for a given driver imple
volume_clear = zero	(String(choices=[none, zero])) Method used to wipe
volume_clear_ionice = None	(String) The flag to pass to ionice to alter the i/o pri
volume_clear_size = 0	(Integer(max=1024)) Size in MiB to wipe at start o
volume_copy_blkio_cgroup_name = cinder-volume-copy	(String) The blkio cgroup name to be used to limit b
volume_copy_bps_limit = 0	(Integer) The upper limit of bandwidth of volume c
volume_dd_blocksize = 1M	(String) The default block size used when copying/
volumes_dir = $state_path/volumes	(String) Volume configuration file storage directory
iscsi_iotype = fileio	(String(choices=[blockio, fileio, auto])) Sets the beh

Table 126: Description of common configuration options

Configuration option = Default value	Description
allow_availability_zone_f = False	(Boolean) If the requested Cinder availability zone is unavailable, fall back to the value of default_availability_zone, then storage_availability_zone, instead of failing.
consistencygroup_api_clas = cinder.consistencygroup.api.API	(String) The full class name of the consistencygroup API class
default_availability_zone = None	(String) Default availability zone for new volumes. If not set, the storage_availability_zone option value is used as the default for new volumes.
default_group_type = None	(String) Default group type to use
default_volume_type = __DEFAULT__	(String) Default volume type to use
enable_new_services = True	(Boolean) Services to be added to the available pool on create
enabled_backends = None	(List of String) A list of backend names to use. These backend names should be backed by a unique [CONFIG] group with its options
host = localhost	(String) Name of this node. This can be an opaque identifier. It is not necessarily a host name, FQDN, or IP address.
monkey_patch = False	(Boolean) Enable monkey patching
monkey_patch_modules = []	(List of String) List of modules/decorators to monkey patch
my_ip = <HOST_IP_ADDRESS>	(HostAddress) IP address of this host
no_snapshot_gb_quota = False	(Boolean) Whether snapshots sizes count against global and per volume type gigabyte quotas. By default snapshots sizes are counted.
rootwrap_config = /etc/cinder/rootwrap.conf	(String) Path to the rootwrap configuration file to use for running commands as root
scheduler_manager = cinder.scheduler.manager.SchedulerManager	(String) Full class name for the Manager for scheduler
service_down_time = 60	(Integer) Maximum time since last check-in for a service to be considered up
snapshot_name_template = snapshot-%s	(String) Template string to be used to generate snapshot names
split_loggers = False	(Boolean) Log requests to multiple loggers.
storage_availability_zone = nova	(String) Availability zone of this node. Can be overridden per volume backend with the option backend_availability_zone.
transfer_api_class = cinder.transfer.api.API	(String) The full class name of the volume transfer API class
volume_manager = cinder.volume.manager.VolumeManager	(String) Full class name for the Manager for volume
volume_name_template = volume-%s	(String) Template string to be used to generate volume names
volume_usage_audit_period = month	(String) Time period for which to generate volume usages. The options are hour, day, month, or year.

Table 127: Description of [profiler] configuration options

Configuration option = Default value	Description
`conne` = `messa /`	(String) Connection string for a notifier backend. Default value is `messaging://` which sets the notifier to oslo_messaging. Examples of possible values: * `messaging://` - use oslo_messaging driver for sending spans. * `redis://127.0.0.1:6379` - use redis driver for sending spans. * `mongodb://127.0.0.1:27017` - use mongodb driver for sending spans. * `elasticsearch://127.0.0.1:9200` - use elasticsearch driver for sending spans. * `jaeger://127.0.0.1:6831` - use jaeger tracing as driver for sending spans.
`enabl` = `False`	(Boolean) Enable the profiling for all services on this node. Default value is False (fully disable the profiling feature). Possible values: * True: Enables the feature * False: Disables the feature. The profiling cannot be started via this project operations. If the profiling is triggered by another project, this project part will be empty.
`es_do` = `notif`	(String) Document type for notification indexing in elasticsearch.
`es_sc` = `10000`	(Integer) Elasticsearch splits large requests in batches. This parameter defines maximum size of each batch (for example: es_scroll_size=10000).
`es_sc` = `2m`	(String) This parameter is a time value parameter (for example: es_scroll_time=2m), indicating for how long the nodes that participate in the search will maintain relevant resources in order to continue and support it.
`filte` = `False`	(Boolean) Enable filter traces that contain error/exception to a separated place. Default value is set to False. Possible values: * True: Enable filter traces that contain error/exception. * False: Disable the filter.
`hmac_` = `SECRE`	(String) Secret key(s) to use for encrypting context data for performance profiling. This string value should have the following format: <key1>[,<key2>,<keyn>], where each key is some random string. A user who triggers the profiling via the REST API has to set one of these keys in the headers of the REST API call to include profiling results of this node for this particular project. Both enabled flag and hmac_keys config options should be set to enable profiling. Also, to generate correct profiling information across all services at least one key needs to be consistent between OpenStack projects. This ensures it can be used from client side to generate the trace, containing information from all possible resources.
`proce` = `{}`	(Dict of String) Set process tracer tags.
`senti` = `mymas`	(String) Redissentinel uses a service name to identify a master redis service. This parameter defines the name (for example: `sentinal_service_name=mymaster`).
`servi` = `None`	(String) Set service name prefix to Jaeger service name.

Table 128: Description of quota configuration options

Configuration option = Default value	Description
max_age = 0	(Integer) Number of seconds between subsequent usage refreshes
per_volume_size_limit = -1	(Integer) Max size allowed per volume, in gigabytes
quota_backup_gigabytes = 1000	(Integer) Total amount of storage, in gigabytes, allowed for backups per project
quota_backups = 10	(Integer) Number of volume backups allowed per project
quota_driver = cinder.quota.DbQuotaDriver	(String) Default driver to use for quota checks
quota_gigabytes = 1000	(Integer) Total amount of storage, in gigabytes, allowed for volumes and snapshots per project
quota_groups = 10	(Integer) Number of groups allowed per project
quota_snapshots = 10	(Integer) Number of volume snapshots allowed per project
quota_volumes = 10	(Integer) Number of volumes allowed per project
reservation_clean_interval = $reservation_expire	(Integer) Interval between periodic task runs to clean expired reservations in seconds.
reservation_expire = 86400	(Integer) Number of seconds until a reservation expires
until_refresh = 0	(Integer) Count of reservations until usage is refreshed
use_default_quota_class = True	(Boolean) Enables or disables use of default quota class with default quota.

Table 129: Description of SAN configuration options

Configuration option = Default value	Description
san_api_port = None	(Port(min=0, max=65535)) Port to use to access the SAN API
san_clustername = <>	(String) Cluster name to use for creating volumes
san_ip = <>	(String) IP address of SAN controller
san_is_local = False	(Boolean) Execute commands locally instead of over SSH; use if the volume service is running on the SAN device
san_login = admin	(String) Username for SAN controller
san_password = <>	(String) Password for SAN controller
san_private_key = <>	(String) Filename of private key to use for SSH authentication
san_ssh_port = 22	(Port(min=0, max=65535)) SSH port to use with SAN
san_thin_provision = True	(Boolean) Use thin provisioning for SAN volumes?
ssh_conn_timeout = 30	(Integer) SSH connection timeout in seconds
ssh_max_pool_conn = 5	(Integer) Maximum ssh connections in the pool
ssh_min_pool_conn = 1	(Integer) Minimum ssh connections in the pool

Table 130: Description of iSER volume driver configuration options

Configuration option = Default value	Description
`iser_helper = tgtadm`	(String) The name of the iSER target user-land tool to use
`iser_ip_address = $my_ip`	(String) The IP address that the iSER daemon is listening on
`iser_port = 3260`	(Port(min=0, max=65535)) The port that the iSER daemon is listening on
`iser_target_prefix = iqn.2010-10.org.openstack:`	(String) Prefix for iSER volumes
`num_iser_scan_tries = 3`	(Integer) The maximum number of times to rescan iSER target to find volume

Table 131: Description of NVMET volume driver configuration options

Configuration option = Default value	Description
`nvmet_ns_id = 10`	(Integer) Namespace id for the subsystem for the LVM volume when not sharing targets. The minimum id value when sharing.Maximum supported value in Linux is 8192
`nvmet_port_id = 1`	(Port(min=0, max=65535)) The id of the NVMe target port definition when not sharing targets. The starting port id value when sharing, incremented for each secondary ip address.

Table 132: Description of SCST volume driver configuration options

Configuration option = Default value	Description
`scst_target_driver = iscsi`	(String) SCST target implementation can choose from multiple SCST target drivers.
`scst_target_iqn_name = None`	(String) Certain ISCSI targets have predefined target names, SCST target driver uses this name.

Table 133: Description of zones configuration options

Configuration option = Default value	Description
`cloned_volume_same_az = True`	(Boolean) Ensure that the new volumes are the same AZ as snapshot or source volume

Block Storage service sample configuration files

All the files in this section can be found in /etc/cinder.

cinder.conf

The cinder.conf file is installed in /etc/cinder by default. When you manually install the Block Storage service, the options in the cinder.conf file are set to default values.

See the on-line version of this documentation for the full example config file.

api-paste.ini

Use the api-paste.ini file to configure the Block Storage API service.

```
#############
# OpenStack #
#############

[composite:osapi_volume]
use = call:cinder.api:root_app_factory
/: apiversions
/v3: openstack_volume_api_v3

[composite:openstack_volume_api_v3]
use = call:cinder.api.middleware.auth:pipeline_factory
noauth = cors http_proxy_to_wsgi request_id faultwrap sizelimit osprofiler␣
↪noauth apiv3
noauth_include_project_id = cors http_proxy_to_wsgi request_id faultwrap␣
↪sizelimit osprofiler noauth_include_project_id apiv3
keystone = cors http_proxy_to_wsgi request_id faultwrap sizelimit osprofiler␣
↪authtoken keystonecontext apiv3
keystone_nolimit = cors http_proxy_to_wsgi request_id faultwrap sizelimit␣
↪osprofiler authtoken keystonecontext apiv3

[filter:request_id]
paste.filter_factory = oslo_middleware.request_id:RequestId.factory

[filter:http_proxy_to_wsgi]
paste.filter_factory = oslo_middleware.http_proxy_to_wsgi:HTTPProxyToWSGI.
↪factory

[filter:cors]
paste.filter_factory = oslo_middleware.cors:filter_factory
oslo_config_project = cinder

[filter:faultwrap]
paste.filter_factory = cinder.api.middleware.fault:FaultWrapper.factory

[filter:osprofiler]
paste.filter_factory = osprofiler.web:WsgiMiddleware.factory
```

(continues on next page)

(continued from previous page)

```
[filter:noauth]
paste.filter_factory = cinder.api.middleware.auth:NoAuthMiddleware.factory

[filter:noauth_include_project_id]
paste.filter_factory = cinder.api.middleware.
↪auth:NoAuthMiddlewareIncludeProjectID.factory

[filter:sizelimit]
paste.filter_factory = oslo_middleware.sizelimit:RequestBodySizeLimiter.
↪factory

[app:apiv3]
paste.app_factory = cinder.api.v3.router:APIRouter.factory

[pipeline:apiversions]
pipeline = cors http_proxy_to_wsgi faultwrap osvolumeversionapp

[app:osvolumeversionapp]
paste.app_factory = cinder.api.versions:Versions.factory

##########
# Shared #
##########

[filter:keystonecontext]
paste.filter_factory = cinder.api.middleware.auth:CinderKeystoneContext.
↪factory

[filter:authtoken]
paste.filter_factory = keystonemiddleware.auth_token:filter_factory
```

policy.yaml

The `policy.yaml` file defines additional access controls that apply to the Block Storage service.

Prior to Cinder 12.0.0 (the Queens release), a JSON policy file was required to run Cinder. From the Queens release onward, the following hold:

- It is possible to run Cinder safely without a policy file, as sensible default values are defined in the code.
- If you wish to run Cinder with policies different from the default, you may write a policy file.
  - Given that JSON does not allow comments, we recommend using YAML to write a custom policy file. (Also, see next item.)
  - OpenStack has deprecated the use of a JSON policy file since the Wallaby release (Cinder 18.0.0). If you are still using the JSON format, there is a oslopolicy-convert-json-to-yaml tool that will migrate your existing JSON-formatted policy file to YAML in a backward-compatible way.
- If you supply a custom policy file, you only need to supply entries for the policies you wish to

change from their default values. For instance, if you want to change the default value of volume:create, you only need to keep this single rule in your policy config file.

- The default policy file location is `/etc/cinder/policy.yaml`. You may override this by specifying a different file location as the value of the `policy_file` configuration option in the `[oslo_policy]` section of the the Cinder configuration file.
- Instructions for generating a sample `policy.yaml` file directly from the Cinder source code can be found in the file `README-policy.generate.md` in the `etc/cinder` directory in the Cinder source code repository (or its github mirror).

A sample policy file is available in the online version of this documentation. Make sure you are looking at the sample file for the OpenStack release you are running as the available policy rules and their default values may change from release to release.

rootwrap.conf

The `rootwrap.conf` file defines configuration values used by the `rootwrap` script when the Block Storage service must escalate its privileges to those of the root user.

```
# Configuration for cinder-rootwrap
# This file should be owned by (and only-writeable by) the root user

[DEFAULT]
# List of directories to load filter definitions from (separated by ',').
# These directories MUST all be only writeable by root !
filters_path=/etc/cinder/rootwrap.d,/usr/share/cinder/rootwrap

# List of directories to search executables in, in case filters do not
# explicitely specify a full path (separated by ',')
# If not specified, defaults to system PATH environment variable.
# These directories MUST all be only writeable by root !
exec_dirs=/sbin,/usr/sbin,/bin,/usr/bin,/usr/local/bin,/usr/local/sbin,/usr/
↪lpp/mmfs/bin

# Enable logging to syslog
# Default value is False
use_syslog=False

# Which syslog facility to use.
# Valid values include auth, authpriv, syslog, local0, local1...
# Default value is 'syslog'
syslog_log_facility=syslog

# Which messages to log.
# INFO means log all usage
# ERROR means only log unsuccessful attempts
syslog_log_level=ERROR
```

Warning

For security reasons **Service Tokens must be configured** in OpenStack for Cinder to operate securely. Pay close attention to the *specific section describing it:*. See https://bugs.launchpad.net/nova/+bug/2004555 for details.

Note

The examples of common configurations for shared service and libraries, such as database connections and RPC messaging, can be seen in Cinders sample configuration file: cinder.conf.sample.

The Block Storage service works with many different storage drivers that you can configure by using these instructions.

3.3.2 All About Cinder Drivers

Cinder Driver Support Matrix

The following support matrix reflects the drivers that are currently available or are available in Cinders driver tree at the time of release.

Note

This matrix replaces the old wiki based version of the Cinder Support Matrix as there was no way to ensure the wiki version was properly maintained. The old matrix will be left for reference but this matrix should be treated as the correct state of Cinder.

Required Driver Functions

There are a number of functions that are required to be accepted as a Cinder driver. Rather than list all the required functionality in the matrix we include the list of required functions here for reference.

- Create Volume
- Delete Volume
- Attach Volume
- Detach Volume
- Extend Volume
- Create Snapshot
- Delete Snapshot
- Create Volume from Snapshot
- Create Volume from Volume (clone)
- Create Image from Volume
- Volume Migration (host assisted)

Note

Since the above functions are required their support is assumed and the matrix only includes support for optional functionality.

Note

This matrix is not dynamically generated. It is maintained by the Cinder team and Vendor driver maintainers. While every effort is made to ensure the accuracy of the data in this matrix, discrepancies with actual functionality are possible. Please refer to your vendors support documentation for additional information.

Summary

F tu	S tu	((iS S a D (i	D a C S a D (I iS	D e S a D (i	D P e F (S S a D (S	D P e M (2 8 S a D (i F	D P e S N D (I	D P e S S a D (i F N T	D P e V M S ri (i F	D S ri S a D (i F	D U S a D (I iS	D V A (2 4 8 9 S a D (I iS	D V (' 2 4 S a D (i F	D V S a D (I iS	D X S a D (I iS	F ji E N D (I iS	F g b S a D (I T	G N R S e e D (I	H 3 S a D (I iS	H M D (i F	H N b S a D (i F	H X S a D (I iS	H ta V S a D (I iS	H 1 S ri D (i F	H D ra V V S ri D (i F	H F V S ri D (i F	H F V S ri D (i F	H F s S a C 1 D ((	H T S ri V D (i F	H T S ri V D (i F	H V S ri D (i F	H V S ri D (i F	II D F il S a D (I	II F S te D (i	II G S a D ((	II S tı A c e a F il D (i F	II S a V tı a iz fa il D (i F	Ir fi S a D (i F	Ir s A S ri D (i F	Ir s A S a D (i	K i- n S a D (i F	K K n D (I	L E D (
S p V d D	o ti	✓	✓	✓	✓	✓	✓	✓	✓	×	✓	✓	✓	×	×	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	×	✓	✓	✓	✓	✓	✓	✓	✓	✓
E te a A tc V u	o ti	✓	×	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	×	×	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	×	×	✓	✓	✓	✓	✓	✓	✓
Q	o ti	×	×	✓	✓	✓	×	✓	×	✓	✓	✓	✓	✓	×	×	×	×	✓	×	×	×	✓	✓	✓	✓	✓	×	×	✓	✓	✓	×	×	×	×	✓	✓	✓	×	×	×	×
V u R ca ti	o ti	✓	×	×	✓	✓	×	✓	×	✓	✓	✓	✓	✓	×	×	×	×	✓	×	✓	×	×	✓	✓	✓	✓	×	×	×	✓	✓	✓	×	×	✓	✓	×	✓	×	✓	×	×
C si te G	o ti	×	×	×	✓	✓	×	✓	×	✓	✓	✓	✓	✓	✓	×	×	×	✓	×	✓	✓	✓	✓	✓	✓	✓	×	×	×	✓	✓	✓	×	×	✓	✓	×	✓	×	×	×	×
T P vi si ir	o ti	✓	×	×	✓	✓	✓	✓	×	✓	✓	✓	✓	✓	✓	✓	×	✓	✓	×	✓	✓	✓	✓	✓	✓	✓	×	×	×	✓	✓	×	×	×	×	✓	✓	×	✓	✓	✓	×
V u M g ti (S a A si	o ti	×	×	×	✓	✓	×	×	×	×	✓	✓	✓	✓	×	×	×	×	×	×	×	×	✓	✓	✓	✓	✓	×	×	×	✓	✓	×	×	×	×	✓	×	×	×	×	×	×
M A S p	o ti	✓	×	×	✓	✓	✓	✓	✓	✓	✓	✓	✓	×	✓	×	×	×	✓	✓	✓	✓	✓	×	×	×	×	×	×	×	×	×	✓	×	×	✓	✓	✓	×	✓	×	×	×
R	o	✓	✓	×	✓	✓	×	✓	×	×	✓	✓	✓	✓	×	×	×	×	✓	×	✓	✓	✓	×	×	×	×	×	×	×	×	×	✓	×	×	×	✓	✓	×	×	×	×	×

Details

- **Supported Vendor Driver Status: optional.**

  Notes: A vendor driver is considered supported if the vendor is running a third party CI that regularly runs and reports accurate results. If a vendor doesnt meet this requirement the driver is marked unsupported and is removed if the problem isnt resolved before the end of the subsequent release.

  Driver Support:

  – **(Ceph) iSCSI Storage Driver (iSCSI):** `complete`
  – **DataCore Storage Driver (FC, iSCSI):** `complete`
  – **Datera Storage Driver (iSCSI):** `complete`
  – **Dell PowerFlex (ScaleIO) Storage Driver (ScaleIO):** `complete`
  – **Dell PowerMax (2000, 8000) Storage Driver (iSCSI, FC):** `complete`
  – **Dell PowerStore NFS Driver (NFS):** `complete`
  – **Dell PowerStore Storage Driver (iSCSI, FC, NVMe-TCP):** `complete`
  – **Dell PowerVault ME Series (iSCSI, FC):** `complete`
  – **Dell SC Series Storage Driver (iSCSI, FC):** `missing`
  – **Dell Unity Storage Driver (FC, iSCSI):** `complete`
  – **Dell VMAX Af (250F, 450F, 850F, 950F) Storage Driver (FC, iSCSI):** `complete`
  – **Dell VMAX3 (100K, 200K, 400K) Storage Driver (iSCSI, FC):** `complete`
  – **Dell VNX Storage Driver (FC, iSCSI):** `missing`
  – **Dell XtremeIO Storage Driver (FC, iSCSI):** `missing`
  – **Fujitsu ETERNUS Driver (FC, iSCSI):** `complete`
  – **Fungible Storage Driver (NVMe-TCP):** `complete`
  – **Generic NFS Reference Driver (NFS):** `complete`
  – **HPE 3PAR Storage Driver (FC, iSCSI):** `complete`
  – **HPE MSA Driver (iSCSI, FC):** `complete`
  – **HPE Nimble Storage Driver (iSCSI, FC):** `complete`
  – **HPE XP Storage Driver (FC, iSCSI):** `complete`
  – **Hitachi VSP Storage Driver (FC, iSCSI):** `complete`
  – **Huawei 18000 Series Driver (iSCSI, FC):** `complete`
  – **Huawei Dorado V3, V6 Series Driver (iSCSI, FC):** `complete`
  – **Huawei F V3 Series Driver (iSCSI, FC):** `complete`
  – **Huawei F V5 Series Driver (iSCSI, FC):** `complete`
  – **Huawei FusionStorage, OceanStor 100D Driver (dsware):** `complete`
  – **Huawei T Series V1 Driver (iSCSI, FC):** `complete`

- **Huawei T Series V2 Driver (iSCSI, FC):** `complete`
- **Huawei V3 Series Driver (iSCSI, FC):** `complete`
- **Huawei V5 Series Driver (iSCSI, FC):** `complete`
- **IBM DS8000 Family Storage Driver (FC):** `complete`
- **IBM FlashSystem Driver (iSCSI):** `missing`
- **IBM GPFS Storage Driver (gpfs):** `complete`
- **IBM Spectrum Accelerate Family Driver (iSCSI, FC):** `complete`
- **IBM Storage Virtualize family Driver (iSCSI, FC):** `complete`
- **Infinidat Storage Driver (iSCSI, FC):** `complete`
- **Inspur AS/HF Series Driver (iSCSI, FC):** `complete`
- **Inspur AS13000 Storage Driver (iSCSI):** `complete`
- **Kaminario Storage Driver (iSCSI, FC):** `complete`
- **Kioxia Kumoscale Driver (NVMeOF):** `complete`
- **LINBIT DRBD/LINSTOR Driver (DRBD):** `complete`
- **Lenovo Storage Driver (FC, iSCSI):** `complete`
- **Lightbits Storage Driver (NVMeTCP):** `complete`
- **Logical Volume Manager (LVM) Reference Driver (iSCSI):** `complete`
- **MacroSAN Storage Driver (iSCSI, FC):** `complete`
- **NEC Storage M Series Driver (iSCSI, FC):** `complete`
- **NEC Storage V Series Driver (iSCSI, FC):** `complete`
- **NetApp Data ONTAP Driver (NFS):** `complete`
- **NetApp Data ONTAP Driver (NVMe/TCP):** `complete`
- **NetApp Data ONTAP Driver (iSCSI,FC):** `complete`
- **NetApp Solidfire Driver (iSCSI):** `complete`
- **Nexenta Driver (iSCSI, NFS):** `complete`
- **Open-E JovianDSS Storage Driver (iSCSI):** `complete`
- **ProphetStor Flexvisor Driver (iSCSI, NFS):** `missing`
- **Pure Storage Driver (iSCSI, FC, NVMe-RoCE, NVMe-TCP):** `complete`
- **QNAP Storage Driver (iSCSI):** `missing`
- **Quobyte Storage Driver (quobyte):** `missing`
- **RBD (Ceph) Storage Driver (RBD):** `complete`
- **SandStone Storage Driver (iSCSI):** `complete`
- **Seagate Driver (iSCSI, FC):** `complete`
- **StorPool Storage Driver (storpool):** `complete`

- **Synology Storage Driver (iSCSI):** `complete`
- **TOYOU NetStor Storage Driver (iSCSI, FC):** `complete`
- **TOYOU NetStor TYDS Storage Driver (iSCSI):** `complete`
- **VMware Storage Driver (vmdk):** `complete`
- **Veritas Access iSCSI Driver (iSCSI):** `missing`
- **Veritas Cluster NFS Driver (NFS):** `missing`
- **Virtuozzo Storage Driver (remotefs):** `missing`
- **Windows SMB Driver:** `missing`
- **Windows iSCSI Driver:** `missing`
- **Yadro Tatlin Unified Driver (iSCSI, FC):** `complete`
- **Zadara Storage Driver (iSCSI, NFS):** `complete`
- **infortrend Storage Driver (iSCSI, FC):** `complete`

• **Extend an Attached Volume Status: optional.**

Notes: Cinder supports the ability to extend a volume that is attached to an instance, but not all drivers are able to do this.

Driver Support:

- **(Ceph) iSCSI Storage Driver (iSCSI):** `complete`
- **DataCore Storage Driver (FC, iSCSI):** `missing`
- **Datera Storage Driver (iSCSI):** `complete`
- **Dell PowerFlex (ScaleIO) Storage Driver (ScaleIO):** `complete`
- **Dell PowerMax (2000, 8000) Storage Driver (iSCSI, FC):** `complete`
- **Dell PowerStore NFS Driver (NFS):** `complete`
- **Dell PowerStore Storage Driver (iSCSI, FC, NVMe-TCP):** `complete`
- **Dell PowerVault ME Series (iSCSI, FC):** `complete`
- **Dell SC Series Storage Driver (iSCSI, FC):** `complete`
- **Dell Unity Storage Driver (FC, iSCSI):** `complete`
- **Dell VMAX Af (250F, 450F, 850F, 950F) Storage Driver (FC, iSCSI):** `complete`
- **Dell VMAX3 (100K, 200K, 400K) Storage Driver (iSCSI, FC):** `complete`
- **Dell VNX Storage Driver (FC, iSCSI):** `complete`
- **Dell XtremeIO Storage Driver (FC, iSCSI):** `complete`
- **Fujitsu ETERNUS Driver (FC, iSCSI):** `complete`
- **Fungible Storage Driver (NVMe-TCP):** `missing`
- **Generic NFS Reference Driver (NFS):** `missing`
- **HPE 3PAR Storage Driver (FC, iSCSI):** `complete`
- **HPE MSA Driver (iSCSI, FC):** `complete`

- **HPE Nimble Storage Driver (iSCSI, FC):** `complete`
- **HPE XP Storage Driver (FC, iSCSI):** `complete`
- **Hitachi VSP Storage Driver (FC, iSCSI):** `complete`
- **Huawei 18000 Series Driver (iSCSI, FC):** `complete`
- **Huawei Dorado V3, V6 Series Driver (iSCSI, FC):** `complete`
- **Huawei F V3 Series Driver (iSCSI, FC):** `complete`
- **Huawei F V5 Series Driver (iSCSI, FC):** `complete`
- **Huawei FusionStorage, OceanStor 100D Driver (dsware):** `complete`
- **Huawei T Series V1 Driver (iSCSI, FC):** `complete`
- **Huawei T Series V2 Driver (iSCSI, FC):** `complete`
- **Huawei V3 Series Driver (iSCSI, FC):** `complete`
- **Huawei V5 Series Driver (iSCSI, FC):** `complete`
- **IBM DS8000 Family Storage Driver (FC):** `complete`
- **IBM FlashSystem Driver (iSCSI):** `complete`
- **IBM GPFS Storage Driver (gpfs):** `missing`
- **IBM Spectrum Accelerate Family Driver (iSCSI, FC):** `missing`
- **IBM Storage Virtualize family Driver (iSCSI, FC):** `complete`
- **Infinidat Storage Driver (iSCSI, FC):** `complete`
- **Inspur AS/HF Series Driver (iSCSI, FC):** `complete`
- **Inspur AS13000 Storage Driver (iSCSI):** `complete`
- **Kaminario Storage Driver (iSCSI, FC):** `complete`
- **Kioxia Kumoscale Driver (NVMeOF):** `complete`
- **LINBIT DRBD/LINSTOR Driver (DRBD):** `complete`
- **Lenovo Storage Driver (FC, iSCSI):** `complete`
- **Lightbits Storage Driver (NVMeTCP):** `complete`
- **Logical Volume Manager (LVM) Reference Driver (iSCSI):** `complete`
- **MacroSAN Storage Driver (iSCSI, FC):** `complete`
- **NEC Storage M Series Driver (iSCSI, FC):** `complete`
- **NEC Storage V Series Driver (iSCSI, FC):** `complete`
- **NetApp Data ONTAP Driver (NFS):** `complete`
- **NetApp Data ONTAP Driver (NVMe/TCP):** `complete`
- **NetApp Data ONTAP Driver (iSCSI,FC):** `complete`
- **NetApp Solidfire Driver (iSCSI):** `complete`
- **Nexenta Driver (iSCSI, NFS):** `complete`

- **Open-E JovianDSS Storage Driver (iSCSI):** `missing`
- **ProphetStor Flexvisor Driver (iSCSI, NFS):** `complete`
- **Pure Storage Driver (iSCSI, FC, NVMe-RoCE, NVMe-TCP):** `complete`
- **QNAP Storage Driver (iSCSI):** `complete`
- **Quobyte Storage Driver (quobyte):** `missing`
- **RBD (Ceph) Storage Driver (RBD):** `complete`
- **SandStone Storage Driver (iSCSI):** `complete`
- **Seagate Driver (iSCSI, FC):** `complete`
- **StorPool Storage Driver (storpool):** `complete`
- **Synology Storage Driver (iSCSI):** `complete`
- **TOYOU NetStor Storage Driver (iSCSI, FC):** `complete`
- **TOYOU NetStor TYDS Storage Driver (iSCSI):** `complete`
- **VMware Storage Driver (vmdk):** `complete`
- **Veritas Access iSCSI Driver (iSCSI):** `complete`
- **Veritas Cluster NFS Driver (NFS):** `complete`
- **Virtuozzo Storage Driver (remotefs):** `complete`
- **Windows SMB Driver:** `complete`
- **Windows iSCSI Driver:** `complete`
- **Yadro Tatlin Unified Driver (iSCSI, FC):** `complete`
- **Zadara Storage Driver (iSCSI, NFS):** `complete`
- **infortrend Storage Driver (iSCSI, FC):** `complete`

• **QoS Status: optional.**

Notes: Vendor drivers that support Quality of Service (QoS) at the backend. This means they are able to utilize QoS Specs associated with volume extra specs to control QoS settings at the storage device on a per volume basis. Drivers that dont support this can utilize frontend QoS via libvirt.

Driver Support:

- **(Ceph) iSCSI Storage Driver (iSCSI):** `missing`
- **DataCore Storage Driver (FC, iSCSI):** `missing`
- **Datera Storage Driver (iSCSI):** `complete`
- **Dell PowerFlex (ScaleIO) Storage Driver (ScaleIO):** `complete`
- **Dell PowerMax (2000, 8000) Storage Driver (iSCSI, FC):** `complete`
- **Dell PowerStore NFS Driver (NFS):** `missing`
- **Dell PowerStore Storage Driver (iSCSI, FC, NVMe-TCP):** `complete`
- **Dell PowerVault ME Series (iSCSI, FC):** `missing`
- **Dell SC Series Storage Driver (iSCSI, FC):** `complete`

- **Dell Unity Storage Driver (FC, iSCSI):** `complete`
- **Dell VMAX Af (250F, 450F, 850F, 950F) Storage Driver (FC, iSCSI):** `complete`
- **Dell VMAX3 (100K, 200K, 400K) Storage Driver (iSCSI, FC):** `complete`
- **Dell VNX Storage Driver (FC, iSCSI):** `complete`
- **Dell XtremeIO Storage Driver (FC, iSCSI):** `missing`
- **Fujitsu ETERNUS Driver (FC, iSCSI):** `missing`
- **Fungible Storage Driver (NVMe-TCP):** `missing`
- **Generic NFS Reference Driver (NFS):** `missing`
- **HPE 3PAR Storage Driver (FC, iSCSI):** `complete`
- **HPE MSA Driver (iSCSI, FC):** `missing`
- **HPE Nimble Storage Driver (iSCSI, FC):** `missing`
- **HPE XP Storage Driver (FC, iSCSI):** `missing`
- **Hitachi VSP Storage Driver (FC, iSCSI):** `complete`
- **Huawei 18000 Series Driver (iSCSI, FC):** `complete`
- **Huawei Dorado V3, V6 Series Driver (iSCSI, FC):** `complete`
- **Huawei F V3 Series Driver (iSCSI, FC):** `complete`
- **Huawei F V5 Series Driver (iSCSI, FC):** `complete`
- **Huawei FusionStorage, OceanStor 100D Driver (dsware):** `missing`
- **Huawei T Series V1 Driver (iSCSI, FC):** `missing`
- **Huawei T Series V2 Driver (iSCSI, FC):** `complete`
- **Huawei V3 Series Driver (iSCSI, FC):** `complete`
- **Huawei V5 Series Driver (iSCSI, FC):** `complete`
- **IBM DS8000 Family Storage Driver (FC):** `missing`
- **IBM FlashSystem Driver (iSCSI):** `missing`
- **IBM GPFS Storage Driver (gpfs):** `missing`
- **IBM Spectrum Accelerate Family Driver (iSCSI, FC):** `missing`
- **IBM Storage Virtualize family Driver (iSCSI, FC):** `complete`
- **Infinidat Storage Driver (iSCSI, FC):** `complete`
- **Inspur AS/HF Series Driver (iSCSI, FC):** `complete`
- **Inspur AS13000 Storage Driver (iSCSI):** `missing`
- **Kaminario Storage Driver (iSCSI, FC):** `missing`
- **Kioxia Kumoscale Driver (NVMeOF):** `missing`
- **LINBIT DRBD/LINSTOR Driver (DRBD):** `missing`
- **Lenovo Storage Driver (FC, iSCSI):** `missing`

  - **Lightbits Storage Driver (NVMeTCP):** `missing`
  - **Logical Volume Manager (LVM) Reference Driver (iSCSI):** `missing`
  - **MacroSAN Storage Driver (iSCSI, FC):** `complete`
  - **NEC Storage M Series Driver (iSCSI, FC):** `complete`
  - **NEC Storage V Series Driver (iSCSI, FC):** `missing`
  - **NetApp Data ONTAP Driver (NFS):** `complete`
  - **NetApp Data ONTAP Driver (NVMe/TCP):** `complete`
  - **NetApp Data ONTAP Driver (iSCSI,FC):** `complete`
  - **NetApp Solidfire Driver (iSCSI):** `complete`
  - **Nexenta Driver (iSCSI, NFS):** `missing`
  - **Open-E JovianDSS Storage Driver (iSCSI):** `missing`
  - **ProphetStor Flexvisor Driver (iSCSI, NFS):** `missing`
  - **Pure Storage Driver (iSCSI, FC, NVMe-RoCE, NVMe-TCP):** `complete`
  - **QNAP Storage Driver (iSCSI):** `missing`
  - **Quobyte Storage Driver (quobyte):** `missing`
  - **RBD (Ceph) Storage Driver (RBD):** `complete`
  - **SandStone Storage Driver (iSCSI):** `complete`
  - **Seagate Driver (iSCSI, FC):** `missing`
  - **StorPool Storage Driver (storpool):** `missing`
  - **Synology Storage Driver (iSCSI):** `missing`
  - **TOYOU NetStor Storage Driver (iSCSI, FC):** `missing`
  - **TOYOU NetStor TYDS Storage Driver (iSCSI):** `missing`
  - **VMware Storage Driver (vmdk):** `missing`
  - **Veritas Access iSCSI Driver (iSCSI):** `missing`
  - **Veritas Cluster NFS Driver (NFS):** `missing`
  - **Virtuozzo Storage Driver (remotefs):** `missing`
  - **Windows SMB Driver:** `missing`
  - **Windows iSCSI Driver:** `missing`
  - **Yadro Tatlin Unified Driver (iSCSI, FC):** `complete`
  - **Zadara Storage Driver (iSCSI, NFS):** `missing`
  - **infortrend Storage Driver (iSCSI, FC):** `missing`

- **Volume Replication Status: optional.**

  Notes: Vendor drivers that support volume replication can report this capability to be utilized by the scheduler allowing users to request replicated volumes via extra specs. Such drivers are also then able to take advantage of Cinders failover and failback commands.

Driver Support:

- **(Ceph) iSCSI Storage Driver (iSCSI):** `complete`
- **DataCore Storage Driver (FC, iSCSI):** `missing`
- **Datera Storage Driver (iSCSI):** `missing`
- **Dell PowerFlex (ScaleIO) Storage Driver (ScaleIO):** `complete`
- **Dell PowerMax (2000, 8000) Storage Driver (iSCSI, FC):** `complete`
- **Dell PowerStore NFS Driver (NFS):** `missing`
- **Dell PowerStore Storage Driver (iSCSI, FC, NVMe-TCP):** `complete`
- **Dell PowerVault ME Series (iSCSI, FC):** `missing`
- **Dell SC Series Storage Driver (iSCSI, FC):** `complete`
- **Dell Unity Storage Driver (FC, iSCSI):** `complete`
- **Dell VMAX Af (250F, 450F, 850F, 950F) Storage Driver (FC, iSCSI):** `complete`
- **Dell VMAX3 (100K, 200K, 400K) Storage Driver (iSCSI, FC):** `complete`
- **Dell VNX Storage Driver (FC, iSCSI):** `complete`
- **Dell XtremeIO Storage Driver (FC, iSCSI):** `missing`
- **Fujitsu ETERNUS Driver (FC, iSCSI):** `missing`
- **Fungible Storage Driver (NVMe-TCP):** `missing`
- **Generic NFS Reference Driver (NFS):** `missing`
- **HPE 3PAR Storage Driver (FC, iSCSI):** `complete`
- **HPE MSA Driver (iSCSI, FC):** `missing`
- **HPE Nimble Storage Driver (iSCSI, FC):** `complete`
- **HPE XP Storage Driver (FC, iSCSI):** `missing`
- **Hitachi VSP Storage Driver (FC, iSCSI):** `missing`
- **Huawei 18000 Series Driver (iSCSI, FC):** `complete`
- **Huawei Dorado V3, V6 Series Driver (iSCSI, FC):** `complete`
- **Huawei F V3 Series Driver (iSCSI, FC):** `complete`
- **Huawei F V5 Series Driver (iSCSI, FC):** `complete`
- **Huawei FusionStorage, OceanStor 100D Driver (dsware):** `missing`
- **Huawei T Series V1 Driver (iSCSI, FC):** `missing`
- **Huawei T Series V2 Driver (iSCSI, FC):** `missing`
- **Huawei V3 Series Driver (iSCSI, FC):** `complete`
- **Huawei V5 Series Driver (iSCSI, FC):** `complete`
- **IBM DS8000 Family Storage Driver (FC):** `complete`
- **IBM FlashSystem Driver (iSCSI):** `missing`

- **IBM GPFS Storage Driver (gpfs):** `missing`
- **IBM Spectrum Accelerate Family Driver (iSCSI, FC):** `complete`
- **IBM Storage Virtualize family Driver (iSCSI, FC):** `complete`
- **Infinidat Storage Driver (iSCSI, FC):** `missing`
- **Inspur AS/HF Series Driver (iSCSI, FC):** `complete`
- **Inspur AS13000 Storage Driver (iSCSI):** `missing`
- **Kaminario Storage Driver (iSCSI, FC):** `complete`
- **Kioxia Kumoscale Driver (NVMeOF):** `missing`
- **LINBIT DRBD/LINSTOR Driver (DRBD):** `missing`
- **Lenovo Storage Driver (FC, iSCSI):** `missing`
- **Lightbits Storage Driver (NVMeTCP):** `missing`
- **Logical Volume Manager (LVM) Reference Driver (iSCSI):** `missing`
- **MacroSAN Storage Driver (iSCSI, FC):** `complete`
- **NEC Storage M Series Driver (iSCSI, FC):** `missing`
- **NEC Storage V Series Driver (iSCSI, FC):** `missing`
- **NetApp Data ONTAP Driver (NFS):** `complete`
- **NetApp Data ONTAP Driver (NVMe/TCP):** `complete`
- **NetApp Data ONTAP Driver (iSCSI,FC):** `complete`
- **NetApp Solidfire Driver (iSCSI):** `complete`
- **Nexenta Driver (iSCSI, NFS):** `missing`
- **Open-E JovianDSS Storage Driver (iSCSI):** `missing`
- **ProphetStor Flexvisor Driver (iSCSI, NFS):** `missing`
- **Pure Storage Driver (iSCSI, FC, NVMe-RoCE, NVMe-TCP):** `complete`
- **QNAP Storage Driver (iSCSI):** `missing`
- **Quobyte Storage Driver (quobyte):** `missing`
- **RBD (Ceph) Storage Driver (RBD):** `complete`
- **SandStone Storage Driver (iSCSI):** `complete`
- **Seagate Driver (iSCSI, FC):** `missing`
- **StorPool Storage Driver (storpool):** `complete`
- **Synology Storage Driver (iSCSI):** `missing`
- **TOYOU NetStor Storage Driver (iSCSI, FC):** `missing`
- **TOYOU NetStor TYDS Storage Driver (iSCSI):** `missing`
- **VMware Storage Driver (vmdk):** `missing`
- **Veritas Access iSCSI Driver (iSCSI):** `missing`

- **Veritas Cluster NFS Driver (NFS):** `missing`
- **Virtuozzo Storage Driver (remotefs):** `missing`
- **Windows SMB Driver:** `missing`
- **Windows iSCSI Driver:** `missing`
- **Yadro Tatlin Unified Driver (iSCSI, FC):** `missing`
- **Zadara Storage Driver (iSCSI, NFS):** `missing`
- **infortrend Storage Driver (iSCSI, FC):** `complete`

• **Consistency Groups Status: optional.**

Notes: Vendor drivers that support consistency groups are able to logically group volumes together for things like snapshotting and deletion. Grouping the volumes ensures that operations are only completed on the group of volumes, not individually, enabling the creation of consistent snapshots across a group.

Driver Support:

- **(Ceph) iSCSI Storage Driver (iSCSI):** `missing`
- **DataCore Storage Driver (FC, iSCSI):** `missing`
- **Datera Storage Driver (iSCSI):** `missing`
- **Dell PowerFlex (ScaleIO) Storage Driver (ScaleIO):** `complete`
- **Dell PowerMax (2000, 8000) Storage Driver (iSCSI, FC):** `complete`
- **Dell PowerStore NFS Driver (NFS):** `missing`
- **Dell PowerStore Storage Driver (iSCSI, FC, NVMe-TCP):** `complete`
- **Dell PowerVault ME Series (iSCSI, FC):** `missing`
- **Dell SC Series Storage Driver (iSCSI, FC):** `complete`
- **Dell Unity Storage Driver (FC, iSCSI):** `complete`
- **Dell VMAX Af (250F, 450F, 850F, 950F) Storage Driver (FC, iSCSI):** `complete`
- **Dell VMAX3 (100K, 200K, 400K) Storage Driver (iSCSI, FC):** `complete`
- **Dell VNX Storage Driver (FC, iSCSI):** `complete`
- **Dell XtremeIO Storage Driver (FC, iSCSI):** `complete`
- **Fujitsu ETERNUS Driver (FC, iSCSI):** `missing`
- **Fungible Storage Driver (NVMe-TCP):** `missing`
- **Generic NFS Reference Driver (NFS):** `missing`
- **HPE 3PAR Storage Driver (FC, iSCSI):** `complete`
- **HPE MSA Driver (iSCSI, FC):** `missing`
- **HPE Nimble Storage Driver (iSCSI, FC):** `complete`
- **HPE XP Storage Driver (FC, iSCSI):** `complete`
- **Hitachi VSP Storage Driver (FC, iSCSI):** `complete`

- **Huawei 18000 Series Driver (iSCSI, FC):** `complete`
- **Huawei Dorado V3, V6 Series Driver (iSCSI, FC):** `complete`
- **Huawei F V3 Series Driver (iSCSI, FC):** `complete`
- **Huawei F V5 Series Driver (iSCSI, FC):** `complete`
- **Huawei FusionStorage, OceanStor 100D Driver (dsware):** `missing`
- **Huawei T Series V1 Driver (iSCSI, FC):** `missing`
- **Huawei T Series V2 Driver (iSCSI, FC):** `missing`
- **Huawei V3 Series Driver (iSCSI, FC):** `complete`
- **Huawei V5 Series Driver (iSCSI, FC):** `complete`
- **IBM DS8000 Family Storage Driver (FC):** `complete`
- **IBM FlashSystem Driver (iSCSI):** `missing`
- **IBM GPFS Storage Driver (gpfs):** `missing`
- **IBM Spectrum Accelerate Family Driver (iSCSI, FC):** `complete`
- **IBM Storage Virtualize family Driver (iSCSI, FC):** `complete`
- **Infinidat Storage Driver (iSCSI, FC):** `missing`
- **Inspur AS/HF Series Driver (iSCSI, FC):** `complete`
- **Inspur AS13000 Storage Driver (iSCSI):** `missing`
- **Kaminario Storage Driver (iSCSI, FC):** `missing`
- **Kioxia Kumoscale Driver (NVMeOF):** `missing`
- **LINBIT DRBD/LINSTOR Driver (DRBD):** `missing`
- **Lenovo Storage Driver (FC, iSCSI):** `missing`
- **Lightbits Storage Driver (NVMeTCP):** `missing`
- **Logical Volume Manager (LVM) Reference Driver (iSCSI):** `missing`
- **MacroSAN Storage Driver (iSCSI, FC):** `missing`
- **NEC Storage M Series Driver (iSCSI, FC):** `missing`
- **NEC Storage V Series Driver (iSCSI, FC):** `complete`
- **NetApp Data ONTAP Driver (NFS):** `complete`
- **NetApp Data ONTAP Driver (NVMe/TCP):** `complete`
- **NetApp Data ONTAP Driver (iSCSI,FC):** `complete`
- **NetApp Solidfire Driver (iSCSI):** `complete`
- **Nexenta Driver (iSCSI, NFS):** `missing`
- **Open-E JovianDSS Storage Driver (iSCSI):** `missing`
- **ProphetStor Flexvisor Driver (iSCSI, NFS):** `complete`
- **Pure Storage Driver (iSCSI, FC, NVMe-RoCE, NVMe-TCP):** `complete`

- **QNAP Storage Driver (iSCSI):** `missing`
- **Quobyte Storage Driver (quobyte):** `missing`
- **RBD (Ceph) Storage Driver (RBD):** `missing`
- **SandStone Storage Driver (iSCSI):** `missing`
- **Seagate Driver (iSCSI, FC):** `missing`
- **StorPool Storage Driver (storpool):** `missing`
- **Synology Storage Driver (iSCSI):** `missing`
- **TOYOU NetStor Storage Driver (iSCSI, FC):** `missing`
- **TOYOU NetStor TYDS Storage Driver (iSCSI):** `missing`
- **VMware Storage Driver (vmdk):** `missing`
- **Veritas Access iSCSI Driver (iSCSI):** `missing`
- **Veritas Cluster NFS Driver (NFS):** `missing`
- **Virtuozzo Storage Driver (remotefs):** `missing`
- **Windows SMB Driver:** `missing`
- **Windows iSCSI Driver:** `missing`
- **Yadro Tatlin Unified Driver (iSCSI, FC):** `missing`
- **Zadara Storage Driver (iSCSI, NFS):** `missing`
- **infortrend Storage Driver (iSCSI, FC):** `missing`

• **Thin Provisioning Status: optional.**

Notes: If a volume driver supports thin provisioning it means that it will allow the scheduler to provision more storage space than physically exists on the backend. This may also be called oversubscription.

Driver Support:

- **(Ceph) iSCSI Storage Driver (iSCSI):** `complete`
- **DataCore Storage Driver (FC, iSCSI):** `missing`
- **Datera Storage Driver (iSCSI):** `missing`
- **Dell PowerFlex (ScaleIO) Storage Driver (ScaleIO):** `complete`
- **Dell PowerMax (2000, 8000) Storage Driver (iSCSI, FC):** `complete`
- **Dell PowerStore NFS Driver (NFS):** `complete`
- **Dell PowerStore Storage Driver (iSCSI, FC, NVMe-TCP):** `complete`
- **Dell PowerVault ME Series (iSCSI, FC):** `missing`
- **Dell SC Series Storage Driver (iSCSI, FC):** `complete`
- **Dell Unity Storage Driver (FC, iSCSI):** `complete`
- **Dell VMAX Af (250F, 450F, 850F, 950F) Storage Driver (FC, iSCSI):** `complete`
- **Dell VMAX3 (100K, 200K, 400K) Storage Driver (iSCSI, FC):** `complete`

- **Dell VNX Storage Driver (FC, iSCSI):** `complete`
- **Dell XtremeIO Storage Driver (FC, iSCSI):** `complete`
- **Fujitsu ETERNUS Driver (FC, iSCSI):** `complete`
- **Fungible Storage Driver (NVMe-TCP):** `missing`
- **Generic NFS Reference Driver (NFS):** `complete`
- **HPE 3PAR Storage Driver (FC, iSCSI):** `complete`
- **HPE MSA Driver (iSCSI, FC):** `missing`
- **HPE Nimble Storage Driver (iSCSI, FC):** `complete`
- **HPE XP Storage Driver (FC, iSCSI):** `complete`
- **Hitachi VSP Storage Driver (FC, iSCSI):** `complete`
- **Huawei 18000 Series Driver (iSCSI, FC):** `complete`
- **Huawei Dorado V3, V6 Series Driver (iSCSI, FC):** `complete`
- **Huawei F V3 Series Driver (iSCSI, FC):** `complete`
- **Huawei F V5 Series Driver (iSCSI, FC):** `complete`
- **Huawei FusionStorage, OceanStor 100D Driver (dsware):** `missing`
- **Huawei T Series V1 Driver (iSCSI, FC):** `missing`
- **Huawei T Series V2 Driver (iSCSI, FC):** `missing`
- **Huawei V3 Series Driver (iSCSI, FC):** `complete`
- **Huawei V5 Series Driver (iSCSI, FC):** `complete`
- **IBM DS8000 Family Storage Driver (FC):** `missing`
- **IBM FlashSystem Driver (iSCSI):** `missing`
- **IBM GPFS Storage Driver (gpfs):** `missing`
- **IBM Spectrum Accelerate Family Driver (iSCSI, FC):** `missing`
- **IBM Storage Virtualize family Driver (iSCSI, FC):** `complete`
- **Infinidat Storage Driver (iSCSI, FC):** `complete`
- **Inspur AS/HF Series Driver (iSCSI, FC):** `missing`
- **Inspur AS13000 Storage Driver (iSCSI):** `complete`
- **Kaminario Storage Driver (iSCSI, FC):** `complete`
- **Kioxia Kumoscale Driver (NVMeOF):** `complete`
- **LINBIT DRBD/LINSTOR Driver (DRBD):** `missing`
- **Lenovo Storage Driver (FC, iSCSI):** `missing`
- **Lightbits Storage Driver (NVMeTCP):** `complete`
- **Logical Volume Manager (LVM) Reference Driver (iSCSI):** `complete`
- **MacroSAN Storage Driver (iSCSI, FC):** `complete`

  - **NEC Storage M Series Driver (iSCSI, FC):** `complete`
  - **NEC Storage V Series Driver (iSCSI, FC):** `complete`
  - **NetApp Data ONTAP Driver (NFS):** `complete`
  - **NetApp Data ONTAP Driver (NVMe/TCP):** `complete`
  - **NetApp Data ONTAP Driver (iSCSI,FC):** `complete`
  - **NetApp Solidfire Driver (iSCSI):** `complete`
  - **Nexenta Driver (iSCSI, NFS):** `missing`
  - **Open-E JovianDSS Storage Driver (iSCSI):** `complete`
  - **ProphetStor Flexvisor Driver (iSCSI, NFS):** `missing`
  - **Pure Storage Driver (iSCSI, FC, NVMe-RoCE, NVMe-TCP):** `complete`
  - **QNAP Storage Driver (iSCSI):** `missing`
  - **Quobyte Storage Driver (quobyte):** `missing`
  - **RBD (Ceph) Storage Driver (RBD):** `complete`
  - **SandStone Storage Driver (iSCSI):** `complete`
  - **Seagate Driver (iSCSI, FC):** `missing`
  - **StorPool Storage Driver (storpool):** `complete`
  - **Synology Storage Driver (iSCSI):** `missing`
  - **TOYOU NetStor Storage Driver (iSCSI, FC):** `complete`
  - **TOYOU NetStor TYDS Storage Driver (iSCSI):** `complete`
  - **VMware Storage Driver (vmdk):** `missing`
  - **Veritas Access iSCSI Driver (iSCSI):** `missing`
  - **Veritas Cluster NFS Driver (NFS):** `missing`
  - **Virtuozzo Storage Driver (remotefs):** `missing`
  - **Windows SMB Driver:** `complete`
  - **Windows iSCSI Driver:** `missing`
  - **Yadro Tatlin Unified Driver (iSCSI, FC):** `complete`
  - **Zadara Storage Driver (iSCSI, NFS):** `missing`
  - **infortrend Storage Driver (iSCSI, FC):** `complete`

- **Volume Migration (Storage Assisted) Status: optional.**

  Notes: Storage assisted volume migration is like host assisted volume migration except that a volume can be migrated without the assistance of the Cinder host. Vendor drivers that implement this can migrate volumes completely through the storage backends functionality.

  Driver Support:

  - **(Ceph) iSCSI Storage Driver (iSCSI):** `missing`
  - **DataCore Storage Driver (FC, iSCSI):** `missing`

- **Datera Storage Driver (iSCSI):** `missing`
- **Dell PowerFlex (ScaleIO) Storage Driver (ScaleIO):** `complete`
- **Dell PowerMax (2000, 8000) Storage Driver (iSCSI, FC):** `complete`
- **Dell PowerStore NFS Driver (NFS):** `missing`
- **Dell PowerStore Storage Driver (iSCSI, FC, NVMe-TCP):** `missing`
- **Dell PowerVault ME Series (iSCSI, FC):** `missing`
- **Dell SC Series Storage Driver (iSCSI, FC):** `missing`
- **Dell Unity Storage Driver (FC, iSCSI):** `complete`
- **Dell VMAX Af (250F, 450F, 850F, 950F) Storage Driver (FC, iSCSI):** `complete`
- **Dell VMAX3 (100K, 200K, 400K) Storage Driver (iSCSI, FC):** `complete`
- **Dell VNX Storage Driver (FC, iSCSI):** `complete`
- **Dell XtremeIO Storage Driver (FC, iSCSI):** `missing`
- **Fujitsu ETERNUS Driver (FC, iSCSI):** `missing`
- **Fungible Storage Driver (NVMe-TCP):** `missing`
- **Generic NFS Reference Driver (NFS):** `missing`
- **HPE 3PAR Storage Driver (FC, iSCSI):** `missing`
- **HPE MSA Driver (iSCSI, FC):** `missing`
- **HPE Nimble Storage Driver (iSCSI, FC):** `missing`
- **HPE XP Storage Driver (FC, iSCSI):** `missing`
- **Hitachi VSP Storage Driver (FC, iSCSI):** `complete`
- **Huawei 18000 Series Driver (iSCSI, FC):** `complete`
- **Huawei Dorado V3, V6 Series Driver (iSCSI, FC):** `complete`
- **Huawei F V3 Series Driver (iSCSI, FC):** `complete`
- **Huawei F V5 Series Driver (iSCSI, FC):** `complete`
- **Huawei FusionStorage, OceanStor 100D Driver (dsware):** `missing`
- **Huawei T Series V1 Driver (iSCSI, FC):** `missing`
- **Huawei T Series V2 Driver (iSCSI, FC):** `missing`
- **Huawei V3 Series Driver (iSCSI, FC):** `complete`
- **Huawei V5 Series Driver (iSCSI, FC):** `complete`
- **IBM DS8000 Family Storage Driver (FC):** `missing`
- **IBM FlashSystem Driver (iSCSI):** `missing`
- **IBM GPFS Storage Driver (gpfs):** `missing`
- **IBM Spectrum Accelerate Family Driver (iSCSI, FC):** `missing`
- **IBM Storage Virtualize family Driver (iSCSI, FC):** `complete`

- **Infinidat Storage Driver (iSCSI, FC):** `missing`
- **Inspur AS/HF Series Driver (iSCSI, FC):** `missing`
- **Inspur AS13000 Storage Driver (iSCSI):** `missing`
- **Kaminario Storage Driver (iSCSI, FC):** `missing`
- **Kioxia Kumoscale Driver (NVMeOF):** `missing`
- **LINBIT DRBD/LINSTOR Driver (DRBD):** `missing`
- **Lenovo Storage Driver (FC, iSCSI):** `missing`
- **Lightbits Storage Driver (NVMeTCP):** `missing`
- **Logical Volume Manager (LVM) Reference Driver (iSCSI):** `missing`
- **MacroSAN Storage Driver (iSCSI, FC):** `complete`
- **NEC Storage M Series Driver (iSCSI, FC):** `complete`
- **NEC Storage V Series Driver (iSCSI, FC):** `missing`
- **NetApp Data ONTAP Driver (NFS):** `complete`
- **NetApp Data ONTAP Driver (NVMe/TCP):** `complete`
- **NetApp Data ONTAP Driver (iSCSI,FC):** `complete`
- **NetApp Solidfire Driver (iSCSI):** `complete`
- **Nexenta Driver (iSCSI, NFS):** `missing`
- **Open-E JovianDSS Storage Driver (iSCSI):** `missing`
- **ProphetStor Flexvisor Driver (iSCSI, NFS):** `missing`
- **Pure Storage Driver (iSCSI, FC, NVMe-RoCE, NVMe-TCP):** `missing`
- **QNAP Storage Driver (iSCSI):** `missing`
- **Quobyte Storage Driver (quobyte):** `missing`
- **RBD (Ceph) Storage Driver (RBD):** `missing`
- **SandStone Storage Driver (iSCSI):** `missing`
- **Seagate Driver (iSCSI, FC):** `missing`
- **StorPool Storage Driver (storpool):** `complete`
- **Synology Storage Driver (iSCSI):** `missing`
- **TOYOU NetStor Storage Driver (iSCSI, FC):** `complete`
- **TOYOU NetStor TYDS Storage Driver (iSCSI):** `missing`
- **VMware Storage Driver (vmdk):** `missing`
- **Veritas Access iSCSI Driver (iSCSI):** `missing`
- **Veritas Cluster NFS Driver (NFS):** `missing`
- **Virtuozzo Storage Driver (remotefs):** `missing`
- **Windows SMB Driver:** `missing`

  - **Windows iSCSI Driver:** `missing`
  - **Yadro Tatlin Unified Driver (iSCSI, FC):** `missing`
  - **Zadara Storage Driver (iSCSI, NFS):** `missing`
  - **infortrend Storage Driver (iSCSI, FC):** `complete`

- **Multi-Attach Support Status: optional.**

  Notes: Vendor drivers that report multi-attach support are able to make one volume available to multiple instances at once. It is important to note that a clustered file system that supports multi-attach functionality is required to use multi- attach functionality otherwise data corruption may occur.

  Driver Support:

  - **(Ceph) iSCSI Storage Driver (iSCSI):** `complete`
  - **DataCore Storage Driver (FC, iSCSI):** `missing`
  - **Datera Storage Driver (iSCSI):** `missing`
  - **Dell PowerFlex (ScaleIO) Storage Driver (ScaleIO):** `complete`
  - **Dell PowerMax (2000, 8000) Storage Driver (iSCSI, FC):** `complete`
  - **Dell PowerStore NFS Driver (NFS):** `complete`
  - **Dell PowerStore Storage Driver (iSCSI, FC, NVMe-TCP):** `complete`
  - **Dell PowerVault ME Series (iSCSI, FC):** `complete`
  - **Dell SC Series Storage Driver (iSCSI, FC):** `complete`
  - **Dell Unity Storage Driver (FC, iSCSI):** `complete`
  - **Dell VMAX Af (250F, 450F, 850F, 950F) Storage Driver (FC, iSCSI):** `complete`
  - **Dell VMAX3 (100K, 200K, 400K) Storage Driver (iSCSI, FC):** `complete`
  - **Dell VNX Storage Driver (FC, iSCSI):** `missing`
  - **Dell XtremeIO Storage Driver (FC, iSCSI):** `complete`
  - **Fujitsu ETERNUS Driver (FC, iSCSI):** `missing`
  - **Fungible Storage Driver (NVMe-TCP):** `missing`
  - **Generic NFS Reference Driver (NFS):** `missing`
  - **HPE 3PAR Storage Driver (FC, iSCSI):** `complete`
  - **HPE MSA Driver (iSCSI, FC):** `complete`
  - **HPE Nimble Storage Driver (iSCSI, FC):** `complete`
  - **HPE XP Storage Driver (FC, iSCSI):** `complete`
  - **Hitachi VSP Storage Driver (FC, iSCSI):** `complete`
  - **Huawei 18000 Series Driver (iSCSI, FC):** `missing`
  - **Huawei Dorado V3, V6 Series Driver (iSCSI, FC):** `missing`
  - **Huawei F V3 Series Driver (iSCSI, FC):** `missing`

- **Huawei F V5 Series Driver (iSCSI, FC):** `missing`
- **Huawei FusionStorage, OceanStor 100D Driver (dsware):** `missing`
- **Huawei T Series V1 Driver (iSCSI, FC):** `missing`
- **Huawei T Series V2 Driver (iSCSI, FC):** `missing`
- **Huawei V3 Series Driver (iSCSI, FC):** `missing`
- **Huawei V5 Series Driver (iSCSI, FC):** `missing`
- **IBM DS8000 Family Storage Driver (FC):** `complete`
- **IBM FlashSystem Driver (iSCSI):** `missing`
- **IBM GPFS Storage Driver (gpfs):** `missing`
- **IBM Spectrum Accelerate Family Driver (iSCSI, FC):** `complete`
- **IBM Storage Virtualize family Driver (iSCSI, FC):** `complete`
- **Infinidat Storage Driver (iSCSI, FC):** `complete`
- **Inspur AS/HF Series Driver (iSCSI, FC):** `missing`
- **Inspur AS13000 Storage Driver (iSCSI):** `complete`
- **Kaminario Storage Driver (iSCSI, FC):** `missing`
- **Kioxia Kumoscale Driver (NVMeOF):** `missing`
- **LINBIT DRBD/LINSTOR Driver (DRBD):** `missing`
- **Lenovo Storage Driver (FC, iSCSI):** `complete`
- **Lightbits Storage Driver (NVMeTCP):** `complete`
- **Logical Volume Manager (LVM) Reference Driver (iSCSI):** `complete`
- **MacroSAN Storage Driver (iSCSI, FC):** `missing`
- **NEC Storage M Series Driver (iSCSI, FC):** `complete`
- **NEC Storage V Series Driver (iSCSI, FC):** `complete`
- **NetApp Data ONTAP Driver (NFS):** `complete`
- **NetApp Data ONTAP Driver (NVMe/TCP):** `complete`
- **NetApp Data ONTAP Driver (iSCSI,FC):** `complete`
- **NetApp Solidfire Driver (iSCSI):** `complete`
- **Nexenta Driver (iSCSI, NFS):** `missing`
- **Open-E JovianDSS Storage Driver (iSCSI):** `complete`
- **ProphetStor Flexvisor Driver (iSCSI, NFS):** `missing`
- **Pure Storage Driver (iSCSI, FC, NVMe-RoCE, NVMe-TCP):** `complete`
- **QNAP Storage Driver (iSCSI):** `missing`
- **Quobyte Storage Driver (quobyte):** `missing`
- **RBD (Ceph) Storage Driver (RBD):** `complete`

- **SandStone Storage Driver (iSCSI):** `complete`
- **Seagate Driver (iSCSI, FC):** `complete`
- **StorPool Storage Driver (storpool):** `complete`
- **Synology Storage Driver (iSCSI):** `missing`
- **TOYOU NetStor Storage Driver (iSCSI, FC):** `complete`
- **TOYOU NetStor TYDS Storage Driver (iSCSI):** `missing`
- **VMware Storage Driver (vmdk):** `missing`
- **Veritas Access iSCSI Driver (iSCSI):** `missing`
- **Veritas Cluster NFS Driver (NFS):** `missing`
- **Virtuozzo Storage Driver (remotefs):** `missing`
- **Windows SMB Driver:** `missing`
- **Windows iSCSI Driver:** `missing`
- **Yadro Tatlin Unified Driver (iSCSI, FC):** `complete`
- **Zadara Storage Driver (iSCSI, NFS):** `complete`
- **infortrend Storage Driver (iSCSI, FC):** `complete`

- **Revert to Snapshot Status: optional.**

  Notes: Vendor drivers that implement the driver assisted function to revert a volume to the last snapshot taken.

  Driver Support:

  - **(Ceph) iSCSI Storage Driver (iSCSI):** `complete`
  - **DataCore Storage Driver (FC, iSCSI):** `complete`
  - **Datera Storage Driver (iSCSI):** `missing`
  - **Dell PowerFlex (ScaleIO) Storage Driver (ScaleIO):** `complete`
  - **Dell PowerMax (2000, 8000) Storage Driver (iSCSI, FC):** `complete`
  - **Dell PowerStore NFS Driver (NFS):** `missing`
  - **Dell PowerStore Storage Driver (iSCSI, FC, NVMe-TCP):** `complete`
  - **Dell PowerVault ME Series (iSCSI, FC):** `missing`
  - **Dell SC Series Storage Driver (iSCSI, FC):** `missing`
  - **Dell Unity Storage Driver (FC, iSCSI):** `complete`
  - **Dell VMAX Af (250F, 450F, 850F, 950F) Storage Driver (FC, iSCSI):** `complete`
  - **Dell VMAX3 (100K, 200K, 400K) Storage Driver (iSCSI, FC):** `complete`
  - **Dell VNX Storage Driver (FC, iSCSI):** `complete`
  - **Dell XtremeIO Storage Driver (FC, iSCSI):** `missing`
  - **Fujitsu ETERNUS Driver (FC, iSCSI):** `missing`
  - **Fungible Storage Driver (NVMe-TCP):** `missing`

- **Generic NFS Reference Driver (NFS):** `missing`
- **HPE 3PAR Storage Driver (FC, iSCSI):** `complete`
- **HPE MSA Driver (iSCSI, FC):** `missing`
- **HPE Nimble Storage Driver (iSCSI, FC):** `complete`
- **HPE XP Storage Driver (FC, iSCSI):** `complete`
- **Hitachi VSP Storage Driver (FC, iSCSI):** `complete`
- **Huawei 18000 Series Driver (iSCSI, FC):** `missing`
- **Huawei Dorado V3, V6 Series Driver (iSCSI, FC):** `missing`
- **Huawei F V3 Series Driver (iSCSI, FC):** `missing`
- **Huawei F V5 Series Driver (iSCSI, FC):** `missing`
- **Huawei FusionStorage, OceanStor 100D Driver (dsware):** `missing`
- **Huawei T Series V1 Driver (iSCSI, FC):** `missing`
- **Huawei T Series V2 Driver (iSCSI, FC):** `missing`
- **Huawei V3 Series Driver (iSCSI, FC):** `missing`
- **Huawei V5 Series Driver (iSCSI, FC):** `missing`
- **IBM DS8000 Family Storage Driver (FC):** `complete`
- **IBM FlashSystem Driver (iSCSI):** `missing`
- **IBM GPFS Storage Driver (gpfs):** `missing`
- **IBM Spectrum Accelerate Family Driver (iSCSI, FC):** `missing`
- **IBM Storage Virtualize family Driver (iSCSI, FC):** `complete`
- **Infinidat Storage Driver (iSCSI, FC):** `complete`
- **Inspur AS/HF Series Driver (iSCSI, FC):** `missing`
- **Inspur AS13000 Storage Driver (iSCSI):** `missing`
- **Kaminario Storage Driver (iSCSI, FC):** `missing`
- **Kioxia Kumoscale Driver (NVMeOF):** `missing`
- **LINBIT DRBD/LINSTOR Driver (DRBD):** `missing`
- **Lenovo Storage Driver (FC, iSCSI):** `missing`
- **Lightbits Storage Driver (NVMeTCP):** `missing`
- **Logical Volume Manager (LVM) Reference Driver (iSCSI):** `complete`
- **MacroSAN Storage Driver (iSCSI, FC):** `missing`
- **NEC Storage M Series Driver (iSCSI, FC):** `complete`
- **NEC Storage V Series Driver (iSCSI, FC):** `complete`
- **NetApp Data ONTAP Driver (NFS):** `complete`
- **NetApp Data ONTAP Driver (NVMe/TCP):** `complete`

- **NetApp Data ONTAP Driver (iSCSI,FC):** `complete`
- **NetApp Solidfire Driver (iSCSI):** `complete`
- **Nexenta Driver (iSCSI, NFS):** `missing`
- **Open-E JovianDSS Storage Driver (iSCSI):** `missing`
- **ProphetStor Flexvisor Driver (iSCSI, NFS):** `missing`
- **Pure Storage Driver (iSCSI, FC, NVMe-RoCE, NVMe-TCP):** `complete`
- **QNAP Storage Driver (iSCSI):** `missing`
- **Quobyte Storage Driver (quobyte):** `missing`
- **RBD (Ceph) Storage Driver (RBD):** `complete`
- **SandStone Storage Driver (iSCSI):** `complete`
- **Seagate Driver (iSCSI, FC):** `missing`
- **StorPool Storage Driver (storpool):** `complete`
- **Synology Storage Driver (iSCSI):** `missing`
- **TOYOU NetStor Storage Driver (iSCSI, FC):** `complete`
- **TOYOU NetStor TYDS Storage Driver (iSCSI):** `missing`
- **VMware Storage Driver (vmdk):** `complete`
- **Veritas Access iSCSI Driver (iSCSI):** `missing`
- **Veritas Cluster NFS Driver (NFS):** `missing`
- **Virtuozzo Storage Driver (remotefs):** `missing`
- **Windows SMB Driver:** `missing`
- **Windows iSCSI Driver:** `missing`
- **Yadro Tatlin Unified Driver (iSCSI, FC):** `complete`
- **Zadara Storage Driver (iSCSI, NFS):** `missing`
- **infortrend Storage Driver (iSCSI, FC):** `missing`

• **Active/Active High Availability Support Status: optional.**

Notes: Vendor drivers that support running in an active/active high availability mode. Indicating support for this means that the driver doesnt contain things, such as local locks, that may impact an active/active configuration and that the driver has been tested to function properly in such a configuration.

Driver Support:

- **(Ceph) iSCSI Storage Driver (iSCSI):** `complete`
- **DataCore Storage Driver (FC, iSCSI):** `missing`
- **Datera Storage Driver (iSCSI):** `missing`
- **Dell PowerFlex (ScaleIO) Storage Driver (ScaleIO):** `complete`
- **Dell PowerMax (2000, 8000) Storage Driver (iSCSI, FC):** `complete`

- **Dell PowerStore NFS Driver (NFS):** `missing`
- **Dell PowerStore Storage Driver (iSCSI, FC, NVMe-TCP):** `complete`
- **Dell PowerVault ME Series (iSCSI, FC):** `missing`
- **Dell SC Series Storage Driver (iSCSI, FC):** `missing`
- **Dell Unity Storage Driver (FC, iSCSI):** `missing`
- **Dell VMAX Af (250F, 450F, 850F, 950F) Storage Driver (FC, iSCSI):** `missing`
- **Dell VMAX3 (100K, 200K, 400K) Storage Driver (iSCSI, FC):** `missing`
- **Dell VNX Storage Driver (FC, iSCSI):** `missing`
- **Dell XtremeIO Storage Driver (FC, iSCSI):** `missing`
- **Fujitsu ETERNUS Driver (FC, iSCSI):** `missing`
- **Fungible Storage Driver (NVMe-TCP):** `missing`
- **Generic NFS Reference Driver (NFS):** `missing`
- **HPE 3PAR Storage Driver (FC, iSCSI):** `missing`
- **HPE MSA Driver (iSCSI, FC):** `missing`
- **HPE Nimble Storage Driver (iSCSI, FC):** `missing`
- **HPE XP Storage Driver (FC, iSCSI):** `missing`
- **Hitachi VSP Storage Driver (FC, iSCSI):** `missing`
- **Huawei 18000 Series Driver (iSCSI, FC):** `missing`
- **Huawei Dorado V3, V6 Series Driver (iSCSI, FC):** `missing`
- **Huawei F V3 Series Driver (iSCSI, FC):** `missing`
- **Huawei F V5 Series Driver (iSCSI, FC):** `missing`
- **Huawei FusionStorage, OceanStor 100D Driver (dsware):** `missing`
- **Huawei T Series V1 Driver (iSCSI, FC):** `missing`
- **Huawei T Series V2 Driver (iSCSI, FC):** `missing`
- **Huawei V3 Series Driver (iSCSI, FC):** `missing`
- **Huawei V5 Series Driver (iSCSI, FC):** `missing`
- **IBM DS8000 Family Storage Driver (FC):** `missing`
- **IBM FlashSystem Driver (iSCSI):** `missing`
- **IBM GPFS Storage Driver (gpfs):** `missing`
- **IBM Spectrum Accelerate Family Driver (iSCSI, FC):** `missing`
- **IBM Storage Virtualize family Driver (iSCSI, FC):** `missing`
- **Infinidat Storage Driver (iSCSI, FC):** `missing`
- **Inspur AS/HF Series Driver (iSCSI, FC):** `missing`
- **Inspur AS13000 Storage Driver (iSCSI):** `missing`

- **Kaminario Storage Driver (iSCSI, FC):** `missing`
- **Kioxia Kumoscale Driver (NVMeOF):** `missing`
- **LINBIT DRBD/LINSTOR Driver (DRBD):** `missing`
- **Lenovo Storage Driver (FC, iSCSI):** `missing`
- **Lightbits Storage Driver (NVMeTCP):** `complete`
- **Logical Volume Manager (LVM) Reference Driver (iSCSI):** `missing`
- **MacroSAN Storage Driver (iSCSI, FC):** `complete`
- **NEC Storage M Series Driver (iSCSI, FC):** `missing`
- **NEC Storage V Series Driver (iSCSI, FC):** `missing`
- **NetApp Data ONTAP Driver (NFS):** `complete`
- **NetApp Data ONTAP Driver (NVMe/TCP):** `missing`
- **NetApp Data ONTAP Driver (iSCSI,FC):** `complete`
- **NetApp Solidfire Driver (iSCSI):** `complete`
- **Nexenta Driver (iSCSI, NFS):** `missing`
- **Open-E JovianDSS Storage Driver (iSCSI):** `missing`
- **ProphetStor Flexvisor Driver (iSCSI, NFS):** `missing`
- **Pure Storage Driver (iSCSI, FC, NVMe-RoCE, NVMe-TCP):** `complete`
- **QNAP Storage Driver (iSCSI):** `missing`
- **Quobyte Storage Driver (quobyte):** `missing`
- **RBD (Ceph) Storage Driver (RBD):** `complete`
- **SandStone Storage Driver (iSCSI):** `complete`
- **Seagate Driver (iSCSI, FC):** `missing`
- **StorPool Storage Driver (storpool):** `missing`
- **Synology Storage Driver (iSCSI):** `missing`
- **TOYOU NetStor Storage Driver (iSCSI, FC):** `missing`
- **TOYOU NetStor TYDS Storage Driver (iSCSI):** `missing`
- **VMware Storage Driver (vmdk):** `missing`
- **Veritas Access iSCSI Driver (iSCSI):** `missing`
- **Veritas Cluster NFS Driver (NFS):** `missing`
- **Virtuozzo Storage Driver (remotefs):** `missing`
- **Windows SMB Driver:** `missing`
- **Windows iSCSI Driver:** `missing`
- **Yadro Tatlin Unified Driver (iSCSI, FC):** `complete`
- **Zadara Storage Driver (iSCSI, NFS):** `missing`

  - **infortrend Storage Driver (iSCSI, FC):** `missing`

Notes:

- **This document is a continuous work in progress**

Driver Removal History

The section will be used to track driver removal starting from the Rocky release.

- **Rocky**
  - CoprHD Storage Driver (FC, iSCSI, ScaleIO)
- **Stein**
  - DRBDManage Driver
  - HGST Flash Storage Suite Driver (vgc)
  - ITRI DISCO Driver
  - NetApp E-Series Driver
- **Train**
  - Tintri Storage Driver
  - Veritas HyperScale Storage Driver
  - Nexenta Edge Storage Driver
- **Ussuri**
  - HPE Lefthand Driver (iSCSI)
  - Sheepdog Driver

Available Drivers

Volume Drivers

Supported Drivers

AS13000Driver

- Version: 1.0.0
- volume_driver=cinder.volume.drivers.inspur.as13000.as13000_driver.AS13000Driver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/Inspur_CI
- Driver Configuration Options:

Table 134: **Driver configuration options**

Name = Default Value	(Type) Description
as13000_ipsan_pools = [Pool0]	(List of String) The Storage Pools Cinder should use, a comma separated list.
as13000_meta_pool = None	(String) The pool which is used as a meta pool when creating a volume, and it should be a replication pool at present. If not set, the driver will choose a replication pool from the value of as13000_ipsan_pools.
as13000_token_available_time = 3300	(Integer(min=600, max=3600)) The effective time of token validity in seconds.

- Description: Driver for Inspur AS13000 storage.

```
Version history:
    1.0.0 - Initial driver
```

Acs5000FCDriver

- Version: 1.0.0
- volume_driver=cinder.volume.drivers.toyou.acs5000.acs5000_fc.Acs5000FCDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/TOYOU_ACS5000_CI
- Driver Configuration Options:

Table 135: **Driver configuration options**

Name = Default Value	(Type) Description
acs5000_copy_interval = 5	(Integer(min=3, max=100)) When volume copy task is going on,refresh volume status interval
acs5000_multiattach = False	(Boolean) Enable to allow volumes attaching to multiple hosts with no limit.
acs5000_volpool_name = [pool01]	(List of String) Comma separated list of storage system storage pools for volumes.
san_ip =	(String) IP address of SAN controller
san_login = admin	(String) Username for SAN controller
san_password =	(String) Password for SAN controller
san_ssh_port = 22	(Port(min=0, max=65535)) SSH port to use with SAN
ssh_conn_timeout = 30	(Integer) SSH connection timeout in seconds
ssh_max_pool_conn = 5	(Integer) Maximum ssh connections in the pool
ssh_min_pool_conn = 1	(Integer) Minimum ssh connections in the pool

- Description: TOYOU ACS5000 storage FC volume driver.

```
Version history:
    1.0.0 - Initial driver
```

Acs5000ISCSIDriver

- Version: 1.0.0
- volume_driver=cinder.volume.drivers.toyou.acs5000.acs5000_iscsi.Acs5000ISCSIDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/TOYOU_ACS5000_CI
- Driver Configuration Options:

Table 136: **Driver configuration options**

Name = Default Value	(Type) Description
acs5000_copy_interval = 5	(Integer(min=3, max=100)) When volume copy task is going on,refresh volume status interval
acs5000_multiattach = False	(Boolean) Enable to allow volumes attaching to multiple hosts with no limit.
acs5000_volpool_name = [pool01]	(List of String) Comma separated list of storage system storage pools for volumes.
san_ip =	(String) IP address of SAN controller
san_login = admin	(String) Username for SAN controller
san_password =	(String) Password for SAN controller
san_ssh_port = 22	(Port(min=0, max=65535)) SSH port to use with SAN
ssh_conn_timeout = 30	(Integer) SSH connection timeout in seconds
ssh_max_pool_conn = 5	(Integer) Maximum ssh connections in the pool
ssh_min_pool_conn = 1	(Integer) Minimum ssh connections in the pool

- Description: TOYOU ACS5000 storage iSCSI volume driver.

```
Version history:
    1.0.0 - Initial driver
```

DSWAREDriver

- Version: 2.0
- volume_driver=cinder.volume.drivers.fusionstorage.dsware.DSWAREDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/Huawei_FusionStorage_CI
- Driver Configuration Options:

Table 137: **Driver configuration options**

Name = Default Value	(Type) Description
clone_volume_timeout = 680	(Integer) Create clone volume timeout
dsware_isthin = False	(Boolean) The flag of thin storage allocation.
dsware_manager =	(String) Fusionstorage manager ip addr for cinder-volume.
dsware_rest_url =	(String) The address of FusionStorage array. For example, dsware_rest_url=xxx
dsware_storage_pools =	(String) The list of pools on the FusionStorage array, the semicolon(;) was used to split the storage pools, dsware_storage_pools = xxx1; xxx2; xxx3
fusionstorageagent =	(String) Fusionstorage agent ip addr range
manager_ips = { }	(Dict of String) This option is to support the FSA to mount across the different nodes. The parameters takes the standard dict config form, manager_ips = host1:ip1, host2:ip2
pool_id_filter = []	(List of String) Pool id permit to use
pool_type = default	(String) Pool type, like sata-2copy

- Description: <None>

DataCoreVolumeDriver

- Version: N/A
- volume_driver=cinder.volume.drivers.datacore.driver.DataCoreVolumeDriver
- Driver Configuration Options:

Table 138: **Driver configuration options**

Name = Default Value	(Type) Description
backend_availability_zone = None	(String) Availability zone for this volume backend. If not set, the storage_availability_zone option value is used as the default for all backends.
chap_password =	(String) Password for specified CHAP account name.
chap_username =	(String) CHAP user name.
chiscsi_conf = /etc/chelsio-iscsi/chiscsi.conf	(String) Chiscsi (CXT) global defaults configuration file
driver_client_cert = None	(String) The path to the client certificate for verification, if the driver supports it.
driver_client_cert_key = None	(String) The path to the client certificate key for verification, if the driver supports it.
driver_data_namespace = None	(String) Namespace for driver private data values to be saved in.
driver_ssl_cert_path = None	(String) Can be used to specify a non default path to a CA_BUNDLE file or directory with certificates of trusted CAs, which will be used to validate the backend
driver_ssl_cert_verify = False	(Boolean) If set to True the http client will validate the SSL certificate of the backend endpoint.
driver_use_ssl = False	(Boolean) Tell driver to use SSL for connection to backend storage if the driver supports it.

continues on next page

Table 138 – continued from previous page

Name = Default Value	(Type) Description
enable_unsupported_driver = False	(Boolean) Set this to True when you want to allow an unsupported driver to start. Drivers that havent maintained a working CI system and testing are marked as unsupported until CI is working again. This also marks a driver as deprecated and may be removed in the next release.
filter_function = None	(String) String representation for an equation that will be used to filter hosts. Only used when the driver filter is set to be used by the Cinder scheduler.
goodness_function = None	(String) String representation for an equation that will be used to determine the goodness of a host. Only used when using the goodness weigher is set to be used by the Cinder scheduler.
iscsi_iotype = fileio	(String(choices=[blockio, fileio, auto])) Sets the behavior of the iSCSI target to either perform blockio or fileio optionally, auto can be set and Cinder will autodetect type of backing device
iscsi_target_flags =	(String) Sets the target-specific flags for the iSCSI target. Only used for tgtadm to specify backing device flags using bsoflags option. The specified string is passed as is to the underlying tool.
iscsi_write_cache = on	(String(choices=[on, off])) Sets the behavior of the iSCSI target to either perform write-back(on) or write-through(off). This parameter is valid if target_helper is set to tgtadm.
max_over_subscription_ratio = 20.0	(String(regex=^(auto|\d*\.\d+|\d+)$)) Representation of the over subscription ratio when thin provisioning is enabled. Default ratio is 20.0, meaning provisioned capacity can be 20 times of the total physical capacity. If the ratio is 10.5, it means provisioned capacity can be 10.5 times of the total physical capacity. A ratio of 1.0 means provisioned capacity cannot exceed the total physical capacity. If ratio is auto, Cinder will automatically calculate the ratio based on the provisioned capacity and the used space. If not set to auto, the ratio has to be a minimum of 1.0.
num_shell_tries = 3	(Integer) Number of times to attempt to run flakey shell commands
num_volume_device_scan_tries = 3	(Integer) The maximum number of times to rescan targets to find volume
replication_device = None	(Dict of String) Multi opt of dictionaries to represent a replication target device. This option may be specified multiple times in a single config section to specify multiple replication target devices. Each entry takes the standard dict config form: replication_device = target_device_id:<required>,key1:value1,key2:value2
report_discard_supported = False	(Boolean) Report to clients of Cinder that the backend supports discard (aka. trim/unmap). This will not actually change the behavior of the backend or the client directly, it will only notify that it can be used.
reserved_percentage = 0	(Integer(min=0, max=100)) The percentage of backend capacity is reserved
storage_protocol = iSCSI	(String(choices=[iSCSI, FC])) Protocol for transferring data between host and storage back-end.

continues on next page

Table 138 – continued from previous page

Name = Default Value	(Type) Description
target_helper = tgtadm	(String(choices=[tgtadm, lioadm, scstadmin, iscsictl, nvmet, spdk-nvmeof, fake])) Target user-land tool to use. tgtadm is default, use lioadm for LIO iSCSI support, scstadmin for SCST target support, iscsictl for Chelsio iSCSI Target, nvmet for NVMEoF support, spdk-nvmeof for SPDK NVMe-oF, or fake for testing.
target_ip_address = $my_ip	(String) The IP address that the iSCSI/NVMEoF daemon is listening on
target_port = 3260	(Port(min=0, max=65535)) The port that the iSCSI/NVMEoF daemon is listening on
target_prefix = iqn.2010-10.org.openstack:	(String) Prefix for iSCSI/NVMEoF volumes
target_protocol = iscsi	(String(choices=[iscsi, iser, nvmet_rdma, nvmet_tcp])) Determines the target protocol for new volumes, created with tgtadm, lioadm and nvmet target helpers. In order to enable RDMA, this parameter should be set with the value iser. The supported iSCSI protocol values are iscsi and iser, in case of nvmet target set to nvmet_rdma or nvmet_tcp.
target_secondary_ip_addresses = []	(List of String) The list of secondary IP addresses of the iSCSI/NVMEoF daemon
trace_flags = None	(List of String) List of options that control which trace info is written to the DEBUG log level to assist developers. Valid values are method and api.
use_chap_auth = False	(Boolean) Option to enable/disable CHAP authentication for targets.
volume_backend_name = None	(String) The backend name for a given driver implementation
volume_clear = zero	(String(choices=[none, zero])) Method used to wipe old volumes
volume_clear_ionice = None	(String) The flag to pass to ionice to alter the i/o priority of the process used to zero a volume after deletion, for example -c3 for idle only priority.
volume_clear_size = 0	(Integer(max=1024)) Size in MiB to wipe at start of old volumes. 1024 MiB at max. 0 => all
volume_copy_blkio_cgroup_name = cinder-volume-copy	(String) The blkio cgroup name to be used to limit bandwidth of volume copy
volume_copy_bps_limit = 0	(Integer) The upper limit of bandwidth of volume copy. 0 => unlimited
volume_dd_blocksize = 1M	(String) The default block size used when copying/clearing volumes
volumes_dir = $state_path/volumes	(String) Volume configuration file storage directory

- Description: DataCore SANsymphony base volume driver.

DateraDriver

- Version: 2019.12.10.0
- volume_driver=cinder.volume.drivers.datera.datera_iscsi.DateraDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/datera-ci

- Driver Configuration Options:

Table 139: **Driver configuration options**

Name = Default Value	(Type) Description
backend_availability_zone = None	(String) Availability zone for this volume backend. If not set, the storage_availability_zone option value is used as the default for all backends.
chap_password =	(String) Password for specified CHAP account name.
chap_username =	(String) CHAP user name.
chiscsi_conf = /etc/chelsio-iscsi/chiscsi.conf	(String) Chiscsi (CXT) global defaults configuration file
driver_client_cert = None	(String) The path to the client certificate for verification, if the driver supports it.
driver_client_cert_key = None	(String) The path to the client certificate key for verification, if the driver supports it.
driver_data_namespace = None	(String) Namespace for driver private data values to be saved in.
driver_ssl_cert_path = None	(String) Can be used to specify a non default path to a CA_BUNDLE file or directory with certificates of trusted CAs, which will be used to validate the backend
driver_ssl_cert_verify = False	(Boolean) If set to True the http client will validate the SSL certificate of the backend endpoint.
driver_use_ssl = False	(Boolean) Tell driver to use SSL for connection to backend storage if the driver supports it.
enable_unsupported_driver = False	(Boolean) Set this to True when you want to allow an unsupported driver to start. Drivers that havent maintained a working CI system and testing are marked as unsupported until CI is working again. This also marks a driver as deprecated and may be removed in the next release.
filter_function = None	(String) String representation for an equation that will be used to filter hosts. Only used when the driver filter is set to be used by the Cinder scheduler.
goodness_function = None	(String) String representation for an equation that will be used to determine the goodness of a host. Only used when using the goodness weigher is set to be used by the Cinder scheduler.
iscsi_iotype = fileio	(String(choices=[blockio, fileio, auto])) Sets the behavior of the iSCSI target to either perform blockio or fileio optionally, auto can be set and Cinder will autodetect type of backing device
iscsi_target_flags =	(String) Sets the target-specific flags for the iSCSI target. Only used for tgtadm to specify backing device flags using bsoflags option. The specified string is passed as is to the underlying tool.
iscsi_write_cache = on	(String(choices=[on, off])) Sets the behavior of the iSCSI target to either perform write-back(on) or write-through(off). This parameter is valid if target_helper is set to tgtadm.

continues on next page

Table 139 – continued from previous page

Name = Default Value	(Type) Description
max_over_subscription_ratio = 20.0	(String(regex=^(auto|\d*\.\d+|\d+)$)) Representation of the over subscription ratio when thin provisioning is enabled. Default ratio is 20.0, meaning provisioned capacity can be 20 times of the total physical capacity. If the ratio is 10.5, it means provisioned capacity can be 10.5 times of the total physical capacity. A ratio of 1.0 means provisioned capacity cannot exceed the total physical capacity. If ratio is auto, Cinder will automatically calculate the ratio based on the provisioned capacity and the used space. If not set to auto, the ratio has to be a minimum of 1.0.
num_shell_tries = 3	(Integer) Number of times to attempt to run flakey shell commands
num_volume_device_scan_tries = 3	(Integer) The maximum number of times to rescan targets to find volume
replication_device = None	(Dict of String) Multi opt of dictionaries to represent a replication target device. This option may be specified multiple times in a single config section to specify multiple replication target devices. Each entry takes the standard dict config form: replication_device = target_device_id:<required>,key1:value1,key2:value2
report_discard_supported = False	(Boolean) Report to clients of Cinder that the backend supports discard (aka. trim/unmap). This will not actually change the behavior of the backend or the client directly, it will only notify that it can be used.
reserved_percentage = 0	(Integer(min=0, max=100)) The percentage of backend capacity is reserved
storage_protocol = iSCSI	(String(choices=[iSCSI, FC])) Protocol for transferring data between host and storage back-end.
target_helper = tgtadm	(String(choices=[tgtadm, lioadm, scstadmin, iscsictl, nvmet, spdk-nvmeof, fake])) Target user-land tool to use. tgtadm is default, use lioadm for LIO iSCSI support, scstadmin for SCST target support, iscsictl for Chelsio iSCSI Target, nvmet for NVMEoF support, spdk-nvmeof for SPDK NVMe-oF, or fake for testing.
target_ip_address = $my_ip	(String) The IP address that the iSCSI/NVMEoF daemon is listening on
target_port = 3260	(Port(min=0, max=65535)) The port that the iSCSI/NVMEoF daemon is listening on
target_prefix = iqn.2010-10.org.openstack:	(String) Prefix for iSCSI/NVMEoF volumes
target_protocol = iscsi	(String(choices=[iscsi, iser, nvmet_rdma, nvmet_tcp])) Determines the target protocol for new volumes, created with tgtadm, lioadm and nvmet target helpers. In order to enable RDMA, this parameter should be set with the value iser. The supported iSCSI protocol values are iscsi and iser, in case of nvmet target set to nvmet_rdma or nvmet_tcp.
target_secondary_ip_addresses = []	(List of String) The list of secondary IP addresses of the iSCSI/NVMEoF daemon
trace_flags = None	(List of String) List of options that control which trace info is written to the DEBUG log level to assist developers. Valid values are method and api.

continues on next page

Table 139 – continued from previous page

Name = Default Value	(Type) Description
use_chap_auth = False	(Boolean) Option to enable/disable CHAP authentication for targets.
volume_backend_name = None	(String) The backend name for a given driver implementation
volume_clear = zero	(String(choices=[none, zero])) Method used to wipe old volumes
volume_clear_ionice = None	(String) The flag to pass to ionice to alter the i/o priority of the process used to zero a volume after deletion, for example -c3 for idle only priority.
volume_clear_size = 0	(Integer(max=1024)) Size in MiB to wipe at start of old volumes. 1024 MiB at max. 0 => all
volume_copy_blkio_cgroup_name = cinder-volume-copy	(String) The blkio cgroup name to be used to limit bandwidth of volume copy
volume_copy_bps_limit = 0	(Integer) The upper limit of bandwidth of volume copy. 0 => unlimited
volume_dd_blocksize = 1M	(String) The default block size used when copying/clearing volumes
volumes_dir = $state_path/volumes	(String) Volume configuration file storage directory

- Description: The OpenStack Datera iSCSI volume driver.

Version history:

- 1.0 - Initial driver
- 1.1 - Look for lun-0 instead of lun-1.
- 2.0 - Update For Datera API v2
- 2.1 - Multipath, ACL and reorg
- **2.2 - Capabilites List, Extended Volume-Type Support**
  Naming convention change, Volume Manage/Unmanage support
- **2.3 - Templates, Tenants, Snapshot Polling,**
  2.1 Api Version Support, Restructure
- 2.3.1 - Scalability bugfixes
- 2.3.2 - Volume Placement, ACL multi-attach bugfix
- 2.4.0 - Fast Retype Support
- 2.5.0 - Glance Image Caching, retyping/QoS bugfixes
- 2.6.0 - Api 2.2 support
- 2.6.1 - Glance interoperability fix
- **2.7.0 - IOPS/GB and BW/GB settings, driver level overrides**
  (API 2.1+ only)
- 2.7.2 - Allowing DF: QoS Spec prefix, QoS type leak bugfix
- 2.7.3 - Fixed bug in clone_image where size was not set correctly

- **2.7.4 - Fix for create_tenant incorrect API call**
  Temporary fix for DAT-15931
- 2.7.5 - Removed force parameter from /initiators v2.1 API requests
- **2.8.0 - iops_per_gb and bandwidth_per_gb are now limited by**
  total_iops_max and total_bandwidth_max (API 2.1+ only) Bugfix for cinder retype with online volume
- 2.8.1 - Bugfix for missing default dict during retype
- 2.8.2 - Updated most retype operations to not detach volume
- **2.8.3 - Bugfix for not allowing fast clones for shared/community**
  volumes
- 2.8.4 - Fixed missing API version pinning in _offline_flip
- 2.8.5 - Membership check for fast image cloning. Metadata API pinning
- 2.8.6 - Added LDAP support and CHAP support
- 2.8.7 - Bugfix for missing tenancy calls in offline_flip
- **2.9.0 - Volumes now correctly renamed during backend migration.**
  Implemented update_migrated_volume (API 2.1+ only), Prevent non-raw image cloning
- **2.9.1 - Added extended metadata attributes during volume creation**
  and attachment. Added datera_disable_extended_metadata option to disable it.
- **2.9.2 - Made ensure_export a no-op. Removed usage of**
  initiator-groups
- **2018.4.5.0 - Switch to new date-based versioning scheme. Removed v2**
  API support
- 2018.4.17.1 - Bugfixes to IP Pools, Templates and Initiators
- 2018.4.25.0 - Snapshot Manage. List Manageable Snapshots support
- **2018.4.27.0 - Major driver revamp/restructure, no functionality**
  change
- 2018.5.1.0 - Bugfix for Map tenant auto-creation
- 2018.5.18.0 - Bugfix for None tenant handling
- 2018.6.7.0 - Bugfix for missing project_id during image clone
- 2018.7.13.0 - Massive update porting to use the Datera Python-SDK
- **2018.7.20.0 - Driver now includes display_name in created backend**
  app_instances.
- 2018.9.17.0 - Requirements and doc changes
- **2018.10.8.0 - Added extra_headers to Python-SDK constructor call.**
  This allows for the SDK to send the type of driver performing each request along with the request. This functionality existed before the Python-SDK revamp, so this change adds the functionality back in.
- **2018.10.8.1 - Adding thread_local to Python-SDK constructor call.**
  This preserves trace_id in the logs

- **2018.10.30.0 - Adding template_override support. Added**
  datera_disable_template_override cfgOpt to disable this feature. Updated required requests version to >=2.20.0 because of a security vulnerability in <=2.19.X. Added support for filter_function and goodness_function.
- **2018.11.1.0 - Adding flash and hybrid capacity info to**
  get_volume_stats
- 2018.11.8.0 - Fixing bug that broke 2.2.X support
- **2018.11.14.0 - Bugfixes for v2.1 API support and unicode character**
  support
- 2019.1.24.0 - Python-SDK requirements update, README updates
- 2019.2.25.0 - Scalability fixes and utility script updates
- 2019.6.4.1 - Added Pypi packaging installation support
- **2019.12.10.0 - Python 3.x support, tox tests, CI ready, live**
  migration support, image cache, bugfixes.

FJDXFCDriver

- Version: 1.4.8
- volume_driver=cinder.volume.drivers.fujitsu.eternus_dx.eternus_dx_fc.FJDXFCDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/Fujitsu_ETERNUS_CI
- Driver Configuration Options:

Table 140: **Driver configuration options**

Name = Default Value	(Type) Description
cinder_eternus_config_file = /etc/cinder/cinder_fujitsu_etern	(String) Config file for cinder eternus_dx volume driver.
fujitsu_passwordless = True	(Boolean) Use SSH key to connect to storage.
fujitsu_private_key_path = $state_path/eternus	(String) Filename of private key for ETERNUS CLI. This option must be set when the fujitsu_passwordless is True.
fujitsu_use_cli_copy = False	(Boolean) If True use CLI command to create snapshot.

- Description: FC Cinder Volume Driver for Fujitsu ETERNUS DX S3 series.

FJDXISCSIDriver

- Version: 1.4.8
- volume_driver=cinder.volume.drivers.fujitsu.eternus_dx.eternus_dx_iscsi.FJDXISCSIDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/Fujitsu_ETERNUS_CI
- Driver Configuration Options:

Table 141: **Driver configuration options**

Name = Default Value	(Type) Description
cinder_eternus_config_file = /etc/cinder/cinder_fujitsu_etern	(String) Config file for cinder eternus_dx volume driver.
fujitsu_passwordless = True	(Boolean) Use SSH key to connect to storage.
fujitsu_private_key_path = $state_path/eternus	(String) Filename of private key for ETERNUS CLI. This option must be set when the fujitsu_passwordless is True.
fujitsu_use_cli_copy = False	(Boolean) If True use CLI command to create snapshot.

- Description: iSCSI Cinder Volume Driver for Fujitsu ETERNUS DX S3 series.

FibreChannelVolumeDriver

- Version: 2.0.0
- volume_driver=cinder.volume.drivers.datacore.fc.FibreChannelVolumeDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/DataCore_CI
- Driver Configuration Options:

Table 142: **Driver configuration options**

Name = Default Value	(Type) Description
datacore_api_timeout = 300	(Integer(min=1)) Seconds to wait for a response from a DataCore API call.
datacore_disk_failed_delay = 300	(Integer(min=0)) Seconds to wait for DataCore virtual disk to come out of the Failed state.
datacore_disk_pools = []	(List of String) List of DataCore disk pools that can be used by volume driver.
datacore_disk_type = single	(String(choices=[single, mirrored])) DataCore virtual disk type (single/mirrored). Mirrored virtual disks require two storage servers in the server group.
datacore_fc_unallowed_targets = []	(List of String) List of FC targets that cannot be used to attach volume. To prevent the DataCore FibreChannel volume driver from using some front-end targets in volume attachment, specify this option and list the iqn and target machine for each target as the value, such as <wwpns:target name>, <wwpns:target name>, <wwpns:target name>.
datacore_storage_profile = None	(String) DataCore virtual disk storage profile.
san_ip =	(String) IP address of SAN controller
san_login = admin	(String) Username for SAN controller
san_password =	(String) Password for SAN controller

- Description: DataCore SANsymphony Fibre Channel volume driver.

Version history:

```
1.0.0 - Initial driver
2.0.0 - Reintroduce the driver
```

FungibleDriver

- Version: 1.0.0
- volume_driver=cinder.volume.drivers.fungible.driver.FungibleDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/Fungible_Storage_CI
- Driver Configuration Options:

Table 143: **Driver configuration options**

Name = Default Value	(Type) Description
api_enable_ssl = True	(Boolean) Specify whether to use SSL or not when accessing the composer APIs
fsc_clone_volume_timeout = 1800	(Integer) Create clone volume timeout in seconds
iops_for_image_migration = 250000	(Integer) Maximum read IOPS that volume can get when reading data from the volume during host assisted migration
nvme_connect_port = 4420	(Port(min=0, max=65535)) The port number to be used when doing nvme connect from host
san_api_port = None	(Port(min=0, max=65535)) Port to use to access the SAN API
san_ip =	(String) IP address of SAN controller
san_login = admin	(String) Username for SAN controller
san_password =	(String) Password for SAN controller

- Description: Fungible Storage driver

Fungible driver is a volume driver for Fungible Storage.

Version history:
1.0.0 - First source driver version

GPFSDriver

- Version: 1.3.1
- volume_driver=cinder.volume.drivers.ibm.gpfs.GPFSDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/IBM_GPFS_CI
- Driver Configuration Options:

Table 144: **Driver configuration options**

Name = Default Value	(Type) Description
gpfs_images_dir = None	(String) Specifies the path of the Image service repository in GPFS. Leave undefined if not storing images in GPFS.
gpfs_images_share_mode = None	(String(choices=[copy, copy_on_write, None])) Specifies the type of image copy to be used. Set this when the Image service repository also uses GPFS so that image files can be transferred efficiently from the Image service to the Block Storage service. There are two valid values: copy specifies that a full copy of the image is made; copy_on_write specifies that copy-on-write optimization strategy is used and unmodified blocks of the image file are shared efficiently.
gpfs_max_clone_depth = 0	(Integer) Specifies an upper limit on the number of indirections required to reach a specific block due to snapshots or clones. A lengthy chain of copy-on-write snapshots or clones can have a negative impact on performance, but improves space utilization. 0 indicates unlimited clone depth.
gpfs_mount_point_base = None	(String) Specifies the path of the GPFS directory where Block Storage volume and snapshot files are stored.
gpfs_sparse_volumes = True	(Boolean) Specifies that volumes are created as sparse files which initially consume no space. If set to False, the volume is created as a fully allocated file, in which case, creation may take a significantly longer time.
gpfs_storage_pool = system	(String) Specifies the storage pool that volumes are assigned to. By default, the system storage pool is used.

- Description: Implements volume functions using GPFS primitives.

```
Version history:

  1.0.0 - Initial driver
  1.1.0 - Add volume retype, refactor volume migration
  1.2.0 - Add consistency group support
  1.3.0 - Add NFS based GPFS storage backend support
  1.3.1 - Add GPFS native encryption (encryption of data at rest) support
```

GPFSNFSDriver

- Version: 1.0
- volume_driver=cinder.volume.drivers.ibm.gpfs.GPFSNFSDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/IBM_GPFS_CI
- Driver Configuration Options:

Table 145: **Driver configuration options**

Name = Default Value	(Type) Description
gpfs_images_dir = None	(String) Specifies the path of the Image service repository in GPFS. Leave undefined if not storing images in GPFS.
gpfs_images_share_mode = None	(String(choices=[copy, copy_on_write, None])) Specifies the type of image copy to be used. Set this when the Image service repository also uses GPFS so that image files can be transferred efficiently from the Image service to the Block Storage service. There are two valid values: copy specifies that a full copy of the image is made; copy_on_write specifies that copy-on-write optimization strategy is used and unmodified blocks of the image file are shared efficiently.
gpfs_max_clone_depth = 0	(Integer) Specifies an upper limit on the number of indirections required to reach a specific block due to snapshots or clones. A lengthy chain of copy-on-write snapshots or clones can have a negative impact on performance, but improves space utilization. 0 indicates unlimited clone depth.
gpfs_mount_point_base = None	(String) Specifies the path of the GPFS directory where Block Storage volume and snapshot files are stored.
gpfs_sparse_volumes = True	(Boolean) Specifies that volumes are created as sparse files which initially consume no space. If set to False, the volume is created as a fully allocated file, in which case, creation may take a significantly longer time.
gpfs_storage_pool = system	(String) Specifies the storage pool that volumes are assigned to. By default, the system storage pool is used.

- Description: GPFS cinder driver extension.

This extends the capability of existing GPFS cinder driver to be able to create cinder volumes when cinder volume service is not running on GPFS node.

GPFSRemoteDriver

- Version: 1.0
- volume_driver=cinder.volume.drivers.ibm.gpfs.GPFSRemoteDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/IBM_GPFS_CI
- Driver Configuration Options:

Table 146: **Driver configuration options**

Name = Default Value	(Type) Description
gpfs_hosts = []	(List of String) Comma-separated list of IP address or hostnames of GPFS nodes.
gpfs_hosts_key_file = $state_path/ssh_known_hosts	(String) File containing SSH host keys for the gpfs nodes with which driver needs to communicate. Default=$state_path/ssh_known_hosts
gpfs_images_dir = None	(String) Specifies the path of the Image service repository in GPFS. Leave undefined if not storing images in GPFS.
gpfs_images_share_mode = None	(String(choices=[copy, copy_on_write, None])) Specifies the type of image copy to be used. Set this when the Image service repository also uses GPFS so that image files can be transferred efficiently from the Image service to the Block Storage service. There are two valid values: copy specifies that a full copy of the image is made; copy_on_write specifies that copy-on-write optimization strategy is used and unmodified blocks of the image file are shared efficiently.
gpfs_max_clone_depth = 0	(Integer) Specifies an upper limit on the number of indirections required to reach a specific block due to snapshots or clones. A lengthy chain of copy-on-write snapshots or clones can have a negative impact on performance, but improves space utilization. 0 indicates unlimited clone depth.
gpfs_mount_point_base = None	(String) Specifies the path of the GPFS directory where Block Storage volume and snapshot files are stored.
gpfs_private_key =	(String) Filename of private key to use for SSH authentication.
gpfs_sparse_volumes = True	(Boolean) Specifies that volumes are created as sparse files which initially consume no space. If set to False, the volume is created as a fully allocated file, in which case, creation may take a significantly longer time.
gpfs_ssh_port = 22	(Port(min=0, max=65535)) SSH port to use.
gpfs_storage_pool = system	(String) Specifies the storage pool that volumes are assigned to. By default, the system storage pool is used.
gpfs_strict_host_key_policy = False	(Boolean) Option to enable strict gpfs host key checking while connecting to gpfs nodes. Default=False
gpfs_user_login = root	(String) Username for GPFS nodes.
gpfs_user_password =	(String) Password for GPFS node user.

- Description: GPFS cinder driver extension.

This extends the capability of existing GPFS cinder driver to be able to run the driver when cinder volume service is not running on GPFS node where as Nova Compute is a GPFS client. This deployment is typically in Container based OpenStack environment.

HBSDFCDriver

- Version: 2.4.0
- volume_driver=cinder.volume.drivers.hitachi.hbsd_fc.HBSDFCDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/Hitachi_VSP_CI
- Driver Configuration Options:

Table 147: **Driver configuration options**

Name = Default Value	(Type) Description
chap_password =	(String) Password for specified CHAP account name.
chap_username =	(String) CHAP user name.
driver_ssl_cert_path = None	(String) Can be used to specify a non default path to a CA_BUNDLE file or directory with certificates of trusted CAs, which will be used to validate the backend
driver_ssl_cert_verify = False	(Boolean) If set to True the http client will validate the SSL certificate of the backend endpoint.
en-force_multipath_for_image_xfe = False	(Boolean) If this is set to True, attachment of volumes for image transfer will be aborted when multipathd is not running. Otherwise, it will fallback to single path. This parameter needs to be configured for each backend section or in [backend_defaults] section as a common configuration for all backends.
hi-tachi_async_copy_check_interv = 10	(Integer(min=1, max=600)) Interval in seconds to check asynchronous copying status during a copy pair deletion or data restoration.
hitachi_compute_target_ports = []	(List of String) IDs of the storage ports used to attach volumes to compute nodes. To specify multiple ports, connect them by commas (e.g. CL1-A,CL2-A).
hitachi_copy_check_interval = 3	(Integer(min=1, max=600)) Interval in seconds to check copying status during a volume copy.
hitachi_copy_speed = 3	(Integer(min=1, max=15)) Copy speed of storage system. 1 or 2 indicates low speed, 3 indicates middle speed, and a value between 4 and 15 indicates high speed.
hitachi_discard_zero_page = True	(Boolean) Enable or disable zero page reclamation in a DP-VOL.
hitachi_exec_retry_interval = 5	(Integer) Retry interval in seconds for REST API execution.
hitachi_extend_timeout = 600	(Integer) Maximum wait time in seconds for a volume extention to complete.
hitachi_group_create = False	(Boolean) If True, the driver will create host groups or iSCSI targets on storage ports as needed.
hitachi_group_delete = False	(Boolean) If True, the driver will delete host groups or iSCSI targets on storage ports as needed.
hitachi_group_name_format = None	(String) Format of host groups, iSCSI targets, and server objects.
hitachi_host_mode_options = []	(List of Integer) Host mode option for host group or iSCSI target.
hitachi_ldev_range = None	(String) Range of the LDEV numbers in the format of xxxx-yyyy that can be used by the driver. Values can be in decimal format (e.g. 1000) or in colon-separated hexadecimal format (e.g. 00:03:E8).
hitachi_lock_timeout = 7200	(Integer) Maximum wait time in seconds for storage to be logined or unlocked.
hitachi_lun_retry_interval = 1	(Integer) Retry interval in seconds for REST API adding a LUN mapping to the server.
hitachi_lun_timeout = 50	(Integer) Maximum wait time in seconds for adding a LUN mapping to the server.

continues on next page

Table 147 – continued from previous page

Name = Default Value	(Type) Description
hitachi_mirror_auth_password = None	(String) iSCSI authentication password
hitachi_mirror_auth_user = None	(String) iSCSI authentication username
hitachi_mirror_compute_target_p = []	(List of String) Target port names of compute node for host group or iSCSI target
hitachi_mirror_ldev_range = None	(String) Logical device range of secondary storage system
hitachi_mirror_pair_target_numb = 0	(Integer(min=0, max=99)) Pair target name of the host group or iSCSI target
hitachi_mirror_pool = None	(String) Pool of secondary storage system
hitachi_mirror_rest_api_ip = None	(String) IP address of REST API server
hitachi_mirror_rest_api_port = 443	(Port(min=0, max=65535)) Port number of REST API server
hitachi_mirror_rest_pair_target_p = []	(List of String) Target port names for pair of the host group or iSCSI target
hitachi_mirror_rest_password = None	(String) Password of secondary storage system for REST API
hitachi_mirror_rest_user = None	(String) Username of secondary storage system for REST API
hitachi_mirror_snap_pool = None	(String) Thin pool of secondary storage system
hitachi_mirror_ssl_cert_path = None	(String) Can be used to specify a non default path to a CA_BUNDLE file or directory with certificates of trusted CAs, which will be used to validate the backend
hitachi_mirror_ssl_cert_verify = False	(Boolean) If set to True the http client will validate the SSL certificate of the backend endpoint.
hitachi_mirror_storage_id = None	(String) ID of secondary storage system
hitachi_mirror_target_ports = []	(List of String) Target port names for host group or iSCSI target
hitachi_mirror_use_chap_auth = False	(Boolean) Whether or not to use iSCSI authentication
hitachi_pair_target_number = 0	(Integer(min=0, max=99)) Pair target name of the host group or iSCSI target
hitachi_path_group_id = 0	(Integer(min=0, max=255)) Path group ID assigned to the remote connection for remote replication
hitachi_pools = []	(List of String) Pool number[s] or pool name[s] of the DP pool.
hitachi_port_scheduler = False	(Boolean) Enable port scheduling of WWNs to the configured ports so that WWNs are registered to ports in a round-robin fashion.

continues on next page

Table 147 – continued from previous page

Name = Default Value	(Type) Description
hitachi_quorum_disk_id = None	(Integer(min=0, max=31)) ID of the Quorum disk used for global-active device
hi-tachi_replication_copy_speed = 3	(Integer(min=1, max=15)) Remote copy speed of storage system. 1 or 2 indicates low speed, 3 indicates middle speed, and a value between 4 and 15 indicates high speed.
hitachi_replication_number = 0	(Integer(min=0, max=255)) Instance number for REST API
hi-tachi_replication_status_check_ = 600	(Integer) Interval at which remote replication pair status is checked. This parameter is applied if the status has not changed to the expected status after the time indicated by this parameter has elapsed.
hi-tachi_replication_status_check_ = 5	(Integer) Initial interval at which remote replication pair status is checked
hi-tachi_replication_status_check_ = 86400	(Integer) Maximum wait time before the remote replication pair status changes to the expected status
hi-tachi_rest_another_ldev_mappe = 600	(Integer) Retry time in seconds when new LUN allocation request fails.
hitachi_rest_connect_timeout = 30	(Integer) Maximum wait time in seconds for connecting to REST API session.
hitachi_rest_disable_io_wait = True	(Boolean) This option will allow detaching volume immediately. If set False, storage may take few minutes to detach volume after I/O.
hi-tachi_rest_get_api_response_tir = 1800	(Integer) Maximum wait time in seconds for a response against sync methods, for example GET
hi-tachi_rest_job_api_response_tir = 1800	(Integer) Maximum wait time in seconds for a response against async methods from REST API, for example PUT and DELETE.
hi-tachi_rest_keep_session_loop_i = 180	(Integer) Loop interval in seconds for keeping REST API session.
hitachi_rest_pair_target_ports = []	(List of String) Target port names for pair of the host group or iSCSI target
hi-tachi_rest_server_busy_timeout = 7200	(Integer) Maximum wait time in seconds when REST API returns busy.
hitachi_rest_tcp_keepalive = True	(Boolean) Enables or disables use of REST API tcp keepalive
hitachi_rest_tcp_keepcnt = 4	(Integer) Maximum number of transmissions for TCP keepalive packet.
hitachi_rest_tcp_keepidle = 60	(Integer) Wait time in seconds for sending a first TCP keepalive packet.
hitachi_rest_tcp_keepintvl = 15	(Integer) Interval of transmissions in seconds for TCP keepalive packet.
hitachi_rest_timeout = 30	(Integer) Maximum wait time in seconds for each REST API request.
hitachi_restore_timeout = 86400	(Integer) Maximum wait time in seconds for the restore operation to complete.

continues on next page

Table 147 – continued from previous page

Name = Default Value	(Type) Description
hitachi_set_mirror_reserve_attrib = True	(Boolean) Whether or not to set the mirror reserve attribute
hitachi_snap_pool = None	(String) Pool number or pool name of the snapshot pool.
hitachi_state_transition_timeout = 900	(Integer) Maximum wait time in seconds for a volume transition to complete.
hitachi_storage_id = None	(String) Product number of the storage system.
hitachi_target_ports = []	(List of String) IDs of the storage ports used to attach volumes to the controller node. To specify multiple ports, connect them by commas (e.g. CL1-A,CL2-A).
hitachi_zoning_request = False	(Boolean) If True, the driver will configure FC zoning between the server and the storage system provided that FC zoning manager is enabled.
max_over_subscription_ratio = 20.0	(String(regex=^(auto|\d*\.\d+|\d+)$)) Representation of the over subscription ratio when thin provisioning is enabled. Default ratio is 20.0, meaning provisioned capacity can be 20 times of the total physical capacity. If the ratio is 10.5, it means provisioned capacity can be 10.5 times of the total physical capacity. A ratio of 1.0 means provisioned capacity cannot exceed the total physical capacity. If ratio is auto, Cinder will automatically calculate the ratio based on the provisioned capacity and the used space. If not set to auto, the ratio has to be a minimum of 1.0.
reserved_percentage = 0	(Integer(min=0, max=100)) The percentage of backend capacity is reserved
san_api_port = None	(Port(min=0, max=65535)) Port to use to access the SAN API
san_ip =	(String) IP address of SAN controller
san_login = admin	(String) Username for SAN controller
san_password =	(String) Password for SAN controller
use_chap_auth = False	(Boolean) Option to enable/disable CHAP authentication for targets.
use_multipath_for_image_xfer = False	(Boolean) Do we attach/detach volumes in cinder using multipath for volume to image and image to volume transfers? This parameter needs to be configured for each backend section or in [backend_defaults] section as a common configuration for all backends.
volume_backend_name = None	(String) The backend name for a given driver implementation
volume_driver = cinder.volume.drivers.lvm.LVMVc	(String) Driver to use for volume creation

- Description: Fibre channel class for Hitachi HBSD Driver.

Version history:

```
1.0.0 - Initial driver.
1.1.0 - Add manage_existing/manage_existing_get_size/unmanage methods
2.0.0 - Major redesign of the driver. This version requires the REST
        API for communication with the storage backend.
2.1.0 - Add Cinder generic volume groups.
2.2.0 - Add maintenance parameters.
```

(continues on next page)

(continued from previous page)

```
2.2.1 - Make the parameters name variable for supporting OEM storages.
2.2.2 - Add Target Port Assignment.
2.2.3 - Add port scheduler.
2.3.0 - Support multi pool.
2.3.1 - Update retype and support storage assisted migration.
2.3.2 - Add specifies format of the names HostGroups/iSCSI Targets.
2.3.3 - Add GAD volume support.
2.3.4 - Support data deduplication and compression.
2.3.5 - Fix key error when backend is down.
2.4.0 - Add QoS support.
```

HBSDISCSIDriver

- Version: 2.4.0
- volume_driver=cinder.volume.drivers.hitachi.hbsd_iscsi.HBSDISCSIDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/Hitachi_VSP_CI
- Driver Configuration Options:

Table 148: **Driver configuration options**

Name = Default Value	(Type) Description
chap_password =	(String) Password for specified CHAP account name.
chap_username =	(String) CHAP user name.
driver_ssl_cert_path = None	(String) Can be used to specify a non default path to a CA_BUNDLE file or directory with certificates of trusted CAs, which will be used to validate the backend
driver_ssl_cert_verify = False	(Boolean) If set to True the http client will validate the SSL certificate of the backend endpoint.
en-force_multipath_for_image_xfe = False	(Boolean) If this is set to True, attachment of volumes for image transfer will be aborted when multipathd is not running. Otherwise, it will fallback to single path. This parameter needs to be configured for each backend section or in [backend_defaults] section as a common configuration for all backends.
hi-tachi_async_copy_check_interv = 10	(Integer(min=1, max=600)) Interval in seconds to check asynchronous copying status during a copy pair deletion or data restoration.
hitachi_compute_target_ports = []	(List of String) IDs of the storage ports used to attach volumes to compute nodes. To specify multiple ports, connect them by commas (e.g. CL1-A,CL2-A).
hitachi_copy_check_interval = 3	(Integer(min=1, max=600)) Interval in seconds to check copying status during a volume copy.
hitachi_copy_speed = 3	(Integer(min=1, max=15)) Copy speed of storage system. 1 or 2 indicates low speed, 3 indicates middle speed, and a value between 4 and 15 indicates high speed.
hitachi_discard_zero_page = True	(Boolean) Enable or disable zero page reclamation in a DP-VOL.

continues on next page

Table 148 – continued from previous page

Name = Default Value	(Type) Description
hitachi_exec_retry_interval = 5	(Integer) Retry interval in seconds for REST API execution.
hitachi_extend_timeout = 600	(Integer) Maximum wait time in seconds for a volume extention to complete.
hitachi_group_create = False	(Boolean) If True, the driver will create host groups or iSCSI targets on storage ports as needed.
hitachi_group_delete = False	(Boolean) If True, the driver will delete host groups or iSCSI targets on storage ports as needed.
hitachi_group_name_format = None	(String) Format of host groups, iSCSI targets, and server objects.
hitachi_host_mode_options = []	(List of Integer) Host mode option for host group or iSCSI target.
hitachi_ldev_range = None	(String) Range of the LDEV numbers in the format of xxxx-yyyy that can be used by the driver. Values can be in decimal format (e.g. 1000) or in colon-separated hexadecimal format (e.g. 00:03:E8).
hitachi_lock_timeout = 7200	(Integer) Maximum wait time in seconds for storage to be logined or unlocked.
hitachi_lun_retry_interval = 1	(Integer) Retry interval in seconds for REST API adding a LUN mapping to the server.
hitachi_lun_timeout = 50	(Integer) Maximum wait time in seconds for adding a LUN mapping to the server.
hi-tachi_mirror_auth_password = None	(String) iSCSI authentication password
hitachi_mirror_auth_user = None	(String) iSCSI authentication username
hi-tachi_mirror_compute_target_p = []	(List of String) Target port names of compute node for host group or iSCSI target
hitachi_mirror_ldev_range = None	(String) Logical device range of secondary storage system
hi-tachi_mirror_pair_target_numb = 0	(Integer(min=0, max=99)) Pair target name of the host group or iSCSI target
hitachi_mirror_pool = None	(String) Pool of secondary storage system
hitachi_mirror_rest_api_ip = None	(String) IP address of REST API server
hitachi_mirror_rest_api_port = 443	(Port(min=0, max=65535)) Port number of REST API server
hi-tachi_mirror_rest_pair_target_p = []	(List of String) Target port names for pair of the host group or iSCSI target
hitachi_mirror_rest_password = None	(String) Password of secondary storage system for REST API
hitachi_mirror_rest_user = None	(String) Username of secondary storage system for REST API
hitachi_mirror_snap_pool = None	(String) Thin pool of secondary storage system

continues on next page

Table 148 – continued from previous page

Name = Default Value	(Type) Description
hitachi_mirror_ssl_cert_path = None	(String) Can be used to specify a non default path to a CA_BUNDLE file or directory with certificates of trusted CAs, which will be used to validate the backend
hi-tachi_mirror_ssl_cert_verify = False	(Boolean) If set to True the http client will validate the SSL certificate of the backend endpoint.
hitachi_mirror_storage_id = None	(String) ID of secondary storage system
hitachi_mirror_target_ports = []	(List of String) Target port names for host group or iSCSI target
hi-tachi_mirror_use_chap_auth = False	(Boolean) Whether or not to use iSCSI authentication
hitachi_pair_target_number = 0	(Integer(min=0, max=99)) Pair target name of the host group or iSCSI target
hitachi_path_group_id = 0	(Integer(min=0, max=255)) Path group ID assigned to the remote connection for remote replication
hitachi_pools = []	(List of String) Pool number[s] or pool name[s] of the DP pool.
hitachi_quorum_disk_id = None	(Integer(min=0, max=31)) ID of the Quorum disk used for global-active device
hi-tachi_replication_copy_speed = 3	(Integer(min=1, max=15)) Remote copy speed of storage system. 1 or 2 indicates low speed, 3 indicates middle speed, and a value between 4 and 15 indicates high speed.
hitachi_replication_number = 0	(Integer(min=0, max=255)) Instance number for REST API
hi-tachi_replication_status_check_ = 600	(Integer) Interval at which remote replication pair status is checked. This parameter is applied if the status has not changed to the expected status after the time indicated by this parameter has elapsed.
hi-tachi_replication_status_check_ = 5	(Integer) Initial interval at which remote replication pair status is checked
hi-tachi_replication_status_check_ = 86400	(Integer) Maximum wait time before the remote replication pair status changes to the expected status
hi-tachi_rest_another_ldev_mappe = 600	(Integer) Retry time in seconds when new LUN allocation request fails.
hitachi_rest_connect_timeout = 30	(Integer) Maximum wait time in seconds for connecting to REST API session.
hitachi_rest_disable_io_wait = True	(Boolean) This option will allow detaching volume immediately. If set False, storage may take few minutes to detach volume after I/O.
hi-tachi_rest_get_api_response_tir = 1800	(Integer) Maximum wait time in seconds for a response against sync methods, for example GET
hi-tachi_rest_job_api_response_tir = 1800	(Integer) Maximum wait time in seconds for a response against async methods from REST API, for example PUT and DELETE.

continues on next page

Table 148 – continued from previous page

Name = Default Value	(Type) Description
hi-tachi_rest_keep_session_loop_i = 180	(Integer) Loop interval in seconds for keeping REST API session.
hitachi_rest_pair_target_ports = []	(List of String) Target port names for pair of the host group or iSCSI target
hi-tachi_rest_server_busy_timeout = 7200	(Integer) Maximum wait time in seconds when REST API returns busy.
hitachi_rest_tcp_keepalive = True	(Boolean) Enables or disables use of REST API tcp keepalive
hitachi_rest_tcp_keepcnt = 4	(Integer) Maximum number of transmissions for TCP keepalive packet.
hitachi_rest_tcp_keepidle = 60	(Integer) Wait time in seconds for sending a first TCP keepalive packet.
hitachi_rest_tcp_keepintvl = 15	(Integer) Interval of transmissions in seconds for TCP keepalive packet.
hitachi_rest_timeout = 30	(Integer) Maximum wait time in seconds for each REST API request.
hitachi_restore_timeout = 86400	(Integer) Maximum wait time in seconds for the restore operation to complete.
hi-tachi_set_mirror_reserve_attrib = True	(Boolean) Whether or not to set the mirror reserve attribute
hitachi_snap_pool = None	(String) Pool number or pool name of the snapshot pool.
hi-tachi_state_transition_timeout = 900	(Integer) Maximum wait time in seconds for a volume transition to complete.
hitachi_storage_id = None	(String) Product number of the storage system.
hitachi_target_ports = []	(List of String) IDs of the storage ports used to attach volumes to the controller node. To specify multiple ports, connect them by commas (e.g. CL1-A,CL2-A).
max_over_subscription_ratio = 20.0	(String(regex=^(auto|\d*\.\d+|\d+)$)) Representation of the over subscription ratio when thin provisioning is enabled. Default ratio is 20.0, meaning provisioned capacity can be 20 times of the total physical capacity. If the ratio is 10.5, it means provisioned capacity can be 10.5 times of the total physical capacity. A ratio of 1.0 means provisioned capacity cannot exceed the total physical capacity. If ratio is auto, Cinder will automatically calculate the ratio based on the provisioned capacity and the used space. If not set to auto, the ratio has to be a minimum of 1.0.
reserved_percentage = 0	(Integer(min=0, max=100)) The percentage of backend capacity is reserved
san_api_port = None	(Port(min=0, max=65535)) Port to use to access the SAN API
san_ip =	(String) IP address of SAN controller
san_login = admin	(String) Username for SAN controller
san_password =	(String) Password for SAN controller
use_chap_auth = False	(Boolean) Option to enable/disable CHAP authentication for targets.

continues on next page

Table 148 – continued from previous page

Name = Default Value	(Type) Description
use_multipath_for_image_xfer = False	(Boolean) Do we attach/detach volumes in cinder using multipath for volume to image and image to volume transfers? This parameter needs to be configured for each backend section or in [backend_defaults] section as a common configuration for all backends.
volume_backend_name = None	(String) The backend name for a given driver implementation
volume_driver = cinder.volume.drivers.lvm.LVMVc	(String) Driver to use for volume creation

- Description: iSCSI class for Hitachi HBSD Driver.

Version history:

```
1.0.0 - Initial driver.
1.1.0 - Add manage_existing/manage_existing_get_size/unmanage methods
2.0.0 - Major redesign of the driver. This version requires the REST
        API for communication with the storage backend.
2.1.0 - Add Cinder generic volume groups.
2.2.0 - Add maintenance parameters.
2.2.1 - Make the parameters name variable for supporting OEM storages.
2.2.2 - Add Target Port Assignment.
2.2.3 - Add port scheduler.
2.3.0 - Support multi pool.
2.3.1 - Update retype and support storage assisted migration.
2.3.2 - Add specifies format of the names HostGroups/iSCSI Targets.
2.3.3 - Add GAD volume support.
2.3.4 - Support data deduplication and compression.
2.3.5 - Fix key error when backend is down.
2.4.0 - Add QoS support.
```

HPE3PARFCDriver

- Version: 4.0.8
- volume_driver=cinder.volume.drivers.hpe.hpe_3par_fc.HPE3PARFCDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/HPE_Storage_CI
- Driver Configuration Options:

Table 149: **Driver configuration options**

Name = Default Value	(Type) Description
hpe3par_api_url =	(String) WSAPI Server URL. This setting applies to: 3PAR, Primera, Alletra 9k and Alletra MP Example 1: for 3PAR, URL is: https://<3par ip>:8080/api/v1 Example 2: for Primera/Alletra 9k/Alletra MP, URL is: https://<primera ip>:443/api/v1
hpe3par_cpg = [OpenStack]	(List of String) List of the 3PAR/Primera/Alletra 9k/Alletra MP CPG(s) to use for volume creation
hpe3par_cpg_snap =	(String) The 3PAR/Primera/Alletra 9k/Alletra MP CPG to use for snapshots of volumes. If empty the userCPG will be used
hpe3par_debug = False	(Boolean) Enable HTTP debugging to 3PAR/Primera/Alletra 9k/Alletra MP
hpe3par_iscsi_chap_enabled = False	(Boolean) Enable CHAP authentication for iSCSI connections.
hpe3par_iscsi_ips = []	(List of String) List of target iSCSI addresses to use.
hpe3par_password =	(String) 3PAR/Primera/Alletra 9k/Alletra MP password for the user specified in hpe3par_username
hpe3par_snapshot_expiration =	(String) The time in hours when a snapshot expires and is deleted. This must be larger than expiration
hpe3par_snapshot_retention =	(String) The time in hours to retain a snapshot. You cant delete it before this expires.
hpe3par_target_nsp =	(String) The nsp of 3PAR/Primera/Alletra 9k/Alletra MP backend to be used when: (1) multipath is not enabled in cinder.conf. (2) Fiber Channel Zone Manager is not used. (3) the backend is prezoned with this specific nsp only. For example if nsp is 2 1 2, the format of the options value is 2:1:2
hpe3par_username =	(String) 3PAR/Primera/Alletra 9k/Alletra MP username with the edit role
max_over_subscription_ratio = 20.0	(String(regex=^(auto|\d*\.\d+|\d+)$)) Representation of the over subscription ratio when thin provisioning is enabled. Default ratio is 20.0, meaning provisioned capacity can be 20 times of the total physical capacity. If the ratio is 10.5, it means provisioned capacity can be 10.5 times of the total physical capacity. A ratio of 1.0 means provisioned capacity cannot exceed the total physical capacity. If ratio is auto, Cinder will automatically calculate the ratio based on the provisioned capacity and the used space. If not set to auto, the ratio has to be a minimum of 1.0.
replication_device = None	(Dict of String) Multi opt of dictionaries to represent a replication target device. This option may be specified multiple times in a single config section to specify multiple replication target devices. Each entry takes the standard dict config form: replication_device = target_device_id:<required>,key1:value1,key2:value2
reserved_percentage = 0	(Integer(min=0, max=100)) The percentage of backend capacity is reserved
san_ip =	(String) IP address of SAN controller
san_login = admin	(String) Username for SAN controller
san_password =	(String) Password for SAN controller
san_private_key =	(String) Filename of private key to use for SSH authentication
san_ssh_port = 22	(Port(min=0, max=65535)) SSH port to use with SAN
ssh_conn_timeout = 30	(Integer) SSH connection timeout in seconds
target_ip_address = $my_ip	(String) The IP address that the iSCSI/NVMEoF daemon is listening on
target_port = 3260	(Port(min=0, max=65535)) The port that the iSCSI/NVMEoF daemon is listening on
unique_fqdn_network = True	(Boolean) Whether or not our private network has unique FQDN on

- Description: OpenStack Fibre Channel driver to enable 3PAR storage array.

Version history:

```
1.0   - Initial driver
1.1   - QoS, extend volume, multiple iscsi ports, remove domain,
        session changes, faster clone, requires 3.1.2 MU2 firmware,
        copy volume <--> Image.
1.2.0 - Updated the use of the hp3parclient to 2.0.0 and refactored
        the drivers to use the new APIs.
1.2.1 - Synchronized extend_volume method.
1.2.2 - Added try/finally around client login/logout.
1.2.3 - Added ability to add WWNs to host.
1.2.4 - Added metadata during attach/detach bug #1258033.
1.3.0 - Removed all SSH code.  We rely on the hp3parclient now.
2.0.0 - Update hp3parclient API uses 3.0.x
2.0.2 - Add back-end assisted volume migrate
2.0.3 - Added initiator-target map for FC Zone Manager
2.0.4 - Added support for managing/unmanaging of volumes
2.0.5 - Only remove FC Zone on last volume detach
2.0.6 - Added support for volume retype
2.0.7 - Only one FC port is used when a single FC path
        is present.  bug #1360001
2.0.8 - Fixing missing login/logout around attach/detach bug #1367429
2.0.9 - Add support for pools with model update
2.0.10 - Migrate without losing type settings bug #1356608
2.0.11 - Removing locks bug #1381190
2.0.12 - Fix queryHost call to specify wwns bug #1398206
2.0.13 - Fix missing host name during attach bug #1398206
2.0.14 - Removed usage of host name cache #1398914
2.0.15 - Added support for updated detach_volume attachment.
2.0.16 - Added encrypted property to initialize_connection #1439917
2.0.17 - Improved VLUN creation and deletion logic. #1469816
2.0.18 - Changed initialize_connection to use getHostVLUNs. #1475064
2.0.19 - Adds consistency group support
2.0.20 - Update driver to use ABC metaclasses
2.0.21 - Added update_migrated_volume. bug # 1492023
3.0.0 - Rebranded HP to HPE.
3.0.1 - Remove db access for consistency groups
3.0.2 - Adds v2 managed replication support
3.0.3 - Adds v2 unmanaged replication support
3.0.4 - Adding manage/unmanage snapshot support
3.0.5 - Optimize array ID retrieval
3.0.6 - Update replication to version 2.1
3.0.7 - Remove metadata that tracks the instance ID. bug #1572665
3.0.8 - NSP feature, creating FC Vlun as match set instead of
        host sees. bug #1577993
3.0.9 - Handling HTTP conflict 409, host WWN/iSCSI name already used
        by another host, while creating 3PAR FC Host. bug #1597454
3.0.10 - Added Entry point tracing
3.0.11 - Handle manage and unmanage hosts present. bug #1648067
```

(continues on next page)

(continued from previous page)

```
3.0.12 - Adds consistency group capability in generic volume groups.
4.0.0 - Adds base class.
4.0.1 - Added check to remove FC zones. bug #1730720
4.0.2 - Create one vlun in single path configuration. bug #1727176
4.0.3 - Create FC vlun as host sees. bug #1734505
4.0.4 - Handle force detach case. bug #1686745
4.0.5 - Set proper backend on subsequent operation, after group
        failover. bug #1773069
4.0.6 - Set NSP for single path attachments. Bug #1809249
4.0.7 - Added Peer Persistence feature
4.0.8 - For PP, return LUN ids from both arrays. Bug #2044255
```

HPE3PARISCSIDriver

- Version: 4.0.9
- volume_driver=cinder.volume.drivers.hpe.hpe_3par_iscsi.HPE3PARISCSIDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/HPE_Storage_CI
- Driver Configuration Options:

Table 150: **Driver configuration options**

Name = Default Value	(Type) Description
hpe3par_api_url =	(String) WSAPI Server URL. This setting applies to: 3PAR, Primera, Alletra 9k and Alletra MP Example 1: for 3PAR, URL is: https://<3par ip>:8080/api/v1 Example 2: for Primera/Alletra 9k/Alletra MP, URL is: https://<primera ip>:443/api/v1
hpe3par_cpg = [OpenStack]	(List of String) List of the 3PAR/Primera/Alletra 9k/Alletra MP CPG(s) to use for volume creation
hpe3par_cpg_snap =	(String) The 3PAR/Primera/Alletra 9k/Alletra MP CPG to use for snapshots of volumes. If empty the userCPG will be used
hpe3par_debug = False	(Boolean) Enable HTTP debugging to 3PAR/Primera/Alletra 9k/Alletra MP
hpe3par_iscsi_chap_enabled = False	(Boolean) Enable CHAP authentication for iSCSI connections.
hpe3par_iscsi_ips = []	(List of String) List of target iSCSI addresses to use.
hpe3par_password =	(String) 3PAR/Primera/Alletra 9k/Alletra MP password for the user specified in hpe3par_username
hpe3par_snapshot_expiration =	(String) The time in hours when a snapshot expires and is deleted. This must be larger than expiration
hpe3par_snapshot_retention =	(String) The time in hours to retain a snapshot. You cant delete it before this expires.
hpe3par_target_nsp =	(String) The nsp of 3PAR/Primera/Alletra 9k/Alletra MP backend to be used when: (1) multipath is not enabled in cinder.conf. (2) Fiber Channel Zone Manager is not used. (3) the backend is prezoned with this specific nsp only. For example if nsp is 2 1 2, the format of the options value is 2:1:2
hpe3par_username =	(String) 3PAR/Primera/Alletra 9k/Alletra MP username with the edit role
max_over_subscription_ratio = 20.0	(String(regex=^(auto|\d*\.\d+|\d+)$)) Representation of the over subscription ratio when thin provisioning is enabled. Default ratio is 20.0, meaning provisioned capacity can be 20 times of the total physical capacity. If the ratio is 10.5, it means provisioned capacity can be 10.5 times of the total physical capacity. A ratio of 1.0 means provisioned capacity cannot exceed the total physical capacity. If ratio is auto, Cinder will automatically calculate the ratio based on the provisioned capacity and the used space. If not set to auto, the ratio has to be a minimum of 1.0.
replication_device = None	(Dict of String) Multi opt of dictionaries to represent a replication target device. This option may be specified multiple times in a single config section to specify multiple replication target devices. Each entry takes the standard dict config form: replication_device = target_device_id:<required>,key1:value1,key2:value2
reserved_percentage = 0	(Integer(min=0, max=100)) The percentage of backend capacity is reserved
san_ip =	(String) IP address of SAN controller
san_login = admin	(String) Username for SAN controller
san_password =	(String) Password for SAN controller
san_private_key =	(String) Filename of private key to use for SSH authentication
san_ssh_port = 22	(Port(min=0, max=65535)) SSH port to use with SAN
ssh_conn_timeout = 30	(Integer) SSH connection timeout in seconds
target_ip_address = $my_ip	(String) The IP address that the iSCSI/NVMEoF daemon is listening on
target_port = 3260	(Port(min=0, max=65535)) The port that the iSCSI/NVMEoF daemon is listening on
unique_fqdn_network = True	(Boolean) Whether or not our private network has unique FQDN on

- Description: OpenStack iSCSI driver to enable 3PAR storage array.

Version history:

```
1.0   - Initial driver
1.1   - QoS, extend volume, multiple iscsi ports, remove domain,
        session changes, faster clone, requires 3.1.2 MU2 firmware.
1.2.0 - Updated the use of the hp3parclient to 2.0.0 and refactored
        the drivers to use the new APIs.
1.2.1 - Synchronized extend_volume method.
1.2.2 - Added try/finally around client login/logout.
1.2.3 - log exceptions before raising
1.2.4 - Fixed iSCSI active path bug #1224594
1.2.5 - Added metadata during attach/detach bug #1258033
1.2.6 - Use least-used iscsi n:s:p for iscsi volume attach bug #1269515
        This update now requires 3.1.2 MU3 firmware
1.3.0 - Removed all SSH code.  We rely on the hp3parclient now.
2.0.0 - Update hp3parclient API uses 3.0.x
2.0.2 - Add back-end assisted volume migrate
2.0.3 - Added support for managing/unmanaging of volumes
2.0.4 - Added support for volume retype
2.0.5 - Added CHAP support, requires 3.1.3 MU1 firmware
        and hp3parclient 3.1.0.
2.0.6 - Fixing missing login/logout around attach/detach bug #1367429
2.0.7 - Add support for pools with model update
2.0.8 - Migrate without losing type settings bug #1356608
2.0.9 - Removing locks bug #1381190
2.0.10 - Add call to queryHost instead SSH based findHost #1398206
2.0.11 - Added missing host name during attach fix #1398206
2.0.12 - Removed usage of host name cache #1398914
2.0.13 - Update LOG usage to fix translations.  bug #1384312
2.0.14 - Do not allow a different iSCSI IP (hp3par_iscsi_ips) to be
         used during live-migration.  bug #1423958
2.0.15 - Added support for updated detach_volume attachment.
2.0.16 - Added encrypted property to initialize_connection #1439917
2.0.17 - Python 3 fixes
2.0.18 - Improved VLUN creation and deletion logic. #1469816
2.0.19 - Changed initialize_connection to use getHostVLUNs. #1475064
2.0.20 - Adding changes to support 3PAR iSCSI multipath.
2.0.21 - Adds consistency group support
2.0.22 - Update driver to use ABC metaclasses
2.0.23 - Added update_migrated_volume. bug # 1492023
3.0.0 - Rebranded HP to HPE.
3.0.1 - Python 3 support
3.0.2 - Remove db access for consistency groups
3.0.3 - Fix multipath dictionary key error. bug #1522062
3.0.4 - Adds v2 managed replication support
3.0.5 - Adds v2 unmanaged replication support
3.0.6 - Adding manage/unmanage snapshot support
3.0.7 - Optimize array ID retrieval
3.0.8 - Update replication to version 2.1
```

(continues on next page)

(continued from previous page)

```
3.0.9 - Use same LUN ID for each VLUN path #1551994
3.0.10 - Remove metadata that tracks the instance ID. bug #1572665
3.0.11 - _create_3par_iscsi_host() now accepts iscsi_iqn as list only.
         Bug #1590180
3.0.12 - Added entry point tracing
3.0.13 - Handling HTTP conflict 409, host WWN/iSCSI name already used
        by another host, while creating 3PAR iSCSI Host. bug #1642945
3.0.14 - Handle manage and unmanage hosts present. bug #1648067
3.0.15 - Adds consistency group capability in generic volume groups.
3.0.16 - Get host from os-brick connector. bug #1690244
4.0.0 - Adds base class.
4.0.1 - Update CHAP on host record when volume is migrated
        to new compute host. bug # 1737181
4.0.2 - Handle force detach case. bug #1686745
4.0.3 - Set proper backend on subsequent operation, after group
        failover. bug #1773069
4.0.4 - Added Peer Persistence feature
4.0.5 - Added Primera array check. bug #1849525
4.0.6 - Allow iSCSI support for Primera 4.2 onwards
4.0.7 - Use vlan iscsi ips. Bug #2015034
4.0.8 - Add ipv6 support. Bug #2045411
4.0.9 - getWsApiVersion now requires login
```

HPEXPFCDriver

- Version: 1.0.0
- volume_driver=cinder.volume.drivers.hpe.xp.hpe_xp_fc.HPEXPFCDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/HPE_XP_Storage_CI
- Driver Configuration Options:

Table 151: **Driver configuration options**

Name = Default Value	(Type) Description
chap_password =	(String) Password for specified CHAP account name.
chap_username =	(String) CHAP user name.
driver_ssl_cert_path = None	(String) Can be used to specify a non default path to a CA_BUNDLE file or directory with certificates of trusted CAs, which will be used to validate the backend
driver_ssl_cert_verify = False	(Boolean) If set to True the http client will validate the SSL certificate of the backend endpoint.
enforce_multipath_for_image_xfe = False	(Boolean) If this is set to True, attachment of volumes for image transfer will be aborted when multipathd is not running. Otherwise, it will fallback to single path. This parameter needs to be configured for each backend section or in [backend_defaults] section as a common configuration for all backends.

continues on next page

Table 151 – continued from previous page

Name = Default Value	(Type) Description
hpexp_async_copy_check_interva = 10	(Integer(min=1, max=600)) Interval in seconds to check copy asynchronously
hpexp_compute_target_ports = []	(List of String) IDs of the storage ports used to attach volumes to compute nodes. To specify multiple ports, connect them by commas (e.g. CL1-A,CL2-A).
hpexp_copy_check_interval = 3	(Integer(min=1, max=600)) Interval in seconds to check copy
hpexp_copy_speed = 3	(Integer(min=1, max=15)) Copy speed of storage system. 1 or 2 indicates low speed, 3 indicates middle speed, and a value between 4 and 15 indicates high speed.
hpexp_discard_zero_page = True	(Boolean) Enable or disable zero page reclamation in a THP V-VOL.
hpexp_exec_retry_interval = 5	(Integer) Retry interval in seconds for REST API execution.
hpexp_extend_timeout = 600	(Integer) Maximum wait time in seconds for a volume extention to complete.
hpexp_group_create = False	(Boolean) If True, the driver will create host groups or iSCSI targets on storage ports as needed.
hpexp_group_delete = False	(Boolean) If True, the driver will delete host groups or iSCSI targets on storage ports as needed.
hpexp_host_mode_options = []	(List of String) Host mode option for host group or iSCSI target.
hpexp_ldev_range = None	(String) Range of the LDEV numbers in the format of xxxx-yyyy that can be used by the driver. Values can be in decimal format (e.g. 1000) or in colon-separated hexadecimal format (e.g. 00:03:E8).
hpexp_lock_timeout = 7200	(Integer) Maximum wait time in seconds for storage to be unlocked.
hpexp_lun_retry_interval = 1	(Integer) Retry interval in seconds for REST API adding a LUN.
hpexp_lun_timeout = 50	(Integer) Maximum wait time in seconds for adding a LUN to complete.
hpexp_pools = []	(List of String) Pool number[s] or pool name[s] of the THP pool.
hpexp_rest_another_ldev_mapped = 600	(Integer) Retry time in seconds when new LUN allocation request fails.
hpexp_rest_connect_timeout = 30	(Integer) Maximum wait time in seconds for REST API connection to complete.
hpexp_rest_disable_io_wait = True	(Boolean) It may take some time to detach volume after I/O. This option will allow detaching volume to complete immediately.
hpexp_rest_get_api_response_tim = 1800	(Integer) Maximum wait time in seconds for a response against GET method of REST API.
hpexp_rest_job_api_response_tim = 1800	(Integer) Maximum wait time in seconds for a response from REST API.
hpexp_rest_keep_session_loop_in = 180	(Integer) Loop interval in seconds for keeping REST API session.

continues on next page

Table 151 – continued from previous page

Name = Default Value	(Type) Description
hpexp_rest_server_busy_timeout = 7200	(Integer) Maximum wait time in seconds when REST API returns busy.
hpexp_rest_tcp_keepalive = True	(Boolean) Enables or disables use of REST API tcp keepalive
hpexp_rest_tcp_keepcnt = 4	(Integer) Maximum number of transmissions for TCP keepalive packet.
hpexp_rest_tcp_keepidle = 60	(Integer) Wait time in seconds for sending a first TCP keepalive packet.
hpexp_rest_tcp_keepintvl = 15	(Integer) Interval of transmissions in seconds for TCP keepalive packet.
hpexp_rest_timeout = 30	(Integer) Maximum wait time in seconds for REST API execution to complete.
hpexp_restore_timeout = 86400	(Integer) Maximum wait time in seconds for the restore operation to complete.
hpexp_snap_pool = None	(String) Pool number or pool name of the snapshot pool.
hpexp_state_transition_timeout = 900	(Integer) Maximum wait time in seconds for a volume transition to complete.
hpexp_storage_id = None	(String) Product number of the storage system.
hpexp_target_ports = []	(List of String) IDs of the storage ports used to attach volumes to the controller node. To specify multiple ports, connect them by commas (e.g. CL1-A,CL2-A).
hpexp_zoning_request = False	(Boolean) If True, the driver will configure FC zoning between the server and the storage system provided that FC zoning manager is enabled.
max_over_subscription_ratio = 20.0	(String(regex=^(auto|\d*\.\d+|\d+)$)) Representation of the over subscription ratio when thin provisioning is enabled. Default ratio is 20.0, meaning provisioned capacity can be 20 times of the total physical capacity. If the ratio is 10.5, it means provisioned capacity can be 10.5 times of the total physical capacity. A ratio of 1.0 means provisioned capacity cannot exceed the total physical capacity. If ratio is auto, Cinder will automatically calculate the ratio based on the provisioned capacity and the used space. If not set to auto, the ratio has to be a minimum of 1.0.
reserved_percentage = 0	(Integer(min=0, max=100)) The percentage of backend capacity is reserved
san_api_port = None	(Port(min=0, max=65535)) Port to use to access the SAN API
san_ip =	(String) IP address of SAN controller
san_login = admin	(String) Username for SAN controller
san_password =	(String) Password for SAN controller
use_chap_auth = False	(Boolean) Option to enable/disable CHAP authentication for targets.
use_multipath_for_image_xfer = False	(Boolean) Do we attach/detach volumes in cinder using multipath for volume to image and image to volume transfers? This parameter needs to be configured for each backend section or in [backend_defaults] section as a common configuration for all backends.
volume_backend_name = None	(String) The backend name for a given driver implementation

continues on next page

Table 151 – continued from previous page

Name = Default Value	(Type) Description
volume_driver = cinder.volume.drivers.lvm.LVMVc	(String) Driver to use for volume creation

- Description: Fibre channel class for Hewlett Packard Enterprise Driver.

Version history:

```
1.0.0 - Initial driver.
```

HPEXPISCSIDriver

- Version: 1.0.0
- volume_driver=cinder.volume.drivers.hpe.xp.hpe_xp_iscsi.HPEXPISCSIDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/HPE_XP_Storage_CI
- Driver Configuration Options:

Table 152: **Driver configuration options**

Name = Default Value	(Type) Description
chap_password =	(String) Password for specified CHAP account name.
chap_username =	(String) CHAP user name.
driver_ssl_cert_path = None	(String) Can be used to specify a non default path to a CA_BUNDLE file or directory with certificates of trusted CAs, which will be used to validate the backend
driver_ssl_cert_verify = False	(Boolean) If set to True the http client will validate the SSL certificate of the backend endpoint.
enforce_multipath_for_image_xfe = False	(Boolean) If this is set to True, attachment of volumes for image transfer will be aborted when multipathd is not running. Otherwise, it will fallback to single path. This parameter needs to be configured for each backend section or in [backend_defaults] section as a common configuration for all backends.
hpexp_async_copy_check_interva = 10	(Integer(min=1, max=600)) Interval in seconds to check copy asynchronously
hpexp_compute_target_ports = []	(List of String) IDs of the storage ports used to attach volumes to compute nodes. To specify multiple ports, connect them by commas (e.g. CL1-A,CL2-A).
hpexp_copy_check_interval = 3	(Integer(min=1, max=600)) Interval in seconds to check copy
hpexp_copy_speed = 3	(Integer(min=1, max=15)) Copy speed of storage system. 1 or 2 indicates low speed, 3 indicates middle speed, and a value between 4 and 15 indicates high speed.
hpexp_discard_zero_page = True	(Boolean) Enable or disable zero page reclamation in a THP V-VOL.
hpexp_exec_retry_interval = 5	(Integer) Retry interval in seconds for REST API execution.
hpexp_extend_timeout = 600	(Integer) Maximum wait time in seconds for a volume extention to complete.

continues on next page

Table 152 – continued from previous page

Name = Default Value	(Type) Description
hpexp_group_create = False	(Boolean) If True, the driver will create host groups or iSCSI targets on storage ports as needed.
hpexp_group_delete = False	(Boolean) If True, the driver will delete host groups or iSCSI targets on storage ports as needed.
hpexp_host_mode_options = []	(List of String) Host mode option for host group or iSCSI target.
hpexp_ldev_range = None	(String) Range of the LDEV numbers in the format of xxxx-yyyy that can be used by the driver. Values can be in decimal format (e.g. 1000) or in colon-separated hexadecimal format (e.g. 00:03:E8).
hpexp_lock_timeout = 7200	(Integer) Maximum wait time in seconds for storage to be unlocked.
hpexp_lun_retry_interval = 1	(Integer) Retry interval in seconds for REST API adding a LUN.
hpexp_lun_timeout = 50	(Integer) Maximum wait time in seconds for adding a LUN to complete.
hpexp_pools = []	(List of String) Pool number[s] or pool name[s] of the THP pool.
hpexp_rest_another_ldev_mapped = 600	(Integer) Retry time in seconds when new LUN allocation request fails.
hpexp_rest_connect_timeout = 30	(Integer) Maximum wait time in seconds for REST API connection to complete.
hpexp_rest_disable_io_wait = True	(Boolean) It may take some time to detach volume after I/O. This option will allow detaching volume to complete immediately.
hpexp_rest_get_api_response_tim = 1800	(Integer) Maximum wait time in seconds for a response against GET method of REST API.
hpexp_rest_job_api_response_tim = 1800	(Integer) Maximum wait time in seconds for a response from REST API.
hpexp_rest_keep_session_loop_in = 180	(Integer) Loop interval in seconds for keeping REST API session.
hpexp_rest_server_busy_timeout = 7200	(Integer) Maximum wait time in seconds when REST API returns busy.
hpexp_rest_tcp_keepalive = True	(Boolean) Enables or disables use of REST API tcp keepalive
hpexp_rest_tcp_keepcnt = 4	(Integer) Maximum number of transmissions for TCP keepalive packet.
hpexp_rest_tcp_keepidle = 60	(Integer) Wait time in seconds for sending a first TCP keepalive packet.
hpexp_rest_tcp_keepintvl = 15	(Integer) Interval of transmissions in seconds for TCP keepalive packet.
hpexp_rest_timeout = 30	(Integer) Maximum wait time in seconds for REST API execution to complete.
hpexp_restore_timeout = 86400	(Integer) Maximum wait time in seconds for the restore operation to complete.
hpexp_snap_pool = None	(String) Pool number or pool name of the snapshot pool.

continues on next page

Table 152 – continued from previous page

Name = Default Value	(Type) Description
hpexp_state_transition_timeout = 900	(Integer) Maximum wait time in seconds for a volume transition to complete.
hpexp_storage_id = None	(String) Product number of the storage system.
hpexp_target_ports = []	(List of String) IDs of the storage ports used to attach volumes to the controller node. To specify multiple ports, connect them by commas (e.g. CL1-A,CL2-A).
max_over_subscription_ratio = 20.0	(String(regex=^(auto|\d*\.\d+|\d+)$)) Representation of the over subscription ratio when thin provisioning is enabled. Default ratio is 20.0, meaning provisioned capacity can be 20 times of the total physical capacity. If the ratio is 10.5, it means provisioned capacity can be 10.5 times of the total physical capacity. A ratio of 1.0 means provisioned capacity cannot exceed the total physical capacity. If ratio is auto, Cinder will automatically calculate the ratio based on the provisioned capacity and the used space. If not set to auto, the ratio has to be a minimum of 1.0.
reserved_percentage = 0	(Integer(min=0, max=100)) The percentage of backend capacity is reserved
san_api_port = None	(Port(min=0, max=65535)) Port to use to access the SAN API
san_ip =	(String) IP address of SAN controller
san_login = admin	(String) Username for SAN controller
san_password =	(String) Password for SAN controller
use_chap_auth = False	(Boolean) Option to enable/disable CHAP authentication for targets.
use_multipath_for_image_xfer = False	(Boolean) Do we attach/detach volumes in cinder using multipath for volume to image and image to volume transfers? This parameter needs to be configured for each backend section or in [backend_defaults] section as a common configuration for all backends.
volume_backend_name = None	(String) The backend name for a given driver implementation
volume_driver = cinder.volume.drivers.lvm.LVMVc	(String) Driver to use for volume creation

- Description: iSCSI class for Hewlett Packard Enterprise Driver.

Version history:

```
1.0.0 - Initial driver.
```

HPMSAFCDriver

- Version: 2.0
- volume_driver=cinder.volume.drivers.san.hp.hpmsa_fc.HPMSAFCDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/HPMSA_CI
- Driver Configuration Options:

Table 153: **Driver configuration options**

Name = Default Value	(Type) Description
driver_ssl_cert_path = None	(String) Can be used to specify a non default path to a CA_BUNDLE file or directory with certificates of trusted CAs, which will be used to validate the backend
driver_ssl_cert_verify = False	(Boolean) If set to True the http client will validate the SSL certificate of the backend endpoint.
driver_use_ssl = False	(Boolean) Tell driver to use SSL for connection to backend storage if the driver supports it.
hpmsa_api_protocol = https	(String(choices=[http, https])) HPMSA API interface protocol.
hpmsa_pool_name = A	(String) Pool or Vdisk name to use for volume creation.
hpmsa_pool_type = virtual	(String(choices=[linear, virtual])) linear (for Vdisk) or virtual (for Pool).
hpmsa_verify_certificate = False	(Boolean) Whether to verify HPMSA array SSL certificate.
hpmsa_verify_certificate_path = None	(String) HPMSA array SSL certificate path.
san_ip =	(String) IP address of SAN controller
san_login = admin	(String) Username for SAN controller
san_password =	(String) Password for SAN controller

- Description: OpenStack Fibre Channel cinder drivers for HPMSA arrays.

```
Version history:
    1.0    - Inheriting from DotHill cinder drivers.
    1.6    - Add management path redundancy and reduce load placed
             on management controller.
    2.0    - DotHill driver renamed to Seagate (STX)
```

HPMSAISCSIDriver

- Version: 2.0
- volume_driver=cinder.volume.drivers.san.hp.hpmsa_iscsi.HPMSAISCSIDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/HPMSA_CI
- Driver Configuration Options:

Table 154: **Driver configuration options**

Name = Default Value	(Type) Description
driver_ssl_cert_path = None	(String) Can be used to specify a non default path to a CA_BUNDLE file or directory with certificates of trusted CAs, which will be used to validate the backend
driver_ssl_cert_verify = False	(Boolean) If set to True the http client will validate the SSL certificate of the backend endpoint.
driver_use_ssl = False	(Boolean) Tell driver to use SSL for connection to backend storage if the driver supports it.
hpmsa_api_protocol = https	(String(choices=[http, https])) HPMSA API interface protocol.
hpmsa_iscsi_ips = []	(List of String) List of comma-separated target iSCSI IP addresses.
hpmsa_pool_name = A	(String) Pool or Vdisk name to use for volume creation.
hpmsa_pool_type = virtual	(String(choices=[linear, virtual])) linear (for Vdisk) or virtual (for Pool).
hpmsa_verify_certificate = False	(Boolean) Whether to verify HPMSA array SSL certificate.
hpmsa_verify_certificate_path = None	(String) HPMSA array SSL certificate path.
san_ip =	(String) IP address of SAN controller
san_login = admin	(String) Username for SAN controller
san_password =	(String) Password for SAN controller

- Description: OpenStack iSCSI cinder drivers for HPMSA arrays.

```
Version history:
    1.0    - Inheriting from DotHill cinder drivers.
    1.6    - Add management path redundancy and reduce load placed
             on management controller.
    2.0    - DotHill driver renamed to Seagate (STX)
```

HedvigISCSIDriver

- Version: 1.0.0
- volume_driver=cinder.volume.drivers.hedvig.hedvig_cinder.HedvigISCSIDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/Hedvig_CI
- Description: OpenStack Cinder driver to enable Hedvig storage.

```
Version history:

   1.0 - Initial driver
```

HuaweiFCDriver

- Version: 2.0.9
- volume_driver=cinder.volume.drivers.huawei.huawei_driver.HuaweiFCDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/Huawei_volume_CI

- Driver Configuration Options:

Table 155: **Driver configuration options**

Name = Default Value	(Type) Description
cinder_huawei_conf_file = /etc/cinder/cinder_huawei_conf	(String) The configuration file for the Cinder Huawei driver.
hypermetro_devices = None	(String) The remote device hypermetro will use.
metro_domain_name = None	(String) The remote metro device domain name.
metro_san_address = None	(String) The remote metro device request url.
metro_san_password = None	(String) The remote metro device san password.
metro_san_user = None	(String) The remote metro device san user.
metro_storage_pools = None	(String) The remote metro device pool names.

- Description: FC driver for Huawei OceanStor storage arrays.

Version history:

```
1.0.0 - Initial driver
1.1.0 - Provide Huawei OceanStor 18000 storage volume driver
1.1.1 - Code refactor
        Multiple pools support
        SmartX support
        Volume migration support
        Volume retype support
        FC zone enhancement
        Volume hypermetro support
2.0.0 - Rename to HuaweiFCDriver
2.0.1 - Manage/unmanage volume support
2.0.2 - Refactor HuaweiFCDriver
2.0.3 - Manage/unmanage snapshot support
2.0.4 - Balanced FC port selection
2.0.5 - Replication V2 support
2.0.7 - Hypermetro support
        Hypermetro consistency group support
        Consistency group support
        Cgsnapshot support
2.0.8 - Backup snapshot optimal path support
2.0.9 - Support reporting disk type of pool
```

HuaweiISCSIDriver

- Version: 2.0.9
- volume_driver=cinder.volume.drivers.huawei.huawei_driver.HuaweiISCSIDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/Huawei_volume_CI
- Driver Configuration Options:

Table 156: **Driver configuration options**

Name = Default Value	(Type) Description
cinder_huawei_conf_file = /etc/cinder/cinder_huawei_conf	(String) The configuration file for the Cinder Huawei driver.
hypermetro_devices = None	(String) The remote device hypermetro will use.
metro_domain_name = None	(String) The remote metro device domain name.
metro_san_address = None	(String) The remote metro device request url.
metro_san_password = None	(String) The remote metro device san password.
metro_san_user = None	(String) The remote metro device san user.
metro_storage_pools = None	(String) The remote metro device pool names.

- Description: ISCSI driver for Huawei storage arrays.

Version history:

```
1.0.0 - Initial driver
1.1.0 - Provide Huawei OceanStor storage 18000 driver
1.1.1 - Code refactor
        CHAP support
        Multiple pools support
        ISCSI multipath support
        SmartX support
        Volume migration support
        Volume retype support
2.0.0 - Rename to HuaweiISCSIDriver
2.0.1 - Manage/unmanage volume support
2.0.2 - Refactor HuaweiISCSIDriver
2.0.3 - Manage/unmanage snapshot support
2.0.5 - Replication V2 support
2.0.6 - Support iSCSI configuration in Replication
2.0.7 - Hypermetro support
        Hypermetro consistency group support
        Consistency group support
        Cgsnapshot support
2.0.8 - Backup snapshot optimal path support
2.0.9 - Support reporting disk type of pool
```

IBMStorageDriver

- Version: 2.3.0
- volume_driver=cinder.volume.drivers.ibm.ibm_storage.ibm_storage.IBMStorageDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/IBM_STORAGE_CI
- Driver Configuration Options:

Table 157: **Driver configuration options**

Name = Default Value	(Type) Description
chap = disabled	(String(choices=[disabled, enabled])) CHAP authentication mode, effective only for iscsi (disabled|enabled)
connection_type = iscsi	(String(choices=[fibre_channel, iscsi])) Connection type to the IBM Storage Array
management_ips =	(String) List of Management IP addresses (separated by commas)
proxy = cinder.volume.drivers.ibm.ibm_stc	(String) Proxy driver that connects to the IBM Storage Array

- Description: IBM Storage driver

IBM Storage driver is a unified Volume driver for IBM XIV, Spectrum Accelerate, FlashSystem A9000, FlashSystem A9000R and DS8000 storage systems.

Version history:

```
2.0 - First open source driver version
2.1.0 - Support Consistency groups through Generic volume groups
      - Support XIV/A9000 Volume independent QoS
      - Support Consistency groups replication
2.3.0 - Support Report backend state
```

ISCSIVolumeDriver

- Version: 2.0.0
- volume_driver=cinder.volume.drivers.datacore.iscsi.ISCSIVolumeDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/DataCore_CI
- Driver Configuration Options:

Table 158: **Driver configuration options**

Name = Default Value	(Type) Description
datacore_api_timeout = 300	(Integer(min=1)) Seconds to wait for a response from a DataCore API call.
datacore_disk_failed_delay = 300	(Integer(min=0)) Seconds to wait for DataCore virtual disk to come out of the Failed state.
datacore_disk_pools = []	(List of String) List of DataCore disk pools that can be used by volume driver.
datacore_disk_type = single	(String(choices=[single, mirrored])) DataCore virtual disk type (single/mirrored). Mirrored virtual disks require two storage servers in the server group.
datacore_iscsi_chap_storage = $state_path/.datacore_chap	(String) Fully qualified file name where dynamically generated iSCSI CHAP secrets are stored. This must be changed to a unique per-backend value if deploying multiple DataCore backends on the same host.
datacore_iscsi_unallowed_targets = []	(List of String) List of iSCSI targets that cannot be used to attach volume. To prevent the DataCore iSCSI volume driver from using some front-end targets in volume attachment, specify this option and list the iqn and target machine for each target as the value, such as <iqn:target name>, <iqn:target name>, <iqn:target name>.
datacore_storage_profile = None	(String) DataCore virtual disk storage profile.
san_ip =	(String) IP address of SAN controller
san_login = admin	(String) Username for SAN controller
san_password =	(String) Password for SAN controller

- Description: DataCore SANsymphony iSCSI volume driver.

Version history:

```
1.0.0 - Initial driver
2.0.0 - Reintroduce the driver
```

InStorageMCSFCDriver

- Version: 1.0.0
- volume_driver=cinder.volume.drivers.inspur.instorage.instorage_fc.InStorageMCSFCDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/Inspur_CI
- Driver Configuration Options:

Table 159: **Driver configuration options**

Name = Default Value	(Type) Description
instorage_mcs_allow_tenant_qos = False	(Boolean) Allow tenants to specify QOS on create
instorage_mcs_localcopy_rate = 50	(Integer(min=1, max=100)) Specifies the InStorage LocalCopy copy rate to be used when creating a full volume copy. The default rate is 50, and the valid rates are 1-100.
instorage_mcs_localcopy_timeout = 120	(Integer(min=1, max=600)) Maximum number of seconds to wait for LocalCopy to be prepared.
instorage_mcs_vol_autoexpand = True	(Boolean) Storage system autoexpand parameter for volumes (True/False)
instorage_mcs_vol_compression = False	(Boolean) Storage system compression option for volumes
instorage_mcs_vol_grainsize = 256	(Integer(min=32, max=256)) Storage system grain size parameter for volumes (32/64/128/256)
instorage_mcs_vol_intier = True	(Boolean) Enable InTier for volumes
instorage_mcs_vol_iogrp = 0	(String) The I/O group in which to allocate volumes. It can be a comma-separated list in which case the driver will select an io_group based on least number of volumes associated with the io_group.
instorage_mcs_vol_rsize = 2	(Integer(min=-1, max=100)) Storage system space-efficiency parameter for volumes (percentage)
instorage_mcs_vol_warning = 0	(Integer(min=-1, max=100)) Storage system threshold for volume capacity warnings (percentage)
instorage_mcs_volpool_name = [volpool]	(List of String) Comma separated list of storage system storage pools for volumes.
instorage_san_secondary_ip = None	(String) Specifies secondary management IP or hostname to be used if san_ip is invalid or becomes inaccessible.

- Description: INSPUR InStorage MCS FC volume driver.

Version history:

```
1.0 - Initial driver
```

InStorageMCSISCSIDriver

- Version: 1.0.0
- volume_driver=cinder.volume.drivers.inspur.instorage.instorage_iscsi.InStorageMCSISCSIDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/Inspur_CI
- Driver Configuration Options:

Table 160: **Driver configuration options**

Name = Default Value	(Type) Description
instorage_mcs_allow_tenant_qos = False	(Boolean) Allow tenants to specify QOS on create
instorage_mcs_localcopy_rate = 50	(Integer(min=1, max=100)) Specifies the InStorage LocalCopy copy rate to be used when creating a full volume copy. The default rate is 50, and the valid rates are 1-100.
instorage_mcs_localcopy_timeout = 120	(Integer(min=1, max=600)) Maximum number of seconds to wait for LocalCopy to be prepared.
instorage_mcs_vol_autoexpand = True	(Boolean) Storage system autoexpand parameter for volumes (True/False)
instorage_mcs_vol_compression = False	(Boolean) Storage system compression option for volumes
instorage_mcs_vol_grainsize = 256	(Integer(min=32, max=256)) Storage system grain size parameter for volumes (32/64/128/256)
instorage_mcs_vol_intier = True	(Boolean) Enable InTier for volumes
instorage_mcs_vol_iogrp = 0	(String) The I/O group in which to allocate volumes. It can be a comma-separated list in which case the driver will select an io_group based on least number of volumes associated with the io_group.
instorage_mcs_vol_rsize = 2	(Integer(min=-1, max=100)) Storage system space-efficiency parameter for volumes (percentage)
instorage_mcs_vol_warning = 0	(Integer(min=-1, max=100)) Storage system threshold for volume capacity warnings (percentage)
instorage_mcs_volpool_name = [volpool]	(List of String) Comma separated list of storage system storage pools for volumes.
instorage_san_secondary_ip = None	(String) Specifies secondary management IP or hostname to be used if san_ip is invalid or becomes inaccessible.

- Description: Inspur InStorage iSCSI volume driver.

Version history:

```
1.0 - Initial driver
```

InfiniboxVolumeDriver

- Version: 1.15
- volume_driver=cinder.volume.drivers.infinidat.InfiniboxVolumeDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/INFINIDAT_CI
- Driver Configuration Options:

Table 161: **Driver configuration options**

Name = Default Value	(Type) Description
chap_password =	(String) Password for specified CHAP account name.
chap_username =	(String) CHAP user name.
driver_use_ssl = False	(Boolean) Tell driver to use SSL for connection to backend storage if the driver supports it.
enforce_multipath_for_image_xfer = False	(Boolean) If this is set to True, attachment of volumes for image transfer will be aborted when multipathd is not running. Otherwise, it will fallback to single path. This parameter needs to be configured for each backend section or in [backend_defaults] section as a common configuration for all backends.
infinidat_iscsi_netspaces = []	(List of String) List of names of network spaces to use for iSCSI connectivity
infinidat_pool_name = None	(String) Name of the pool from which volumes are allocated
infinidat_storage_protocol = fc	(String(choices=[iscsi, fc])) Protocol for transferring data between host and storage back-end.
infinidat_use_compression = None	(Boolean) Specifies whether to enable (true) or disable (false) compression for all newly created volumes. Leave this unset (commented out) for all created volumes to inherit their compression setting from their parent pool at creation time. The default value is unset.
max_over_subscription_ratio = 20.0	(String(regex=^(auto|\d*\.\d+|\d+)$)) Representation of the over subscription ratio when thin provisioning is enabled. Default ratio is 20.0, meaning provisioned capacity can be 20 times of the total physical capacity. If the ratio is 10.5, it means provisioned capacity can be 10.5 times of the total physical capacity. A ratio of 1.0 means provisioned capacity cannot exceed the total physical capacity. If ratio is auto, Cinder will automatically calculate the ratio based on the provisioned capacity and the used space. If not set to auto, the ratio has to be a minimum of 1.0.
num_volume_device_scan_tries = 3	(Integer) The maximum number of times to rescan targets to find volume
san_ip =	(String) IP address of SAN controller
san_login = admin	(String) Username for SAN controller
san_password =	(String) Password for SAN controller
san_thin_provision = True	(Boolean) Use thin provisioning for SAN volumes?
suppress_requests_ssl_warnings = False	(Boolean) Suppress requests library SSL certificate warnings.
use_chap_auth = False	(Boolean) Option to enable/disable CHAP authentication for targets.
use_multipath_for_image_xfer = False	(Boolean) Do we attach/detach volumes in cinder using multipath for volume to image and image to volume transfers? This parameter needs to be configured for each backend section or in [backend_defaults] section as a common configuration for all backends.
volume_dd_blocksize = 1M	(String) The default block size used when copying/clearing volumes

- Description: INFINIDAT InfiniBox Cinder driver.

Version history:

```
1.0 - initial release
1.1 - switched to use infinisdk package
1.2 - added support for iSCSI protocol
1.3 - added generic volume groups support
1.4 - added support for QoS
1.5 - added support for volume compression
1.6 - added support for volume multi-attach
1.7 - fixed iSCSI to return all portals
1.8 - added revert to snapshot
1.9 - added manage/unmanage/manageable-list volume/snapshot
1.10 - added support for TLS/SSL communication
1.11 - fixed generic volume migration
1.12 - fixed volume multi-attach
1.13 - fixed consistency groups feature
1.14 - added storage assisted volume migration
1.15 - fixed backup for attached volume
```

InfortrendCLIFCDriver

- Version: 2.1.4
- volume_driver=cinder.volume.drivers.infortrend.infortrend_fc_cli.InfortrendCLIFCDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/Infortrend_Storage_CI
- Driver Configuration Options:

Table 162: **Driver configuration options**

Name = Default Value	(Type) Description
infortrend_cli_cache = False	(Boolean) The Infortrend CLI cache. While set True, the RAID status report will use cache stored in the CLI. Never enable this unless the RAID is managed only by Openstack and only by one infortrend cinder-volume backend. Otherwise, CLI might report out-dated status to cinder and thus there might be some race condition among all backend/CLIs.
infortrend_cli_max_retries = 5	(Integer) The maximum retry times if a command fails.
infortrend_cli_path = /opt/bin/Infortrend/raidcmd_ES	(String) The Infortrend CLI absolute path.
infortrend_cli_timeout = 60	(Integer) The timeout for CLI in seconds.
infortrend_iqn_prefix = iqn.2002-10.com.infortrend	(String) Infortrend iqn prefix for iSCSI.
infortrend_pools_name =	(List of String) The Infortrend logical volumes name list. It is separated with comma.
infortrend_slots_a_channels_id =	(List of String) Infortrend raid channel ID list on Slot A for OpenStack usage. It is separated with comma.
infortrend_slots_b_channels_id =	(List of String) Infortrend raid channel ID list on Slot B for OpenStack usage. It is separated with comma.
java_path = /usr/bin/java	(String) The Java absolute path.

- Description: <None>

InfortrendCLIISCSIDriver

- Version: 2.1.4
- volume_driver=cinder.volume.drivers.infortrend.infortrend_iscsi_cli.InfortrendCLIISCSIDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/Infortrend_Storage_CI
- Driver Configuration Options:

Table 163: **Driver configuration options**

Name = Default Value	(Type) Description
infortrend_cli_cache = False	(Boolean) The Infortrend CLI cache. While set True, the RAID status report will use cache stored in the CLI. Never enable this unless the RAID is managed only by Openstack and only by one infortrend cinder-volume backend. Otherwise, CLI might report out-dated status to cinder and thus there might be some race condition among all backend/CLIs.
infortrend_cli_max_retries = 5	(Integer) The maximum retry times if a command fails.
infortrend_cli_path = /opt/bin/Infortrend/raidcmd_ES	(String) The Infortrend CLI absolute path.
infortrend_cli_timeout = 60	(Integer) The timeout for CLI in seconds.
infortrend_iqn_prefix = iqn.2002-10.com.infortrend	(String) Infortrend iqn prefix for iSCSI.
infortrend_pools_name =	(List of String) The Infortrend logical volumes name list. It is separated with comma.
infortrend_slots_a_channels_id =	(List of String) Infortrend raid channel ID list on Slot A for OpenStack usage. It is separated with comma.
infortrend_slots_b_channels_id =	(List of String) Infortrend raid channel ID list on Slot B for OpenStack usage. It is separated with comma.
java_path = /usr/bin/java	(String) The Java absolute path.

- Description: <None>

JovianISCSIDriver

- Version: 1.0.3
- volume_driver=cinder.volume.drivers.open_e.iscsi.JovianISCSIDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/Open-E_JovianDSS_CI
- Driver Configuration Options:

Table 164: **Driver configuration options**

Name = Default Value	(Type) Description
backend_availability_zone = None	(String) Availability zone for this volume backend. If not set, the storage_availability_zone option value is used as the default for all backends.
chap_password =	(String) Password for specified CHAP account name.
chap_username =	(String) CHAP user name.
chiscsi_conf = /etc/chelsio-iscsi/chiscsi.conf	(String) Chiscsi (CXT) global defaults configuration file
driver_client_cert = None	(String) The path to the client certificate for verification, if the driver supports it.
driver_client_cert_key = None	(String) The path to the client certificate key for verification, if the driver supports it.
driver_data_namespace = None	(String) Namespace for driver private data values to be saved in.
driver_ssl_cert_path = None	(String) Can be used to specify a non default path to a CA_BUNDLE file or directory with certificates of trusted CAs, which will be used to validate the backend
driver_ssl_cert_verify = False	(Boolean) If set to True the http client will validate the SSL certificate of the backend endpoint.
driver_use_ssl = False	(Boolean) Tell driver to use SSL for connection to backend storage if the driver supports it.
enable_unsupported_driver = False	(Boolean) Set this to True when you want to allow an unsupported driver to start. Drivers that havent maintained a working CI system and testing are marked as unsupported until CI is working again. This also marks a driver as deprecated and may be removed in the next release.
filter_function = None	(String) String representation for an equation that will be used to filter hosts. Only used when the driver filter is set to be used by the Cinder scheduler.
goodness_function = None	(String) String representation for an equation that will be used to determine the goodness of a host. Only used when using the goodness weigher is set to be used by the Cinder scheduler.
iscsi_iotype = fileio	(String(choices=[blockio, fileio, auto])) Sets the behavior of the iSCSI target to either perform blockio or fileio optionally, auto can be set and Cinder will autodetect type of backing device
iscsi_target_flags =	(String) Sets the target-specific flags for the iSCSI target. Only used for tgtadm to specify backing device flags using bsoflags option. The specified string is passed as is to the underlying tool.
iscsi_write_cache = on	(String(choices=[on, off])) Sets the behavior of the iSCSI target to either perform write-back(on) or write-through(off). This parameter is valid if target_helper is set to tgtadm.

continues on next page

Table 164 – continued from previous page

Name = Default Value	(Type) Description
max_over_subscription_ratio = 20.0	(String(regex=^(auto|\d*\.\d+|\d+)$)) Representation of the over subscription ratio when thin provisioning is enabled. Default ratio is 20.0, meaning provisioned capacity can be 20 times of the total physical capacity. If the ratio is 10.5, it means provisioned capacity can be 10.5 times of the total physical capacity. A ratio of 1.0 means provisioned capacity cannot exceed the total physical capacity. If ratio is auto, Cinder will automatically calculate the ratio based on the provisioned capacity and the used space. If not set to auto, the ratio has to be a minimum of 1.0.
num_shell_tries = 3	(Integer) Number of times to attempt to run flakey shell commands
num_volume_device_scan_tries = 3	(Integer) The maximum number of times to rescan targets to find volume
replication_device = None	(Dict of String) Multi opt of dictionaries to represent a replication target device. This option may be specified multiple times in a single config section to specify multiple replication target devices. Each entry takes the standard dict config form: replication_device = target_device_id:<required>,key1:value1,key2:value2
report_discard_supported = False	(Boolean) Report to clients of Cinder that the backend supports discard (aka. trim/unmap). This will not actually change the behavior of the backend or the client directly, it will only notify that it can be used.
reserved_percentage = 0	(Integer(min=0, max=100)) The percentage of backend capacity is reserved
storage_protocol = iSCSI	(String(choices=[iSCSI, FC])) Protocol for transferring data between host and storage back-end.
target_helper = tgtadm	(String(choices=[tgtadm, lioadm, scstadmin, iscsictl, nvmet, spdk-nvmeof, fake])) Target user-land tool to use. tgtadm is default, use lioadm for LIO iSCSI support, scstadmin for SCST target support, iscsictl for Chelsio iSCSI Target, nvmet for NVMEoF support, spdk-nvmeof for SPDK NVMe-oF, or fake for testing.
target_ip_address = $my_ip	(String) The IP address that the iSCSI/NVMEoF daemon is listening on
target_port = 3260	(Port(min=0, max=65535)) The port that the iSCSI/NVMEoF daemon is listening on
target_prefix = iqn.2010-10.org.openstack:	(String) Prefix for iSCSI/NVMEoF volumes
target_protocol = iscsi	(String(choices=[iscsi, iser, nvmet_rdma, nvmet_tcp])) Determines the target protocol for new volumes, created with tgtadm, lioadm and nvmet target helpers. In order to enable RDMA, this parameter should be set with the value iser. The supported iSCSI protocol values are iscsi and iser, in case of nvmet target set to nvmet_rdma or nvmet_tcp.
target_secondary_ip_addresses = []	(List of String) The list of secondary IP addresses of the iSCSI/NVMEoF daemon
trace_flags = None	(List of String) List of options that control which trace info is written to the DEBUG log level to assist developers. Valid values are method and api.

continues on next page

Table 164 – continued from previous page

Name = Default Value	(Type) Description
use_chap_auth = False	(Boolean) Option to enable/disable CHAP authentication for targets.
volume_backend_name = None	(String) The backend name for a given driver implementation
volume_clear = zero	(String(choices=[none, zero])) Method used to wipe old volumes
volume_clear_ionice = None	(String) The flag to pass to ionice to alter the i/o priority of the process used to zero a volume after deletion, for example -c3 for idle only priority.
volume_clear_size = 0	(Integer(max=1024)) Size in MiB to wipe at start of old volumes. 1024 MiB at max. 0 => all
volume_copy_blkio_cgroup_name = cinder-volume-copy	(String) The blkio cgroup name to be used to limit bandwidth of volume copy
volume_copy_bps_limit = 0	(Integer) The upper limit of bandwidth of volume copy. 0 => unlimited
volume_dd_blocksize = 1M	(String) The default block size used when copying/clearing volumes
volumes_dir = $state_path/volumes	(String) Volume configuration file storage directory

- Description: Executes volume driver commands on Open-E JovianDSS.

Version history:

```
1.0.0 - Open-E JovianDSS driver with basic functionality
1.0.1 - Added certificate support
        Added revert to snapshot support
1.0.2 - Added multi-attach support
        Added 16K block support
1.0.3 - Driver rework and optimisation
        Abandon recursive volume deletion
        Removed revert to snapshot support
```

KaminarioISCSIDriver

- Version: 1.4
- volume_driver=cinder.volume.drivers.kaminario.kaminario_iscsi.KaminarioISCSIDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/Kaminario_K2_CI
- Driver Configuration Options:

Table 165: **Driver configuration options**

Name = Default Value	(Type) Description
auto_calc_max_oversubscriptio = False	(Boolean) K2 driver will calculate max_oversubscription_ratio on setting this option as True.
disable_discovery = False	(Boolean) Disabling iSCSI discovery (sendtargets) for multipath connections on K2 driver.
replication_device = None	(Dict of String) Multi opt of dictionaries to represent a replication target device. This option may be specified multiple times in a single config section to specify multiple replication target devices. Each entry takes the standard dict config form: replication_device = target_device_id:<required>,key1:value1,key2:value2
san_ip =	(String) IP address of SAN controller
san_login = admin	(String) Username for SAN controller
san_password =	(String) Password for SAN controller
unique_fqdn_network = True	(Boolean) Whether or not our private network has unique FQDN on each initiator or not. For example networks with QA systems usually have multiple servers/VMs with the same FQDN. When true this will create host entries on 3PAR using the FQDN, when false it will use the reversed IQN/WWNN.
volume_dd_blocksize = 1M	(String) The default block size used when copying/clearing volumes

- Description: Kaminario K2 iSCSI Volume Driver.

```
Version history:

   1.0 - Initial driver
   1.1 - Added manage/unmanage and extra-specs support for nodedup
   1.2 - Added replication support
   1.3 - Added retype support
   1.4 - Added replication failback support
```

KumoScaleBaseVolumeDriver

- Version: 1.0.0
- volume_driver=cinder.volume.drivers.kioxia.kumoscale.KumoScaleBaseVolumeDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/KIOXIA_CI
- Driver Configuration Options:

Table 166: **Driver configuration options**

Name = Default Value	(Type) Description
kioxia_block_size = 4096	(Integer) Volume block size in bytes - 512 or 4096 (Default).
kioxia_cafile = None	(String) Cert for provisioner REST API SSL
kioxia_desired_bw_per_gb = 0	(Integer) Desired bandwidth in B/s per GB.
kioxia_desired_iops_per_gb = 0	(Integer) Desired IOPS/GB.
kioxia_max_bw_per_gb = 0	(Integer) Upper limit for bandwidth in B/s per GB.
kioxia_max_iops_per_gb = 0	(Integer) Upper limit for IOPS/GB.
kioxia_max_replica_down_time = 0	(Integer) Replicated volume max downtime for replica in minutes.
kioxia_num_replicas = 1	(Integer) Number of volume replicas.
kioxia_provisioning_type = THICK	(String(choices=[THICK, THIN])) Thin or thick volume, Default thick.
kioxia_same_rack_allowed = False	(Boolean) Can more than one replica be allocated to same rack.
kioxia_snap_reserved_space_pe = 0	(Integer) Percentage of the parent volume to be used for log.
kioxia_snap_vol_reserved_spac = 0	(Integer) Writable snapshot percentage of parent volume used for log.
kioxia_snap_vol_span_allowed = True	(Boolean) Allow span in snapshot volume - Default True.
kioxia_span_allowed = True	(Boolean) Allow span - Default True.
kioxia_token = None	(String) KumoScale Provisioner auth token.
kioxia_url = None	(String) KumoScale provisioner REST API URL
kioxia_vol_reserved_space_per = 0	(Integer) Thin volume reserved capacity allocation percentage.
kioxia_writable = False	(Boolean) Volumes from snapshot writeable or not.

- Description: Performs volume management on KumoScale Provisioner.

Version history:

```
1.0.0 - Initial driver version.
```

LVMVolumeDriver

- Version: 3.0.0
- volume_driver=cinder.volume.drivers.lvm.LVMVolumeDriver
- Driver Configuration Options:

Table 167: **Driver configuration options**

Name = Default Value	(Type) Description
iscsi_iotype = fileio	(String(choices=[blockio, fileio, auto])) Sets the behavior of the iSCSI target to either perform blockio or fileio optionally, auto can be set and Cinder will autodetect type of backing device

continues on next page

Table 167 – continued from previous page

Name = Default Value	(Type) Description
iscsi_target_flags =	(String) Sets the target-specific flags for the iSCSI target. Only used for tgtadm to specify backing device flags using bsoflags option. The specified string is passed as is to the underlying tool.
iscsi_write_cache = on	(String(choices=[on, off])) Sets the behavior of the iSCSI target to either perform write-back(on) or write-through(off). This parameter is valid if target_helper is set to tgtadm.
lvm_conf_file = /etc/cinder/lvm.conf	(String) LVM conf file to use for the LVM driver in Cinder; this setting is ignored if the specified file does not exist (You can also specify None to not use a conf file even if one exists).
lvm_mirrors = 0	(Integer) If >0, create LVs with multiple mirrors. Note that this requires lvm_mirrors + 2 PVs with available space
lvm_share_target = False	(Boolean) Whether to share the same target for all LUNs or not (currently only supported by nvmet.
lvm_suppress_fd_warnings = False	(Boolean) Suppress leaked file descriptor warnings in LVM commands.
lvm_type = auto	(String(choices=[default, thin, auto])) Type of LVM volumes to deploy; (default, thin, or auto). Auto defaults to thin if thin is supported.
max_over_subscription_ratio = 20.0	(String(regex=^(auto|\d*\.\d+|\d+)$)) Representation of the over subscription ratio when thin provisioning is enabled. Default ratio is 20.0, meaning provisioned capacity can be 20 times of the total physical capacity. If the ratio is 10.5, it means provisioned capacity can be 10.5 times of the total physical capacity. A ratio of 1.0 means provisioned capacity cannot exceed the total physical capacity. If ratio is auto, Cinder will automatically calculate the ratio based on the provisioned capacity and the used space. If not set to auto, the ratio has to be a minimum of 1.0.
nvmet_ns_id = 10	(Integer) Namespace id for the subsystem for the LVM volume when not sharing targets. The minimum id value when sharing.Maximum supported value in Linux is 8192
nvmet_port_id = 1	(Port(min=0, max=65535)) The id of the NVMe target port definition when not sharing targets. The starting port id value when sharing, incremented for each secondary ip address.
reserved_percentage = 0	(Integer(min=0, max=100)) The percentage of backend capacity is reserved
scst_target_driver = iscsi	(String) SCST target implementation can choose from multiple SCST target drivers.
scst_target_iqn_name = None	(String) Certain ISCSI targets have predefined target names, SCST target driver uses this name.
spdk_max_queue_depth = 64	(Integer(min=1, max=128)) Queue depth for rdma transport.
spdk_rpc_ip = None	(String) The NVMe target remote configuration IP address.
spdk_rpc_password = None	(String) The NVMe target remote configuration password.
spdk_rpc_port = 8000	(Port(min=0, max=65535)) The NVMe target remote configuration port.
spdk_rpc_username = None	(String) The NVMe target remote configuration username.

continues on next page

Table 167 – continued from previous page

Name = Default Value	(Type) Description
target_helper = tgtadm	(String(choices=[tgtadm, lioadm, scstadmin, iscsictl, nvmet, spdk-nvmeof, fake])) Target user-land tool to use. tgtadm is default, use lioadm for LIO iSCSI support, scstadmin for SCST target support, iscsictl for Chelsio iSCSI Target, nvmet for NVMEoF support, spdk-nvmeof for SPDK NVMe-oF, or fake for testing.
target_ip_address = $my_ip	(String) The IP address that the iSCSI/NVMEoF daemon is listening on
target_port = 3260	(Port(min=0, max=65535)) The port that the iSCSI/NVMEoF daemon is listening on
target_prefix = iqn.2010-10.org.openstack:	(String) Prefix for iSCSI/NVMEoF volumes
target_protocol = iscsi	(String(choices=[iscsi, iser, nvmet_rdma, nvmet_tcp])) Determines the target protocol for new volumes, created with tgtadm, lioadm and nvmet target helpers. In order to enable RDMA, this parameter should be set with the value iser. The supported iSCSI protocol values are iscsi and iser, in case of nvmet target set to nvmet_rdma or nvmet_tcp.
target_secondary_ip_addresses = []	(List of String) The list of secondary IP addresses of the iSCSI/NVMEoF daemon
volume_clear = zero	(String(choices=[none, zero])) Method used to wipe old volumes
volume_clear_size = 0	(Integer(max=1024)) Size in MiB to wipe at start of old volumes. 1024 MiB at max. 0 => all
volume_dd_blocksize = 1M	(String) The default block size used when copying/clearing volumes
volume_group = cinder-volumes	(String) Name for the VG that will contain exported volumes
volumes_dir = $state_path/volumes	(String) Volume configuration file storage directory

- Description: Executes commands relating to Volumes.

LenovoFCDriver

- Version: 2.0
- volume_driver=cinder.volume.drivers.lenovo.lenovo_fc.LenovoFCDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/Lenovo_Storage_CI
- Driver Configuration Options:

Table 168: **Driver configuration options**

Name = Default Value	(Type) Description
driver_ssl_cert_path = None	(String) Can be used to specify a non default path to a CA_BUNDLE file or directory with certificates of trusted CAs, which will be used to validate the backend
driver_ssl_cert_verify = False	(Boolean) If set to True the http client will validate the SSL certificate of the backend endpoint.
driver_use_ssl = False	(Boolean) Tell driver to use SSL for connection to backend storage if the driver supports it.
lenovo_api_protocol = https	(String(choices=[http, https])) Lenovo api interface protocol.
lenovo_pool_name = A	(String) Pool or Vdisk name to use for volume creation.
lenovo_pool_type = virtual	(String(choices=[linear, virtual])) linear (for VDisk) or virtual (for Pool).
lenovo_verify_certificate = False	(Boolean) Whether to verify Lenovo array SSL certificate.
lenovo_verify_certificate_path = None	(String) Lenovo array SSL certificate path.
san_ip =	(String) IP address of SAN controller
san_login = admin	(String) Username for SAN controller
san_password =	(String) Password for SAN controller

- Description: OpenStack Fibre Channel cinder drivers for Lenovo Storage arrays.

```
Version history:
    1.0    - Inheriting from DotHill cinder drivers.
    1.6    - Add management path redundancy and reduce load placed
             on management controller.
    2.0    - DotHill driver renamed to Seagate (STX)
```

LenovoISCSIDriver

- Version: 2.0
- volume_driver=cinder.volume.drivers.lenovo.lenovo_iscsi.LenovoISCSIDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/Lenovo_Storage_CI
- Driver Configuration Options:

Table 169: **Driver configuration options**

Name = Default Value	(Type) Description
driver_ssl_cert_path = None	(String) Can be used to specify a non default path to a CA_BUNDLE file or directory with certificates of trusted CAs, which will be used to validate the backend
driver_ssl_cert_verify = False	(Boolean) If set to True the http client will validate the SSL certificate of the backend endpoint.
driver_use_ssl = False	(Boolean) Tell driver to use SSL for connection to backend storage if the driver supports it.
lenovo_api_protocol = https	(String(choices=[http, https])) Lenovo api interface protocol.
lenovo_iscsi_ips = []	(List of String) List of comma-separated target iSCSI IP addresses.
lenovo_pool_name = A	(String) Pool or Vdisk name to use for volume creation.
lenovo_pool_type = virtual	(String(choices=[linear, virtual])) linear (for VDisk) or virtual (for Pool).
lenovo_verify_certificate = False	(Boolean) Whether to verify Lenovo array SSL certificate.
lenovo_verify_certificate_path = None	(String) Lenovo array SSL certificate path.
san_ip =	(String) IP address of SAN controller
san_login = admin	(String) Username for SAN controller
san_password =	(String) Password for SAN controller

- Description: OpenStack iSCSI cinder drivers for Lenovo Storage arrays.

```
Version history:
    1.0    - Inheriting from DotHill cinder drivers.
    1.6    - Add management path redundancy and reduce load placed
             on management controller.
    2.0    - DotHill driver renamed to Seagate (STX)
```

LightOSVolumeDriver

- Version: 2.3.12
- volume_driver=cinder.volume.drivers.lightos.LightOSVolumeDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/LightbitsLabs_CI
- Driver Configuration Options:

Table 170: **Driver configuration options**

Name = Default Value	(Type) Description
driver_ssl_cert_verify = False	(Boolean) If set to True the http client will validate the SSL certificate of the backend endpoint.
lightos_api_address = None	(List of IPAddress) The IP addresses of the LightOS API servers separated by commas.
lightos_api_port = 443	(Port(min=0, max=65535)) The TCP/IP port at which the LightOS API endpoints listen. Port 443 is used for HTTPS and other values are used for HTTP.
lightos_api_service_snapshots_ = 5	(Integer) The maximum number of calls to the LightOS when creating snapshots. The default is 5 calls.
lightos_api_service_timeout = 30	(Integer) The default amount of time (in seconds) to wait for an API endpoint response.
lightos_default_compression_e = False	(Boolean) Set to True to create new volumes compressed assuming no other compression setting is specified via the volumes type.
lightos_default_num_replicas = 3	(Integer(min=1, max=3)) The default number of replicas to create for each volume.
lightos_jwt = None	(String) JWT to be used for volume and snapshot operations with the LightOS cluster. Do not set this parameter if the cluster is installed with multi-tenancy disabled.
lightos_use_ipacl = True	(Boolean) IPACL work in conjunction with the standard NVME ACL. A host must be in both the IPACL and the ACL of a volume to access that volume. Cinder always sets the volume's ACL. If lightos_use_ipacl is set to True, Cinder will also add the host's IP addresses to a volume IPACL. If set to False, any IP address may access the volume. The default is True.
reserved_percentage = 0	(Integer(min=0, max=100)) The percentage of backend capacity is reserved
volume_backend_name = None	(String) The backend name for a given driver implementation

- Description: OpenStack NVMe/TCP cinder drivers for Lightbits LightOS.

```
Version history:
    2.3.12 - Initial upstream driver version.
```

LinstorDrbdDriver

- Version: 1.0.1
- volume_driver=cinder.volume.drivers.linstordrv.LinstorDrbdDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/LINBIT_LINSTOR_CI
- Driver Configuration Options:

Table 171: **Driver configuration options**

Name = Default Value	(Type) Description
linstor_autoplace_count = 0	(Integer) Autoplace replication count on volume deployment. 0 = Full cluster replication without autoplace, 1 = Single node deployment without replication, 2 or greater = Replicated deployment with autoplace.
linstor_controller_diskless = True	(Boolean) True means Cinder node is a diskless LINSTOR node.
linstor_default_blocksize = 4096	(Integer) Default Block size for Image restoration. When using iSCSI transport, this option specifies the block size.
linstor_default_storage_pool_nam = DfltStorPool	(String) Default Storage Pool name for LINSTOR.
linstor_default_uri = linstor://localhost	(String) Default storage URI for LINSTOR.
linstor_default_volume_group_na = drbd-vg	(String) Default Volume Group name for LINSTOR. Not Cinder Volume.
linstor_volume_downsize_factor = 4096	(Float) Default volume downscale size in KiB = 4 MiB.

- Description: Cinder DRBD driver that uses LINSTOR for storage.

LinstorIscsiDriver

- Version: 1.0.1
- volume_driver=cinder.volume.drivers.linstordrv.LinstorIscsiDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/LINBIT_LINSTOR_CI
- Driver Configuration Options:

Table 172: **Driver configuration options**

Name = Default Value	(Type) Description
linstor_autoplace_count = 0	(Integer) Autoplace replication count on volume deployment. 0 = Full cluster replication without autoplace, 1 = Single node deployment without replication, 2 or greater = Replicated deployment with autoplace.
linstor_controller_diskless = True	(Boolean) True means Cinder node is a diskless LINSTOR node.
linstor_default_blocksize = 4096	(Integer) Default Block size for Image restoration. When using iSCSI transport, this option specifies the block size.
linstor_default_storage_pool_nam = DfltStorPool	(String) Default Storage Pool name for LINSTOR.
linstor_default_uri = linstor://localhost	(String) Default storage URI for LINSTOR.
linstor_default_volume_group_na = drbd-vg	(String) Default Volume Group name for LINSTOR. Not Cinder Volume.
linstor_volume_downsize_factor = 4096	(Float) Default volume downscale size in KiB = 4 MiB.

- Description: Cinder iSCSI driver that uses LINSTOR for storage.

MStorageFCDriver

- Version: 1.11.1
- volume_driver=cinder.volume.drivers.nec.volume.MStorageFCDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/NEC_Cinder_CI
- Driver Configuration Options:

Table 173: **Driver configuration options**

Name = Default Value	(Type) Description
nec_actual_free_capacity = False	(Boolean) Return actual free capacity.
nec_auto_accesscontrol = True	(Boolean) Configure access control automatically.
nec_backend_max_ld_count = 1024	(Integer) Maximum number of managing sessions.
nec_backup_ldname_format = LX:%s	(String) M-Series Storage LD name format for snapshots.
nec_backup_pools = []	(List of String) M-Series Storage backup pool number to be used.
nec_cv_ldname_format = LX:__ControlVolume_%xh	(String) M-Series Storage Control Volume name format.
nec_diskarray_name =	(String) Diskarray name of M-Series Storage.
nec_iscsi_portals_per_cont = 0	(Integer) Max number of iSCSI portals per controller. 0 => unlimited. This option is deprecated and may be removed in the next release.
nec_ismcli_fip = None	(IPAddress) FIP address of M-Series Storage iSMCLI.
nec_ismcli_password =	(String) Password for M-Series Storage iSMCLI.
nec_ismcli_privkey =	(String) Filename of RSA private key for M-Series Storage iSMCLI.
nec_ismcli_user =	(String) User name for M-Series Storage iSMCLI.
nec_ismview_alloptimize = False	(Boolean) Use legacy iSMCLI command with optimization.
nec_ismview_dir = /tmp/nec/cinder	(String) Output path of iSMview file.
nec_ldname_format = LX:%s	(String) M-Series Storage LD name format for volumes.
nec_ldset =	(String) M-Series Storage LD Set name for Compute Node.
nec_pools = []	(List of String) M-Series Storage pool numbers list to be used.
nec_queryconfig_view = False	(Boolean) Use legacy iSMCLI command.
nec_ssh_pool_port_number = 22	(Integer) Port number of ssh pool.
nec_unpairthread_timeout = 3600	(Integer) Timeout value of Unpairthread.

- Description: M-Series Storage Snapshot FC Driver.

```
Version history:

  1.8.1 - First open source driver version.
  1.8.2 - Code refactoring.
  1.9.1 - Support optimal path for non-disruptive backup.
  1.9.2 - Support manage/unmanage and manage/unmanage snapshot.
          Delete an unused configuration
          parameter (ldset_controller_node_name).
          Fixed bug #1705001: driver fails to start.
  1.10.1 - Support automatic configuration of SAN access control.
           Fixed bug #1753375: SAN access remains permitted on the
           source node.
  1.10.2 - Delete max volumes per pool limit.
```

(continues on next page)

(continued from previous page)

```
1.10.3 - Add faster clone status check.
         Fixed bug #1777385: driver removed access permission from
         the destination node after live-migraion.
         Fixed bug #1778669: LUNs of detached volumes are never reused.
1.11.1 - Add support python 3.
         Add support for multi-attach.
         Add support of more than 4 iSCSI portals for a node.
         Add support to revert a volume to a snapshot.
         Add support storage assist retype and fixed bug #1838955:
         a volume in NEC Storage was left undeleted when the volume
         was retyped to another storage.
```

MStorageISCSIDriver

- Version: 1.11.1
- volume_driver=cinder.volume.drivers.nec.volume.MStorageISCSIDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/NEC_Cinder_CI
- Driver Configuration Options:

Table 174: **Driver configuration options**

Name = Default Value	(Type) Description
nec_actual_free_capacity = False	(Boolean) Return actual free capacity.
nec_auto_accesscontrol = True	(Boolean) Configure access control automatically.
nec_backend_max_ld_count = 1024	(Integer) Maximum number of managing sessions.
nec_backup_ldname_format = LX:%s	(String) M-Series Storage LD name format for snapshots.
nec_backup_pools = []	(List of String) M-Series Storage backup pool number to be used.
nec_cv_ldname_format = LX:__ControlVolume_%xh	(String) M-Series Storage Control Volume name format.
nec_diskarray_name =	(String) Diskarray name of M-Series Storage.
nec_iscsi_portals_per_cont = 0	(Integer) Max number of iSCSI portals per controller. 0 => unlimited. This option is deprecated and may be removed in the next release.
nec_ismcli_fip = None	(IPAddress) FIP address of M-Series Storage iSMCLI.
nec_ismcli_password =	(String) Password for M-Series Storage iSMCLI.
nec_ismcli_privkey =	(String) Filename of RSA private key for M-Series Storage iSMCLI.
nec_ismcli_user =	(String) User name for M-Series Storage iSMCLI.
nec_ismview_alloptimize = False	(Boolean) Use legacy iSMCLI command with optimization.
nec_ismview_dir = /tmp/nec/cinder	(String) Output path of iSMview file.
nec_ldname_format = LX:%s	(String) M-Series Storage LD name format for volumes.
nec_ldset =	(String) M-Series Storage LD Set name for Compute Node.
nec_pools = []	(List of String) M-Series Storage pool numbers list to be used.
nec_queryconfig_view = False	(Boolean) Use legacy iSMCLI command.
nec_ssh_pool_port_number = 22	(Integer) Port number of ssh pool.
nec_unpairthread_timeout = 3600	(Integer) Timeout value of Unpairthread.

- Description: M-Series Storage Snapshot iSCSI Driver.

```
Version history:

  1.8.1 - First open source driver version.
  1.8.2 - Code refactoring.
  1.9.1 - Support optimal path for non-disruptive backup.
  1.9.2 - Support manage/unmanage and manage/unmanage snapshot.
          Delete an unused configuration
          parameter (ldset_controller_node_name).
          Fixed bug #1705001: driver fails to start.
  1.10.1 - Support automatic configuration of SAN access control.
           Fixed bug #1753375: SAN access remains permitted on the
           source node.
  1.10.2 - Delete max volumes per pool limit.
```

(continues on next page)

(continued from previous page)

```
1.10.3 - Add faster clone status check.
         Fixed bug #1777385: driver removed access permission from
         the destination node after live-migraion.
         Fixed bug #1778669: LUNs of detached volumes are never reused.
1.11.1 - Add support python 3.
         Add support for multi-attach.
         Add support of more than 4 iSCSI portals for a node.
         Add support to revert a volume to a snapshot.
         Add support storage assist retype and fixed bug #1838955:
         a volume in NEC Storage was left undeleted when the volume
         was retyped to another storage.
```

MacroSANFCDriver

- Version: 1.0.1
- volume_driver=cinder.volume.drivers.macrosan.driver.MacroSANFCDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/MacroSAN_Volume_CI
- Driver Configuration Options:

Table 175: **Driver configuration options**

Name = Default Value	(Type) Description
backend_availability_zone = None	(String) Availability zone for this volume backend. If not set, the storage_availability_zone option value is used as the default for all backends.
chap_password =	(String) Password for specified CHAP account name.
chap_username =	(String) CHAP user name.
chiscsi_conf = /etc/chelsio-iscsi/chiscsi.conf	(String) Chiscsi (CXT) global defaults configuration file
driver_client_cert = None	(String) The path to the client certificate for verification, if the driver supports it.
driver_client_cert_key = None	(String) The path to the client certificate key for verification, if the driver supports it.
driver_data_namespace = None	(String) Namespace for driver private data values to be saved in.
driver_ssl_cert_path = None	(String) Can be used to specify a non default path to a CA_BUNDLE file or directory with certificates of trusted CAs, which will be used to validate the backend
driver_ssl_cert_verify = False	(Boolean) If set to True the http client will validate the SSL certificate of the backend endpoint.
driver_use_ssl = False	(Boolean) Tell driver to use SSL for connection to backend storage if the driver supports it.
enable_unsupported_driver = False	(Boolean) Set this to True when you want to allow an unsupported driver to start. Drivers that havent maintained a working CI system and testing are marked as unsupported until CI is working again. This also marks a driver as deprecated and may be removed in the next release.

continues on next page

Table 175 – continued from previous page

Name = Default Value	(Type) Description
filter_function = None	(String) String representation for an equation that will be used to filter hosts. Only used when the driver filter is set to be used by the Cinder scheduler.
goodness_function = None	(String) String representation for an equation that will be used to determine the goodness of a host. Only used when using the goodness weigher is set to be used by the Cinder scheduler.
iscsi_iotype = fileio	(String(choices=[blockio, fileio, auto])) Sets the behavior of the iSCSI target to either perform blockio or fileio optionally, auto can be set and Cinder will autodetect type of backing device
iscsi_target_flags =	(String) Sets the target-specific flags for the iSCSI target. Only used for tgtadm to specify backing device flags using bsoflags option. The specified string is passed as is to the underlying tool.
iscsi_write_cache = on	(String(choices=[on, off])) Sets the behavior of the iSCSI target to either perform write-back(on) or write-through(off). This parameter is valid if target_helper is set to tgtadm.
max_over_subscription_ratio = 20.0	(String(regex=^(auto|\d*\.\d+|\d+)$)) Representation of the over subscription ratio when thin provisioning is enabled. Default ratio is 20.0, meaning provisioned capacity can be 20 times of the total physical capacity. If the ratio is 10.5, it means provisioned capacity can be 10.5 times of the total physical capacity. A ratio of 1.0 means provisioned capacity cannot exceed the total physical capacity. If ratio is auto, Cinder will automatically calculate the ratio based on the provisioned capacity and the used space. If not set to auto, the ratio has to be a minimum of 1.0.
num_shell_tries = 3	(Integer) Number of times to attempt to run flakey shell commands
num_volume_device_scan_tries = 3	(Integer) The maximum number of times to rescan targets to find volume
replication_device = None	(Dict of String) Multi opt of dictionaries to represent a replication target device. This option may be specified multiple times in a single config section to specify multiple replication target devices. Each entry takes the standard dict config form: replication_device = target_device_id:<required>,key1:value1,key2:value2
report_discard_supported = False	(Boolean) Report to clients of Cinder that the backend supports discard (aka. trim/unmap). This will not actually change the behavior of the backend or the client directly, it will only notify that it can be used.
reserved_percentage = 0	(Integer(min=0, max=100)) The percentage of backend capacity is reserved
storage_protocol = iSCSI	(String(choices=[iSCSI, FC])) Protocol for transferring data between host and storage back-end.
target_helper = tgtadm	(String(choices=[tgtadm, lioadm, scstadmin, iscsictl, nvmet, spdk-nvmeof, fake])) Target user-land tool to use. tgtadm is default, use lioadm for LIO iSCSI support, scstadmin for SCST target support, iscsictl for Chelsio iSCSI Target, nvmet for NVMEoF support, spdk-nvmeof for SPDK NVMe-oF, or fake for testing.
target_ip_address = $my_ip	(String) The IP address that the iSCSI/NVMEoF daemon is listening on

continues on next page

Table 175 – continued from previous page

Name = Default Value	(Type) Description
target_port = 3260	(Port(min=0, max=65535)) The port that the iSCSI/NVMEoF daemon is listening on
target_prefix = iqn.2010-10.org.openstack:	(String) Prefix for iSCSI/NVMEoF volumes
target_protocol = iscsi	(String(choices=[iscsi, iser, nvmet_rdma, nvmet_tcp])) Determines the target protocol for new volumes, created with tgtadm, lioadm and nvmet target helpers. In order to enable RDMA, this parameter should be set with the value iser. The supported iSCSI protocol values are iscsi and iser, in case of nvmet target set to nvmet_rdma or nvmet_tcp.
target_secondary_ip_addresses = []	(List of String) The list of secondary IP addresses of the iSCSI/NVMEoF daemon
trace_flags = None	(List of String) List of options that control which trace info is written to the DEBUG log level to assist developers. Valid values are method and api.
use_chap_auth = False	(Boolean) Option to enable/disable CHAP authentication for targets.
volume_backend_name = None	(String) The backend name for a given driver implementation
volume_clear = zero	(String(choices=[none, zero])) Method used to wipe old volumes
volume_clear_ionice = None	(String) The flag to pass to ionice to alter the i/o priority of the process used to zero a volume after deletion, for example -c3 for idle only priority.
volume_clear_size = 0	(Integer(max=1024)) Size in MiB to wipe at start of old volumes. 1024 MiB at max. 0 => all
volume_copy_blkio_cgroup_name = cinder-volume-copy	(String) The blkio cgroup name to be used to limit bandwidth of volume copy
volume_copy_bps_limit = 0	(Integer) The upper limit of bandwidth of volume copy. 0 => unlimited
volume_dd_blocksize = 1M	(String) The default block size used when copying/clearing volumes
volumes_dir = $state_path/volumes	(String) Volume configuration file storage directory

- Description: FC driver for MacroSan storage arrays.

Version history:

```
1.0.0 - Initial driver
1.0.1 - Adjust some log level and text prompts; Remove some useless
functions; Add Cinder trace decorator. #1837920
```

MacroSANISCSIDriver

- Version: 1.0.1
- volume_driver=cinder.volume.drivers.macrosan.driver.MacroSANISCSIDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/MacroSAN_Volume_CI
- Driver Configuration Options:

Table 176: **Driver configuration options**

Name = Default Value	(Type) Description
backend_availability_zone = None	(String) Availability zone for this volume backend. If not set, the storage_availability_zone option value is used as the default for all backends.
chap_password =	(String) Password for specified CHAP account name.
chap_username =	(String) CHAP user name.
chiscsi_conf = /etc/chelsio-iscsi/chiscsi.conf	(String) Chiscsi (CXT) global defaults configuration file
driver_client_cert = None	(String) The path to the client certificate for verification, if the driver supports it.
driver_client_cert_key = None	(String) The path to the client certificate key for verification, if the driver supports it.
driver_data_namespace = None	(String) Namespace for driver private data values to be saved in.
driver_ssl_cert_path = None	(String) Can be used to specify a non default path to a CA_BUNDLE file or directory with certificates of trusted CAs, which will be used to validate the backend
driver_ssl_cert_verify = False	(Boolean) If set to True the http client will validate the SSL certificate of the backend endpoint.
driver_use_ssl = False	(Boolean) Tell driver to use SSL for connection to backend storage if the driver supports it.
enable_unsupported_driver = False	(Boolean) Set this to True when you want to allow an unsupported driver to start. Drivers that havent maintained a working CI system and testing are marked as unsupported until CI is working again. This also marks a driver as deprecated and may be removed in the next release.
filter_function = None	(String) String representation for an equation that will be used to filter hosts. Only used when the driver filter is set to be used by the Cinder scheduler.
goodness_function = None	(String) String representation for an equation that will be used to determine the goodness of a host. Only used when using the goodness weigher is set to be used by the Cinder scheduler.
iscsi_iotype = fileio	(String(choices=[blockio, fileio, auto])) Sets the behavior of the iSCSI target to either perform blockio or fileio optionally, auto can be set and Cinder will autodetect type of backing device
iscsi_target_flags =	(String) Sets the target-specific flags for the iSCSI target. Only used for tgtadm to specify backing device flags using bsoflags option. The specified string is passed as is to the underlying tool.
iscsi_write_cache = on	(String(choices=[on, off])) Sets the behavior of the iSCSI target to either perform write-back(on) or write-through(off). This parameter is valid if target_helper is set to tgtadm.

continues on next page

Table 176 – continued from previous page

Name = Default Value	(Type) Description
max_over_subscription_ratio = 20.0	(String(regex=^(auto|\d*\.\d+|\d+)$)) Representation of the over subscription ratio when thin provisioning is enabled. Default ratio is 20.0, meaning provisioned capacity can be 20 times of the total physical capacity. If the ratio is 10.5, it means provisioned capacity can be 10.5 times of the total physical capacity. A ratio of 1.0 means provisioned capacity cannot exceed the total physical capacity. If ratio is auto, Cinder will automatically calculate the ratio based on the provisioned capacity and the used space. If not set to auto, the ratio has to be a minimum of 1.0.
num_shell_tries = 3	(Integer) Number of times to attempt to run flakey shell commands
num_volume_device_scan_tries = 3	(Integer) The maximum number of times to rescan targets to find volume
replication_device = None	(Dict of String) Multi opt of dictionaries to represent a replication target device. This option may be specified multiple times in a single config section to specify multiple replication target devices. Each entry takes the standard dict config form: replication_device = target_device_id:<required>,key1:value1,key2:value2
report_discard_supported = False	(Boolean) Report to clients of Cinder that the backend supports discard (aka. trim/unmap). This will not actually change the behavior of the backend or the client directly, it will only notify that it can be used.
reserved_percentage = 0	(Integer(min=0, max=100)) The percentage of backend capacity is reserved
storage_protocol = iSCSI	(String(choices=[iSCSI, FC])) Protocol for transferring data between host and storage back-end.
target_helper = tgtadm	(String(choices=[tgtadm, lioadm, scstadmin, iscsictl, nvmet, spdk-nvmeof, fake])) Target user-land tool to use. tgtadm is default, use lioadm for LIO iSCSI support, scstadmin for SCST target support, iscsictl for Chelsio iSCSI Target, nvmet for NVMEoF support, spdk-nvmeof for SPDK NVMe-oF, or fake for testing.
target_ip_address = $my_ip	(String) The IP address that the iSCSI/NVMEoF daemon is listening on
target_port = 3260	(Port(min=0, max=65535)) The port that the iSCSI/NVMEoF daemon is listening on
target_prefix = iqn.2010-10.org.openstack:	(String) Prefix for iSCSI/NVMEoF volumes
target_protocol = iscsi	(String(choices=[iscsi, iser, nvmet_rdma, nvmet_tcp])) Determines the target protocol for new volumes, created with tgtadm, lioadm and nvmet target helpers. In order to enable RDMA, this parameter should be set with the value iser. The supported iSCSI protocol values are iscsi and iser, in case of nvmet target set to nvmet_rdma or nvmet_tcp.
target_secondary_ip_addresses = []	(List of String) The list of secondary IP addresses of the iSCSI/NVMEoF daemon
trace_flags = None	(List of String) List of options that control which trace info is written to the DEBUG log level to assist developers. Valid values are method and api.

continues on next page

Table 176 – continued from previous page

Name = Default Value	(Type) Description
use_chap_auth = False	(Boolean) Option to enable/disable CHAP authentication for targets.
volume_backend_name = None	(String) The backend name for a given driver implementation
volume_clear = zero	(String(choices=[none, zero])) Method used to wipe old volumes
volume_clear_ionice = None	(String) The flag to pass to ionice to alter the i/o priority of the process used to zero a volume after deletion, for example -c3 for idle only priority.
volume_clear_size = 0	(Integer(max=1024)) Size in MiB to wipe at start of old volumes. 1024 MiB at max. 0 => all
volume_copy_blkio_cgroup_name = cinder-volume-copy	(String) The blkio cgroup name to be used to limit bandwidth of volume copy
volume_copy_bps_limit = 0	(Integer) The upper limit of bandwidth of volume copy. 0 => unlimited
volume_dd_blocksize = 1M	(String) The default block size used when copying/clearing volumes
volumes_dir = $state_path/volumes	(String) Volume configuration file storage directory

- Description: ISCSI driver for MacroSan storage arrays.

Version history:

```
1.0.0 - Initial driver
1.0.1 - Adjust some log level and text prompts; Remove some useless
functions; Add Cinder trace decorator. #1837920
```

NetAppCmodeFibreChannelDriver

- Version: 4.0.0
- volume_driver=cinder.volume.drivers.netapp.dataontap.fc_cmode.NetAppCmodeFibreChannelDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/NetApp_CI
- Driver Configuration Options:

Table 177: **Driver configuration options**

Name = Default Value	(Type) Description
netapp_vserver = None	(String) This option specifies the virtual storage server (Vserver) name on the storage cluster on which provisioning of block storage volumes should occur.

- Description: NetApp C-mode FibreChannel volume driver.

Version history:

```
1.0.0 - Driver development before Wallaby
2.0.0 - Wallaby driver version bump
3.0.0 - Add support for Intra-cluster Storage assisted volume migration
```

(continues on next page)

(continued from previous page)

```
        Add support for revert to snapshot
4.0.0 - Add Cinder Active/Active support (High Availability)
        Implement Active/Active replication support
```

NetAppCmodeISCSIDriver

- Version: 4.0.0
- volume_driver=cinder.volume.drivers.netapp.dataontap.iscsi_cmode.NetAppCmodeISCSIDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/NetApp_CI
- Driver Configuration Options:

Table 178: **Driver configuration options**

Name = Default Value	(Type) Description
netapp_vserver = None	(String) This option specifies the virtual storage server (Vserver) name on the storage cluster on which provisioning of block storage volumes should occur.

- Description: NetApp C-mode iSCSI volume driver.

NetAppCmodeNVMeDriver

- Version: 1.0.0
- volume_driver=cinder.volume.drivers.netapp.dataontap.nvme_cmode.NetAppCmodeNVMeDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/NetApp_CI
- Driver Configuration Options:

Table 179: **Driver configuration options**

Name = Default Value	(Type) Description
netapp_vserver = None	(String) This option specifies the virtual storage server (Vserver) name on the storage cluster on which provisioning of block storage volumes should occur.

- Description: NetApp C-mode NVMe volume driver.

Version history:

```
1.0.0 - Initial driver
```

NetAppCmodeNfsDriver

- Version: 4.0.0
- volume_driver=cinder.volume.drivers.netapp.dataontap.nfs_cmode.NetAppCmodeNfsDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/NetApp_CI

- Driver Configuration Options:

Table 180: **Driver configuration options**

Name = Default Value	(Type) Description
nas_host =	(String) IP address or Hostname of NAS system.
nas_login = admin	(String) User name to connect to NAS system.
nas_mount_options = None	(String) Options used to mount the storage backend file system where Cinder volumes are stored.
nas_password =	(String) Password to connect to NAS system.
nas_private_key =	(String) Filename of private key to use for SSH authentication.
nas_secure_file_operations = auto	(String) Allow network-attached storage systems to operate in a secure environment where root level access is not permitted. If set to False, access is as the root user and insecure. If set to True, access is not as root. If set to auto, a check is done to determine if this is a new installation: True is used if so, otherwise False. Default is auto.
nas_secure_file_permissions = auto	(String) Set more secure file permissions on network-attached storage volume files to restrict broad other/world access. If set to False, volumes are created with open permissions. If set to True, volumes are created with permissions for the cinder user and group (660). If set to auto, a check is done to determine if this is a new installation: True is used if so, otherwise False. Default is auto.
nas_share_path =	(String) Path to the share to use for storing Cinder volumes. For example: /srv/export1 for an NFS server export available at 10.0.5.10:/srv/export1 .
nas_ssh_port = 22	(Port(min=0, max=65535)) SSH port to use to connect to NAS system.
nfs_mount_attempts = 3	(Integer) The number of attempts to mount NFS shares before raising an error. At least one attempt will be made to mount an NFS share, regardless of the value specified.
nfs_mount_options = None	(String) Mount options passed to the NFS client. See the NFS(5) man page for details.
nfs_mount_point_base = $state_path/mnt	(String) Base dir containing mount points for NFS shares.
nfs_qcow2_volumes = False	(Boolean) Create volumes as QCOW2 files rather than raw files.
nfs_shares_config = /etc/cinder/nfs_shares	(String) File with the list of available NFS shares.
nfs_snapshot_support = False	(Boolean) Enable support for snapshots on the NFS driver. Platforms using libvirt <1.2.7 will encounter issues with this feature.
nfs_sparsed_volumes = True	(Boolean) Create volumes as sparsed files which take no space. If set to False volume is created as regular file. In such case volume creation takes a lot of time.

- Description: NetApp NFS driver for Data ONTAP (Cluster-mode).

Version history:

```
1.0.0 - Driver development before Wallaby
2.0.0 - Add support for QoS minimums specs
        Add support for dynamic Adaptive QoS policy group creation
        Implement FlexGroup pool
```

(continues on next page)

(continued from previous page)

```
3.0.0 - Add support for Intra-cluster Storage assisted volume migration
        Add support for revert to snapshot
4.0.0 - Add Cinder Active/Active support (High Availability)
        Implement Active/Active replication support
```

NexentaISCSIDriver

- Version: 1.3.1
- volume_driver=cinder.volume.drivers.nexenta.iscsi.NexentaISCSIDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/Nexenta_CI
- Driver Configuration Options:

Table 181: **Driver configuration options**

Name = Default Value	(Type) Description
nexenta_blocksize = 4096	(Integer) Block size for datasets
nexenta_dataset_compression = on	(String(choices=[on, off, gzip, gzip-1, gzip-2, gzip-3, gzip-4, gzip-5, gzip-6, gzip-7, gzip-8, gzip-9, lzjb, zle, lz4])) Compression value for new ZFS folders.
nexenta_dataset_dedup = off	(String(choices=[on, off, sha256, verify, sha256, verify])) Deduplication value for new ZFS folders.
nexenta_dataset_description =	(String) Human-readable description for the folder.
nexenta_folder =	(String) A folder where cinder created datasets will reside.
nexenta_group_snapshot_template = group-snapshot-%s	(String) Template string to generate group snapshot name
nexenta_host =	(String) IP address of NexentaStor Appliance
nexenta_host_group_prefix = cinder	(String) Prefix for iSCSI host groups on NexentaStor
nexenta_iscsi_target_host_group = all	(String) Group of hosts which are allowed to access volumes
nexenta_iscsi_target_portal_groups =	(String) NexentaStor target portal groups
nexenta_iscsi_target_portal_port = 3260	(Integer) Nexenta appliance iSCSI target portal port
nexenta_iscsi_target_portals =	(String) Comma separated list of portals for NexentaStor5, in format of IP1:port1,IP2:port2. Port is optional, default=3260. Example: 10.10.10.1:3267,10.10.1.2
nexenta_lu_writebackcache_disabled = False	(Boolean) Postponed write to backing store or not
nexenta_luns_per_target = 100	(Integer) Amount of LUNs per iSCSI target
nexenta_ns5_blocksize = 32	(Integer) Block size for datasets

continues on next page

Table 181 – continued from previous page

Name = Default Value	(Type) Description
nexenta_origin_snapshot_template = origin-snapshot-%s	(String) Template string to generate origin name of clone
nexenta_password = nexenta	(String) Password to connect to NexentaStor management REST API server
nexenta_rest_address =	(String) IP address of NexentaStor management REST API endpoint
nexenta_rest_backoff_factor = 0.5	(Float) Specifies the backoff factor to apply between connection attempts to NexentaStor management REST API server
nexenta_rest_connect_timeout = 30	(Float) Specifies the time limit (in seconds), within which the connection to NexentaStor management REST API server must be established
nexenta_rest_port = 0	(Integer) HTTP(S) port to connect to NexentaStor management REST API server. If it is equal zero, 8443 for HTTPS and 8080 for HTTP is used
nexenta_rest_protocol = auto	(String(choices=[http, https, auto])) Use http or https for NexentaStor management REST API connection (default auto)
nexenta_rest_read_timeout = 300	(Float) Specifies the time limit (in seconds), within which NexentaStor management REST API server must send a response
nexenta_rest_retry_count = 3	(Integer) Specifies the number of times to repeat NexentaStor management REST API call in case of connection errors and NexentaStor appliance EBUSY or ENOENT errors
nexenta_rrmgr_compression = 0	(Integer) Enable stream compression, level 1..9. 1 - gives best speed; 9 - gives best compression.
nexenta_rrmgr_connections = 2	(Integer) Number of TCP connections.
nexenta_rrmgr_tcp_buf_size = 4096	(Integer) TCP Buffer size in KiloBytes.
nexenta_sparse = False	(Boolean) Enables or disables the creation of sparse datasets
nexenta_target_group_prefix = cinder	(String) Prefix for iSCSI target groups on NexentaStor
nexenta_target_prefix = iqn.1986-03.com.sun:02:cinder	(String) iqn prefix for NexentaStor iSCSI targets
nexenta_use_https = True	(Boolean) Use HTTP secure protocol for NexentaStor management REST API connections
nexenta_user = admin	(String) User name to connect to NexentaStor management REST API server
nexenta_volume = cinder	(String) NexentaStor pool name that holds all volumes
nexenta_volume_group = iscsi	(String) Volume group for NexentaStor5 iSCSI

- Description: Executes volume driver commands on Nexenta Appliance.

Version history:

```
1.0.0 - Initial driver version.
1.0.1 - Fixed bug #1236626: catch "does not exist" exception of
        lu_exists.
1.1.0 - Changed class name to NexentaISCSIDriver.
1.1.1 - Ignore "does not exist" exception of nms.snapshot.destroy.
```

(continues on next page)

(continued from previous page)

```
1.1.2 - Optimized create_cloned_volume, replaced zfs send recv with zfs
        clone.
1.1.3 - Extended volume stats provided by _update_volume_stats method.
1.2.0 - Added volume migration with storage assist method.
1.2.1 - Fixed bug #1263258: now migrate_volume update provider_location
        of migrated volume; after migrating volume migrate_volume
        destroy snapshot on migration destination.
1.3.0 - Added retype method.
1.3.0.1 - Target creation refactor.
1.3.1 - Added ZFS cleanup.
```

NexentaISCSIDriver

- Version: 1.4.3
- volume_driver=cinder.volume.drivers.nexenta.ns5.iscsi.NexentaISCSIDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/Nexenta_CI
- Driver Configuration Options:

Table 182: **Driver configuration options**

Name = Default Value	(Type) Description
nexenta_blocksize = 4096	(Integer) Block size for datasets
nexenta_dataset_compression = on	(String(choices=[on, off, gzip, gzip-1, gzip-2, gzip-3, gzip-4, gzip-5, gzip-6, gzip-7, gzip-8, gzip-9, lzjb, zle, lz4])) Compression value for new ZFS folders.
nexenta_dataset_dedup = off	(String(choices=[on, off, sha256, verify, sha256, verify])) Deduplication value for new ZFS folders.
nexenta_dataset_description =	(String) Human-readable description for the folder.
nexenta_folder =	(String) A folder where cinder created datasets will reside.
nexenta_group_snapshot_template = group-snapshot-%s	(String) Template string to generate group snapshot name
nexenta_host =	(String) IP address of NexentaStor Appliance
nexenta_host_group_prefix = cinder	(String) Prefix for iSCSI host groups on NexentaStor
nexenta_iscsi_target_host_group = all	(String) Group of hosts which are allowed to access volumes
nexenta_iscsi_target_portal_groups =	(String) NexentaStor target portal groups
nexenta_iscsi_target_portal_port = 3260	(Integer) Nexenta appliance iSCSI target portal port
nexenta_iscsi_target_portals =	(String) Comma separated list of portals for NexentaStor5, in format of IP1:port1,IP2:port2. Port is optional, default=3260. Example: 10.10.10.1:3267,10.10.1.2

continues on next page

Table 182 – continued from previous page

Name = Default Value	(Type) Description
nexenta_lu_writebackcache_disabled = False	(Boolean) Postponed write to backing store or not
nexenta_luns_per_target = 100	(Integer) Amount of LUNs per iSCSI target
nexenta_ns5_blocksize = 32	(Integer) Block size for datasets
nexenta_origin_snapshot_template = origin-snapshot-%s	(String) Template string to generate origin name of clone
nexenta_password = nexenta	(String) Password to connect to NexentaStor management REST API server
nexenta_rest_address =	(String) IP address of NexentaStor management REST API endpoint
nexenta_rest_backoff_factor = 0.5	(Float) Specifies the backoff factor to apply between connection attempts to NexentaStor management REST API server
nexenta_rest_connect_timeout = 30	(Float) Specifies the time limit (in seconds), within which the connection to NexentaStor management REST API server must be established
nexenta_rest_port = 0	(Integer) HTTP(S) port to connect to NexentaStor management REST API server. If it is equal zero, 8443 for HTTPS and 8080 for HTTP is used
nexenta_rest_protocol = auto	(String(choices=[http, https, auto])) Use http or https for NexentaStor management REST API connection (default auto)
nexenta_rest_read_timeout = 300	(Float) Specifies the time limit (in seconds), within which NexentaStor management REST API server must send a response
nexenta_rest_retry_count = 3	(Integer) Specifies the number of times to repeat NexentaStor management REST API call in case of connection errors and NexentaStor appliance EBUSY or ENOENT errors
nexenta_sparse = False	(Boolean) Enables or disables the creation of sparse datasets
nexenta_target_group_prefix = cinder	(String) Prefix for iSCSI target groups on NexentaStor
nexenta_target_prefix = iqn.1986-03.com.sun:02:cinder	(String) iqn prefix for NexentaStor iSCSI targets
nexenta_use_https = True	(Boolean) Use HTTP secure protocol for NexentaStor management REST API connections
nexenta_user = admin	(String) User name to connect to NexentaStor management REST API server
nexenta_volume = cinder	(String) NexentaStor pool name that holds all volumes
nexenta_volume_group = iscsi	(String) Volume group for NexentaStor5 iSCSI

- Description: Executes volume driver commands on Nexenta Appliance.

Version history:

```
1.0.0 - Initial driver version.
1.1.0 - Added HTTPS support.
      - Added use of sessions for REST calls.
      - Added abandoned volumes and snapshots cleanup.
1.2.0 - Failover support.
```

(continues on next page)

(continued from previous page)

```
1.2.1 - Configurable luns per parget, target prefix.
1.3.0 - Removed target/TG caching, added support for target portals
        and host groups.
1.3.1 - Refactored _do_export to query exact lunMapping.
1.3.2 - Revert to snapshot support.
1.3.3 - Refactored LUN creation, use host group for LUN mappings.
1.3.4 - Adapted NexentaException for the latest Cinder.
1.3.5 - Added deferred deletion for snapshots.
1.3.6 - Fixed race between volume/clone deletion.
1.3.7 - Added consistency group support.
1.3.8 - Added volume multi-attach.
1.4.0 - Refactored iSCSI driver.
      - Added pagination support.
      - Added configuration parameters for REST API connect/read
        timeouts, connection retries and backoff factor.
      - Fixed HA failover.
      - Added retries on EBUSY errors.
      - Fixed HTTP authentication.
      - Added coordination for dataset operations.
1.4.1 - Support for NexentaStor tenants.
1.4.2 - Added manage/unmanage/manageable-list volume/snapshot support.
1.4.3 - Added consistency group capability to generic volume group.
```

NexentaNfsDriver

- Version: 1.3.1
- volume_driver=cinder.volume.drivers.nexenta.nfs.NexentaNfsDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/Nexenta_CI
- Driver Configuration Options:

Table 183: **Driver configuration options**

Name = Default Value	(Type) Description
nexenta_blocksize = 4096	(Integer) Block size for datasets
nexenta_dataset_compression = on	(String(choices=[on, off, gzip, gzip-1, gzip-2, gzip-3, gzip-4, gzip-5, gzip-6, gzip-7, gzip-8, gzip-9, lzjb, zle, lz4])) Compression value for new ZFS folders.
nexenta_dataset_dedup = off	(String(choices=[on, off, sha256, verify, sha256, verify])) Deduplication value for new ZFS folders.
nexenta_dataset_description =	(String) Human-readable description for the folder.
nexenta_folder =	(String) A folder where cinder created datasets will reside.
nexenta_group_snapshot_template = group-snapshot-%s	(String) Template string to generate group snapshot name
nexenta_host =	(String) IP address of NexentaStor Appliance
nexenta_lu_writebackcache_disabl = False	(Boolean) Postponed write to backing store or not
nexenta_mount_point_base = $state_path/mnt	(String) Base directory that contains NFS share mount points
nexenta_nms_cache_volroot = True	(Boolean) If set True cache NexentaStor appliance volroot option value.
nexenta_ns5_blocksize = 32	(Integer) Block size for datasets
nexenta_origin_snapshot_template = origin-snapshot-%s	(String) Template string to generate origin name of clone
nexenta_password = nexenta	(String) Password to connect to NexentaStor management REST API server
nexenta_qcow2_volumes = False	(Boolean) Create volumes as QCOW2 files rather than raw files
nexenta_rest_address =	(String) IP address of NexentaStor management REST API endpoint
nexenta_rest_backoff_factor = 0.5	(Float) Specifies the backoff factor to apply between connection attempts to NexentaStor management REST API server
nexenta_rest_connect_timeout = 30	(Float) Specifies the time limit (in seconds), within which the connection to NexentaStor management REST API server must be established
nexenta_rest_port = 0	(Integer) HTTP(S) port to connect to NexentaStor management REST API server. If it is equal zero, 8443 for HTTPS and 8080 for HTTP is used
nexenta_rest_protocol = auto	(String(choices=[http, https, auto])) Use http or https for NexentaStor management REST API connection (default auto)
nexenta_rest_read_timeout = 300	(Float) Specifies the time limit (in seconds), within which NexentaStor management REST API server must send a response
nexenta_rest_retry_count = 3	(Integer) Specifies the number of times to repeat NexentaStor management REST API call in case of connection errors and NexentaStor appliance EBUSY or ENOENT errors
nexenta_rrmgr_compression = 0	(Integer) Enable stream compression, level 1..9. 1 - gives best speed; 9 - gives best compression.
nexenta_rrmgr_connections = 2	(Integer) Number of TCP connections.
nexenta_rrmgr_tcp_buf_size = 4096	(Integer) TCP Buffer size in KiloBytes.
nexenta_shares_config = /etc/cinder/nfs_shares	(String) File with the list of available nfs shares
nexenta_sparse = False	(Boolean) Enables or disables the creation of sparse datasets
nexenta_sparsed_volumes =	(Boolean) Enables or disables the creation of volumes as sparsed files

- Description: Executes volume driver commands on Nexenta Appliance.

Version history:

```
1.0.0 - Initial driver version.
1.1.0 - Auto sharing for enclosing folder.
1.1.1 - Added caching for NexentaStor appliance 'volroot' value.
1.1.2 - Ignore "folder does not exist" error in delete_volume and
        delete_snapshot method.
1.1.3 - Redefined volume_backend_name attribute inherited from
        RemoteFsDriver.
1.2.0 - Added migrate and retype methods.
1.3.0 - Extend volume method.
1.3.1 - Cache capacity info and check shared folders on setup.
```

NexentaNfsDriver

- Version: 1.8.3
- volume_driver=cinder.volume.drivers.nexenta.ns5.nfs.NexentaNfsDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/Nexenta_CI
- Driver Configuration Options:

Table 184: **Driver configuration options**

Name = Default Value	(Type) Description
nexenta_blocksize = 4096	(Integer) Block size for datasets
nexenta_dataset_compression = on	(String(choices=[on, off, gzip, gzip-1, gzip-2, gzip-3, gzip-4, gzip-5, gzip-6, gzip-7, gzip-8, gzip-9, lzjb, zle, lz4])) Compression value for new ZFS folders.
nexenta_dataset_dedup = off	(String(choices=[on, off, sha256, verify, sha256, verify])) Deduplication value for new ZFS folders.
nexenta_dataset_description =	(String) Human-readable description for the folder.
nexenta_folder =	(String) A folder where cinder created datasets will reside.
nexenta_group_snapshot_template = group-snapshot-%s	(String) Template string to generate group snapshot name
nexenta_host =	(String) IP address of NexentaStor Appliance
nexenta_lu_writebackcache_disabl = False	(Boolean) Postponed write to backing store or not
nexenta_mount_point_base = $state_path/mnt	(String) Base directory that contains NFS share mount points
nexenta_nms_cache_volroot = True	(Boolean) If set True cache NexentaStor appliance volroot option value.
nexenta_ns5_blocksize = 32	(Integer) Block size for datasets
nexenta_origin_snapshot_template = origin-snapshot-%s	(String) Template string to generate origin name of clone
nexenta_password = nexenta	(String) Password to connect to NexentaStor management REST API server
nexenta_qcow2_volumes = False	(Boolean) Create volumes as QCOW2 files rather than raw files
nexenta_rest_address =	(String) IP address of NexentaStor management REST API endpoint
nexenta_rest_backoff_factor = 0.5	(Float) Specifies the backoff factor to apply between connection attempts to NexentaStor management REST API server
nexenta_rest_connect_timeout = 30	(Float) Specifies the time limit (in seconds), within which the connection to NexentaStor management REST API server must be established
nexenta_rest_port = 0	(Integer) HTTP(S) port to connect to NexentaStor management REST API server. If it is equal zero, 8443 for HTTPS and 8080 for HTTP is used
nexenta_rest_protocol = auto	(String(choices=[http, https, auto])) Use http or https for NexentaStor management REST API connection (default auto)
nexenta_rest_read_timeout = 300	(Float) Specifies the time limit (in seconds), within which NexentaStor management REST API server must send a response
nexenta_rest_retry_count = 3	(Integer) Specifies the number of times to repeat NexentaStor management REST API call in case of connection errors and NexentaStor appliance EBUSY or ENOENT errors
nexenta_shares_config = /etc/cinder/nfs_shares	(String) File with the list of available nfs shares
nexenta_sparse = False	(Boolean) Enables or disables the creation of sparse datasets
nexenta_sparsed_volumes = True	(Boolean) Enables or disables the creation of volumes as sparsed files that take no space. If disabled (False), volume is created as a regular file, which takes a long time.
nexenta_use_https = True	(Boolean) Use HTTP secure protocol for NexentaStor management REST API connections
nexenta_user = admin	(String) User name to connect to NexentaStor management REST API server

- Description: Executes volume driver commands on Nexenta Appliance.

Version history:

```
1.0.0 - Initial driver version.
1.1.0 - Support for extend volume.
1.2.0 - Added HTTPS support.
      - Added use of sessions for REST calls.
      - Added abandoned volumes and snapshots cleanup.
1.3.0 - Failover support.
1.4.0 - Migrate volume support and new NEF API calls.
1.5.0 - Revert to snapshot support.
1.6.0 - Get mountPoint from API to support old style mount points.
      - Mount and umount shares on each operation to avoid mass
        mounts on controller. Clean up mount folders on delete.
1.6.1 - Fixed volume from image creation.
1.6.2 - Removed redundant share mount from initialize_connection.
1.6.3 - Adapted NexentaException for the latest Cinder.
1.6.4 - Fixed volume mount/unmount.
1.6.5 - Added driver_ssl_cert_verify for HA failover.
1.6.6 - Destroy unused snapshots after deletion of it's last clone.
1.6.7 - Fixed volume migration for HA environment.
1.6.8 - Added deferred deletion for snapshots.
1.6.9 - Fixed race between volume/clone deletion.
1.7.0 - Added consistency group support.
1.7.1 - Removed redundant hpr/activate call from initialize_connection.
1.7.2 - Merged upstream changes for umount.
1.8.0 - Refactored NFS driver.
      - Added pagination support.
      - Added configuration parameters for REST API connect/read
        timeouts, connection retries and backoff factor.
      - Fixed HA failover.
      - Added retries on EBUSY errors.
      - Fixed HTTP authentication.
      - Disabled non-blocking mandatory locks.
      - Added coordination for dataset operations.
1.8.1 - Support for NexentaStor tenants.
1.8.2 - Added manage/unmanage/manageable-list volume/snapshot support.
1.8.3 - Added consistency group capability to generic volume group.
1.8.4 - Disabled SmartCompression feature.
```

NfsDriver

- Version: 1.4.0
- volume_driver=cinder.volume.drivers.nfs.NfsDriver
- Driver Configuration Options:

Table 185: **Driver configuration options**

Name = Default Value	(Type) Description
nas_host =	(String) IP address or Hostname of NAS system.
nas_login = admin	(String) User name to connect to NAS system.
nas_mount_options = None	(String) Options used to mount the storage backend file system where Cinder volumes are stored.
nas_password =	(String) Password to connect to NAS system.
nas_private_key =	(String) Filename of private key to use for SSH authentication.
nas_secure_file_operations = auto	(String) Allow network-attached storage systems to operate in a secure environment where root level access is not permitted. If set to False, access is as the root user and insecure. If set to True, access is not as root. If set to auto, a check is done to determine if this is a new installation: True is used if so, otherwise False. Default is auto.
nas_secure_file_permissions = auto	(String) Set more secure file permissions on network-attached storage volume files to restrict broad other/world access. If set to False, volumes are created with open permissions. If set to True, volumes are created with permissions for the cinder user and group (660). If set to auto, a check is done to determine if this is a new installation: True is used if so, otherwise False. Default is auto.
nas_share_path =	(String) Path to the share to use for storing Cinder volumes. For example: /srv/export1 for an NFS server export available at 10.0.5.10:/srv/export1 .
nas_ssh_port = 22	(Port(min=0, max=65535)) SSH port to use to connect to NAS system.
nfs_mount_attempts = 3	(Integer) The number of attempts to mount NFS shares before raising an error. At least one attempt will be made to mount an NFS share, regardless of the value specified.
nfs_mount_options = None	(String) Mount options passed to the NFS client. See the NFS(5) man page for details.
nfs_mount_point_base = $state_path/mnt	(String) Base dir containing mount points for NFS shares.
nfs_qcow2_volumes = False	(Boolean) Create volumes as QCOW2 files rather than raw files.
nfs_shares_config = /etc/cinder/nfs_shares	(String) File with the list of available NFS shares.
nfs_snapshot_support = False	(Boolean) Enable support for snapshots on the NFS driver. Platforms using libvirt <1.2.7 will encounter issues with this feature.
nfs_sparsed_volumes = True	(Boolean) Create volumes as sparsed files which take no space. If set to False volume is created as regular file. In such case volume creation takes a lot of time.

- Description: NFS based cinder driver.

Creates file on NFS share for using it as block device on hypervisor.

NimbleFCDriver

- Version: 4.3.0
- volume_driver=cinder.volume.drivers.hpe.nimble.NimbleFCDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/HPE_Nimble_Storage_CI

- Driver Configuration Options:

Table 186: **Driver configuration options**

Name = Default Value	(Type) Description
max_over_subscription_ratio = 20.0	(String(regex=^(auto|\d*\.\d+|\d+)$)) Representation of the over subscription ratio when thin provisioning is enabled. Default ratio is 20.0, meaning provisioned capacity can be 20 times of the total physical capacity. If the ratio is 10.5, it means provisioned capacity can be 10.5 times of the total physical capacity. A ratio of 1.0 means provisioned capacity cannot exceed the total physical capacity. If ratio is auto, Cinder will automatically calculate the ratio based on the provisioned capacity and the used space. If not set to auto, the ratio has to be a minimum of 1.0.
nimble_pool_name = default	(String) Nimble Controller pool name
nimble_subnet_label = *	(String) Nimble Subnet Label
nimble_verify_cert_path = None	(String) Path to Nimble Array SSL certificate
nimble_verify_certificate = False	(Boolean) Whether to verify Nimble SSL Certificate

- Description: OpenStack driver to enable Nimble FC Driver Controller.

NimbleISCSIDriver

- Version: 4.3.0
- volume_driver=cinder.volume.drivers.hpe.nimble.NimbleISCSIDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/HPE_Nimble_Storage_CI
- Driver Configuration Options:

Table 187: **Driver configuration options**

Name = Default Value	(Type) Description
max_over_subscription_ratio = 20.0	(String(regex=^(auto|\d*\.\d+|\d+)$)) Representation of the over subscription ratio when thin provisioning is enabled. Default ratio is 20.0, meaning provisioned capacity can be 20 times of the total physical capacity. If the ratio is 10.5, it means provisioned capacity can be 10.5 times of the total physical capacity. A ratio of 1.0 means provisioned capacity cannot exceed the total physical capacity. If ratio is auto, Cinder will automatically calculate the ratio based on the provisioned capacity and the used space. If not set to auto, the ratio has to be a minimum of 1.0.
nimble_pool_name = default	(String) Nimble Controller pool name
nimble_subnet_label = *	(String) Nimble Subnet Label
nimble_verify_cert_path = None	(String) Path to Nimble Array SSL certificate
nimble_verify_certificate = False	(Boolean) Whether to verify Nimble SSL Certificate

- Description: OpenStack driver to enable Nimble ISCSI Controller.

PVMEFCDriver

- Version: 2.0
- volume_driver=cinder.volume.drivers.dell_emc.powervault.fc.PVMEFCDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/DellEMC_PowerVault_ME_CI
- Driver Configuration Options:

Table 188: **Driver configuration options**

Name = Default Value	(Type) Description
driver_ssl_cert_path = None	(String) Can be used to specify a non default path to a CA_BUNDLE file or directory with certificates of trusted CAs, which will be used to validate the backend
driver_ssl_cert_verify = False	(Boolean) If set to True the http client will validate the SSL certificate of the backend endpoint.
driver_use_ssl = False	(Boolean) Tell driver to use SSL for connection to backend storage if the driver supports it.
pvme_pool_name = A	(String) Pool or Vdisk name to use for volume creation.
san_ip =	(String) IP address of SAN controller
san_login = admin	(String) Username for SAN controller
san_password =	(String) Password for SAN controller

- Description: Cinder FC driver for Dell EMC PowerVault ME-Series arrays.

```
Version history:
    1.0    - Inheriting from Seagate Cinder driver.
```

PVMEISCSIDriver

- Version: 2.0
- volume_driver=cinder.volume.drivers.dell_emc.powervault.iscsi.PVMEISCSIDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/DellEMC_PowerVault_ME_CI
- Driver Configuration Options:

Table 189: **Driver configuration options**

Name = Default Value	(Type) Description
driver_ssl_cert_path = None	(String) Can be used to specify a non default path to a CA_BUNDLE file or directory with certificates of trusted CAs, which will be used to validate the backend
driver_ssl_cert_verify = False	(Boolean) If set to True the http client will validate the SSL certificate of the backend endpoint.
driver_use_ssl = False	(Boolean) Tell driver to use SSL for connection to backend storage if the driver supports it.
pvme_iscsi_ips = []	(List of String) List of comma-separated target iSCSI IP addresses.
pvme_pool_name = A	(String) Pool or Vdisk name to use for volume creation.
san_ip =	(String) IP address of SAN controller
san_login = admin	(String) Username for SAN controller
san_password =	(String) Password for SAN controller

- Description: Cinder iSCSI driver for Dell EMC PowerVault ME-Series arrays.

```
Version history:
    1.0    - Inheriting from Seagate Cinder driver.
```

PowerFlexDriver

- Version: 3.5.8
- volume_driver=cinder.volume.drivers.dell_emc.powerflex.driver.PowerFlexDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/DellEMC_PowerFlex_CI
- Driver Configuration Options:

Table 190: **Driver configuration options**

Name = Default Value	(Type) Description
powerflex_allow_migration_during_r = False	(Boolean) Allow volume migration during rebuild.
powerflex_allow_non_padded_volum = False	(Boolean) Allow volumes to be created in Storage Pools when zero padding is disabled. This option should not be enabled if multiple tenants will utilize volumes from a shared Storage Pool.
powerflex_max_over_subscription_ra = 10.0	(Float) max_over_subscription_ratio setting for the driver. Maximum value allowed is 10.0.
powerflex_rest_server_port = 443	(Port(min=0, max=65535)) Gateway REST server port.
powerflex_round_volume_capacity = True	(Boolean) Round volume sizes up to 8GB boundaries. PowerFlex/VxFlex OS requires volumes to be sized in multiples of 8GB. If set to False, volume creation will fail for volumes not sized properly
powerflex_server_api_version = None	(String) PowerFlex/ScaleIO API version. This value should be left as the default value unless otherwise instructed by technical support.
powerflex_storage_pools = None	(String) Storage Pools. Comma separated list of storage pools used to provide volumes. Each pool should be specified as a protection_domain_name:storage_pool_name value
powerflex_unmap_volume_before_de = False	(Boolean) Unmap volumes before deletion.
rest_api_connect_timeout = 30	(Integer(min=1)) Use this value to specify connect timeout value (in seconds) for rest call.
rest_api_read_timeout = 30	(Integer(min=1)) Use this value to specify read timeout value (in seconds) for rest call.
vxflexos_allow_migration_duri = False	(Boolean) renamed to powerflex_allow_migration_during_rebuild.
vxflexos_allow_non_padded_vo = False	(Boolean) renamed to powerflex_allow_non_padded_volumes.
vxflexos_max_over_subscriptio = 10.0	(Float) renamed to powerflex_max_over_subscription_ratio.
vxflexos_rest_server_port = 443	(Port(min=0, max=65535)) renamed to powerflex_rest_server_port.
vxflexos_round_volume_capaci = True	(Boolean) renamed to powerflex_round_volume_capacity.
vxflexos_server_api_version = None	(String) renamed to powerflex_server_api_version.
vxflexos_storage_pools = None	(String) renamed to powerflex_storage_pools.
vxflexos_unmap_volume_befor = False	(Boolean) renamed to powerflex_round_volume_capacity.

- Description: Cinder PowerFlex(formerly named Dell EMC VxFlex OS) Driver

```
Version history:
    2.0.1 - Added support for SIO 1.3x in addition to 2.0.x
```

(continues on next page)

(continued from previous page)

```
    2.0.2 - Added consistency group support to generic volume groups
    2.0.3 - Added cache for storage pool and protection domains info
    2.0.4 - Added compatibility with os_brick>1.15.3
    2.0.5 - Change driver name, rename config file options
    3.0.0 - Add support for VxFlex OS 3.0.x and for volumes compression
    3.5.0 - Add support for PowerFlex 3.5.x
    3.5.1 - Add volume replication v2.1 support for PowerFlex 3.5.x
    3.5.2 - Add volume migration support
    3.5.3 - Add revert volume to snapshot support
    3.5.4 - Fix for Bug #1823200. See OSSN-0086 for details.
    3.5.5 - Rebrand VxFlex OS to PowerFlex.
    3.5.6 - Fix for Bug #1897598 when volume can be migrated without
            conversion of its type.
    3.5.7 - Report trim/discard support.
    3.5.8 - Added Cinder active/active support.
```

PowerMaxFCDriver

- Version: 4.5.2
- volume_driver=cinder.volume.drivers.dell_emc.powermax.fc.PowerMaxFCDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/DellEMC_PowerMAX_CI
- Driver Configuration Options:

Table 191: **Driver configuration options**

Name = Default Value	(Type) Description
driver_ssl_cert_verify = False	(Boolean) If set to True the http client will validate the SSL certificate of the backend endpoint.
initiator_check = False	(Boolean) Use this value to enable the initiator_check.
interval = 3	(Integer) Use this value to specify length of the interval in seconds.
load_balance = False	(Boolean) Enable/disable load balancing for a PowerMax backend.
load_balance_real_time = False	(Boolean) Enable/disable real-time performance metrics for Port level load balancing for a PowerMax backend.
load_data_format = Avg	(String) Performance data format, not applicable for real-time metrics. Available options are avg and max.
load_look_back = 60	(Integer) How far in minutes to look back for diagnostic performance metrics in load calculation, minimum of 0 maximum of 1440 (24 hours).
load_look_back_real_time = 1	(Integer) How far in minutes to look back for real-time performance metrics in load calculation, minimum of 1 maximum of 10.

continues on next page

Table 191 – continued from previous page

Name = Default Value	(Type) Description
max_over_subscription_ratio = 20.0	(String(regex=^(auto|\d*\.\d+|\d+)$)) Representation of the over subscription ratio when thin provisioning is enabled. Default ratio is 20.0, meaning provisioned capacity can be 20 times of the total physical capacity. If the ratio is 10.5, it means provisioned capacity can be 10.5 times of the total physical capacity. A ratio of 1.0 means provisioned capacity cannot exceed the total physical capacity. If ratio is auto, Cinder will automatically calculate the ratio based on the provisioned capacity and the used space. If not set to auto, the ratio has to be a minimum of 1.0.
port_group_load_metric = PercentBusy	(String) Metric used for port group load calculation.
port_load_metric = PercentBusy	(String) Metric used for port load calculation.
powermax_array = None	(String) Serial number of the array to connect to.
powermax_array_tag_list = None	(List of String) List of user assigned name for storage array.
powermax_port_group_name_templa = portGroupName	(String) User defined override for port group name.
powermax_port_groups = None	(List of String) List of port groups containing frontend ports configured prior for server connection.
powermax_service_level = None	(String) Service level to use for provisioning storage. Setting this as an extra spec in pool_name is preferable.
powermax_short_host_name_templat = shortHostName	(String) User defined override for short host name.
powermax_srp = None	(String) Storage resource pool on array to use for provisioning.
replication_device = None	(Dict of String) Multi opt of dictionaries to represent a replication target device. This option may be specified multiple times in a single config section to specify multiple replication target devices. Each entry takes the standard dict config form: replication_device = target_device_id:<required>,key1:value1,key2:value2
reserved_percentage = 0	(Integer(min=0, max=100)) The percentage of backend capacity is reserved
rest_api_connect_timeout = 30	(Integer(min=1)) Use this value to specify connect timeout value (in seconds) for rest call.
rest_api_read_timeout = 30	(Integer(min=1)) Use this value to specify read timeout value (in seconds) for rest call.
retries = 200	(Integer) Use this value to specify number of retries.
san_ip =	(String) IP address of SAN controller
san_login = admin	(String) Username for SAN controller
san_password =	(String) Password for SAN controller
snapvx_unlink_symforce = False	(Boolean) Enable SnapVx unlink symforce, which forces the operation to execute when normally it is rejected.
u4p_failover_autofailback = True	(Boolean) If the driver should automatically failback to the primary instance of Unisphere when a successful connection is re-established.

continues on next page

Table 191 – continued from previous page

Name = Default Value	(Type) Description
u4p_failover_backoff_factor = 1	(Integer) A backoff factor to apply between attempts after the second try (most errors are resolved immediately by a second try without a delay). Retries will sleep for: {backoff factor} * (2 ^ ({number of total retries} - 1)) seconds.
u4p_failover_retries = 3	(Integer) The maximum number of retries each connection should attempt. Note, this applies only to failed DNS lookups, socket connections and connection timeouts, never to requests where data has made it to the server.
u4p_failover_target = None	(Dict of String) Dictionary of Unisphere failover target info.
u4p_failover_timeout = 20.0	(Integer) How long to wait for the server to send data before giving up.
vmax_workload = None	(String) Workload, setting this as an extra spec in pool_name is preferable.

- Description: FC Drivers for PowerMax using REST.

Version history:

```
1.0.0 - Initial driver
1.1.0 - Multiple pools and thick/thin provisioning,
        performance enhancement.
2.0.0 - Add driver requirement functions
2.1.0 - Add consistency group functions
2.1.1 - Fixed issue with mismatched config (bug #1442376)
2.1.2 - Clean up failed clones (bug #1440154)
2.1.3 - Fixed a problem with FAST support (bug #1435069)
2.2.0 - Add manage/unmanage
2.2.1 - Support for SE 8.0.3
2.2.2 - Update Consistency Group
2.2.3 - Pool aware scheduler(multi-pool) support
2.2.4 - Create CG from CG snapshot
2.3.0 - Name change for MV and SG for FAST (bug #1515181)
      - Fix for randomly choosing port group. (bug #1501919)
      - get_short_host_name needs to be called in find_device_number
        (bug #1520635)
      - Proper error handling for invalid SLOs (bug #1512795)
      - Extend Volume for VMAX3, SE8.1.0.3
      https://blueprints.launchpad.net/cinder/+spec/vmax3-extend-volume
      - Incorrect SG selected on an attach (#1515176)
      - Cleanup Zoning (bug #1501938)  NOTE: FC only
      - Last volume in SG fix
      - _remove_last_vol_and_delete_sg is not being called
        for VMAX3 (bug #1520549)
      - necessary updates for CG changes (#1534616)
      - Changing PercentSynced to CopyState (bug #1517103)
      - Getting iscsi ip from port in existing masking view
      - Replacement of EMCGetTargetEndpoints api (bug #1512791)
      - VMAX3 snapvx improvements (bug #1522821)
      - Operations and timeout issues (bug #1538214)
```

(continues on next page)

(continued from previous page)

```
2.4.0 - EMC VMAX - locking SG for concurrent threads (bug #1554634)
      - SnapVX licensing checks for VMAX3 (bug #1587017)
      - VMAX oversubscription Support (blueprint vmax-oversubscription)
      - QoS support (blueprint vmax-qos)
2.5.0 - Attach and detach snapshot (blueprint vmax-attach-snapshot)
      - MVs and SGs not reflecting correct protocol (bug #1640222)
      - Storage assisted volume migration via retype
        (bp vmax-volume-migration)
      - Support for compression on All Flash
      - Volume replication 2.1 (bp add-vmax-replication)
      - rename and restructure driver (bp vmax-rename-dell-emc)
3.0.0 - REST based driver
      - Retype (storage-assisted migration)
      - QoS support
      - Support for compression on All Flash
      - Support for volume replication
      - Support for live migration
      - Support for Generic Volume Group
3.1.0 - Support for replication groups (Tiramisu)
      - Deprecate backend xml configuration
      - Support for async replication (vmax-replication-enhancements)
      - Support for SRDF/Metro (vmax-replication-enhancements)
      - Support for manage/unmanage snapshots
        (vmax-manage-unmanage-snapshot)
      - Support for revert to volume snapshot
3.2.0 - Support for retyping replicated volumes (bp
        vmax-retype-replicated-volumes)
      - Support for multiattach volumes (bp vmax-allow-multi-attach)
      - Support for list manageable volumes and snapshots
        (bp/vmax-list-manage-existing)
      - Fix for SSL verification/cert application (bug #1772924)
      - Log VMAX metadata of a volume (bp vmax-metadata)
      - Fix for get-pools command (bug #1784856)
4.0.0 - Fix for initiator retrieval and short hostname unmapping
        (bugs #1783855 #1783867)
      - Fix for HyperMax OS Upgrade Bug (bug #1790141)
      - Support for failover to secondary Unisphere
        (bp/vmax-unisphere-failover)
      - Rebrand from VMAX to PowerMax(bp/vmax-powermax-rebrand)
      - Change from 84 to 90 REST endpoints (bug #1808539)
      - Fix for PowerMax OS replication settings (bug #1812685)
      - Support for storage-assisted in-use retype
        (bp/powermax-storage-assisted-inuse-retype)
4.1.0 - Changing from 90 to 91 rest endpoints
      - Support for Rapid TDEV Delete (bp powermax-tdev-deallocation)
      - PowerMax OS Metro formatted volumes fix (bug #1829876)
      - Support for Metro ODE (bp/powermax-metro-ode)
      - Removal of san_rest_port from PowerMax cinder.conf config
      - SnapVX noCopy mode enabled for all links
```

(continues on next page)

(continued from previous page)

```
      - Volume/Snapshot backed metadata inclusion
      - Debug metadata compression and service level info fix
4.2.0 - Support of Unisphere storage group and array tags
      - User defined override for short host name and port group name
        (bp powermax-user-defined-hostname-portgroup)
      - Switch to Unisphere REST API public replication endpoints
      - Support for multiple replication devices
      - Pools bug fix allowing 'None' variants (bug #1873253)
4.3.0 - Changing from 91 to 92 REST endpoints
      - Support for Port Group and Port load balancing
        (bp powermax-port-load-balance)
      - Fix to enable legacy volumes to live migrate (#1867163)
      - Use of snap id instead of generation (bp powermax-snapset-ids)
      - Support for Failover Abilities (bp/powermax-failover-abilities)
4.4.0 - Early check for status of port
4.4.1 - Report trim/discard support
4.5.0 - Add PowerMax v4 support
4.5.1 - Add active/active compliance
4.5.2 - Add 'disable_protected_snap' option
```

PowerMaxISCSIDriver

- Version: 4.5.2
- volume_driver=cinder.volume.drivers.dell_emc.powermax.iscsi.PowerMaxISCSIDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/DellEMC_PowerMAX_CI
- Driver Configuration Options:

Table 192: **Driver configuration options**

Name = Default Value	(Type) Description
chap_password =	(String) Password for specified CHAP account name.
chap_username =	(String) CHAP user name.
driver_ssl_cert_verify = False	(Boolean) If set to True the http client will validate the SSL certificate of the backend endpoint.
initiator_check = False	(Boolean) Use this value to enable the initiator_check.
interval = 3	(Integer) Use this value to specify length of the interval in seconds.
load_balance = False	(Boolean) Enable/disable load balancing for a PowerMax backend.
load_balance_real_time = False	(Boolean) Enable/disable real-time performance metrics for Port level load balancing for a PowerMax backend.
load_data_format = Avg	(String) Performance data format, not applicable for real-time metrics. Available options are avg and max.
load_look_back = 60	(Integer) How far in minutes to look back for diagnostic performance metrics in load calculation, minimum of 0 maximum of 1440 (24 hours).
load_look_back_real_time = 1	(Integer) How far in minutes to look back for real-time performance metrics in load calculation, minimum of 1 maximum of 10.

continues on next page

Table 192 – continued from previous page

Name = Default Value	(Type) Description
max_over_subscription_ratio = 20.0	(String(regex=^(auto|\d*\.\d+|\d+)$)) Representation of the over subscription ratio when thin provisioning is enabled. Default ratio is 20.0, meaning provisioned capacity can be 20 times of the total physical capacity. If the ratio is 10.5, it means provisioned capacity can be 10.5 times of the total physical capacity. A ratio of 1.0 means provisioned capacity cannot exceed the total physical capacity. If ratio is auto, Cinder will automatically calculate the ratio based on the provisioned capacity and the used space. If not set to auto, the ratio has to be a minimum of 1.0.
port_group_load_metric = PercentBusy	(String) Metric used for port group load calculation.
port_load_metric = PercentBusy	(String) Metric used for port load calculation.
powermax_array = None	(String) Serial number of the array to connect to.
powermax_array_tag_list = None	(List of String) List of user assigned name for storage array.
powermax_port_group_name_templa = portGroupName	(String) User defined override for port group name.
powermax_port_groups = None	(List of String) List of port groups containing frontend ports configured prior for server connection.
powermax_service_level = None	(String) Service level to use for provisioning storage. Setting this as an extra spec in pool_name is preferable.
powermax_short_host_name_templat = shortHostName	(String) User defined override for short host name.
powermax_srp = None	(String) Storage resource pool on array to use for provisioning.
replication_device = None	(Dict of String) Multi opt of dictionaries to represent a replication target device. This option may be specified multiple times in a single config section to specify multiple replication target devices. Each entry takes the standard dict config form: replication_device = target_device_id:<required>,key1:value1,key2:value2
reserved_percentage = 0	(Integer(min=0, max=100)) The percentage of backend capacity is reserved
rest_api_connect_timeout = 30	(Integer(min=1)) Use this value to specify connect timeout value (in seconds) for rest call.
rest_api_read_timeout = 30	(Integer(min=1)) Use this value to specify read timeout value (in seconds) for rest call.
retries = 200	(Integer) Use this value to specify number of retries.
san_ip =	(String) IP address of SAN controller
san_login = admin	(String) Username for SAN controller
san_password =	(String) Password for SAN controller
snapvx_unlink_symforce = False	(Boolean) Enable SnapVx unlink symforce, which forces the operation to execute when normally it is rejected.
u4p_failover_autofailback = True	(Boolean) If the driver should automatically failback to the primary instance of Unisphere when a successful connection is re-established.

continues on next page

Table 192 – continued from previous page

Name = Default Value	(Type) Description
u4p_failover_backoff_factor = 1	(Integer) A backoff factor to apply between attempts after the second try (most errors are resolved immediately by a second try without a delay). Retries will sleep for: {backoff factor} * (2 ^ ({number of total retries} - 1)) seconds.
u4p_failover_retries = 3	(Integer) The maximum number of retries each connection should attempt. Note, this applies only to failed DNS lookups, socket connections and connection timeouts, never to requests where data has made it to the server.
u4p_failover_target = None	(Dict of String) Dictionary of Unisphere failover target info.
u4p_failover_timeout = 20.0	(Integer) How long to wait for the server to send data before giving up.
use_chap_auth = False	(Boolean) Option to enable/disable CHAP authentication for targets.
vmax_workload = None	(String) Workload, setting this as an extra spec in pool_name is preferable.

- Description: ISCSI Drivers for PowerMax using Rest.

Version history:

```
1.0.0 - Initial driver
1.1.0 - Multiple pools and thick/thin provisioning,
        performance enhancement.
2.0.0 - Add driver requirement functions
2.1.0 - Add consistency group functions
2.1.1 - Fixed issue with mismatched config (bug #1442376)
2.1.2 - Clean up failed clones (bug #1440154)
2.1.3 - Fixed a problem with FAST support (bug #1435069)
2.2.0 - Add manage/unmanage
2.2.1 - Support for SE 8.0.3
2.2.2 - Update Consistency Group
2.2.3 - Pool aware scheduler(multi-pool) support
2.2.4 - Create CG from CG snapshot
2.3.0 - Name change for MV and SG for FAST (bug #1515181)
      - Fix for randomly choosing port group. (bug #1501919)
      - get_short_host_name needs to be called in find_device_number
        (bug #1520635)
      - Proper error handling for invalid SLOs (bug #1512795)
      - Extend Volume for VMAX3, SE8.1.0.3
      https://blueprints.launchpad.net/cinder/+spec/vmax3-extend-volume
      - Incorrect SG selected on an attach (#1515176)
      - Cleanup Zoning (bug #1501938)  NOTE: FC only
      - Last volume in SG fix
      - _remove_last_vol_and_delete_sg is not being called
        for VMAX3 (bug #1520549)
      - necessary updates for CG changes (#1534616)
      - Changing PercentSynced to CopyState (bug #1517103)
      - Getting iscsi ip from port in existing masking view
      - Replacement of EMCGetTargetEndpoints api (bug #1512791)
      - VMAX3 snapvx improvements (bug #1522821)
```

(continues on next page)

(continued from previous page)

```
      - Operations and timeout issues (bug #1538214)
2.4.0 - EMC VMAX - locking SG for concurrent threads (bug #1554634)
      - SnapVX licensing checks for VMAX3 (bug #1587017)
      - VMAX oversubscription Support (blueprint vmax-oversubscription)
      - QoS support (blueprint vmax-qos)
      - VMAX2/VMAX3 iscsi multipath support (iscsi only)
      https://blueprints.launchpad.net/cinder/+spec/vmax-iscsi-multipath
2.5.0 - Attach and detach snapshot (blueprint vmax-attach-snapshot)
      - MVs and SGs not reflecting correct protocol (bug #1640222)
      - Storage assisted volume migration via retype
        (bp vmax-volume-migration)
      - Support for compression on All Flash
      - Volume replication 2.1 (bp add-vmax-replication)
      - rename and restructure driver (bp vmax-rename-dell-emc)
3.0.0 - REST based driver
      - Retype (storage-assisted migration)
      - QoS support
      - Support for compression on All Flash
      - Support for volume replication
      - Support for live migration
      - Support for Generic Volume Group
3.1.0 - Support for replication groups (Tiramisu)
      - Deprecate backend xml configuration
      - Support for async replication (vmax-replication-enhancements)
      - Support for SRDF/Metro (vmax-replication-enhancements)
      - Support for manage/unmanage snapshots
        (vmax-manage-unmanage-snapshot)
      - Support for revert to volume snapshot
3.2.0 - Support for retyping replicated volumes (bp
        vmax-retype-replicated-volumes)
      - Support for multiattach volumes (bp vmax-allow-multi-attach)
      - Support for list manageable volumes and snapshots
        (bp/vmax-list-manage-existing)
      - Fix for SSL verification/cert application (bug #1772924)
      - Log VMAX metadata of a volume (bp vmax-metadata)
      - Fix for get-pools command (bug #1784856)
4.0.0 - Fix for initiator retrieval and short hostname unmapping
        (bugs #1783855 #1783867)
      - Fix for HyperMax OS Upgrade Bug (bug #1790141)
      - Support for failover to secondary Unisphere
        (bp/vmax-unisphere-failover)
      - Rebrand from VMAX to PowerMax(bp/vmax-powermax-rebrand)
      - Change from 84 to 90 REST endpoints (bug #1808539)
      - Fix for PowerMax OS replication settings (bug #1812685)
      - Support for storage-assisted in-use retype
        (bp/powermax-storage-assisted-inuse-retype)
4.1.0 - Changing from 90 to 91 rest endpoints
      - Support for Rapid TDEV Delete (bp powermax-tdev-deallocation)
      - PowerMax OS Metro formatted volumes fix (bug #1829876)
```

(continues on next page)

(continued from previous page)

```
      - Support for Metro ODE (bp/powermax-metro-ode)
      - Removal of san_rest_port from PowerMax cinder.conf config
      - SnapVX noCopy mode enabled for all links
      - Volume/Snapshot backed metadata inclusion
      - Debug metadata compression and service level info fix
4.2.0 - Support of Unisphere storage group and array tags
      - User defined override for short host name and port group name
        (bp powermax-user-defined-hostname-portgroup)
      - Switch to Unisphere REST API public replication endpoints
      - Support for multiple replication devices
      - Pools bug fix allowing 'None' variants (bug #1873253)
4.3.0 - Changing from 91 to 92 REST endpoints
      - Support for Port Group and Port load balancing
        (bp powermax-port-load-balance)
      - Fix to enable legacy volumes to live migrate (#1867163)
      - Use of snap id instead of generation (bp powermax-snapset-ids)
      - Support for Failover Abilities (bp/powermax-failover-abilities)
4.4.0 - Early check for status of port
4.4.1 - Report trim/discard support
4.5.0 - Add PowerMax v4 support
4.5.1 - Add active/active compliance
4.5.2 - Add 'disable_protected_snap' option
```

PowerStoreDriver

- Version: 1.2.3
- volume_driver=cinder.volume.drivers.dell_emc.powerstore.driver.PowerStoreDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/DellEMC_PowerStore_CI
- Driver Configuration Options:

Table 193: **Driver configuration options**

Name = Default Value	(Type) Description
powerstore_appliances = []	(List of String) Appliances names. Comma separated list of PowerStore appliances names used to provision volumes.
powerstore_nvme = False	(Boolean) Connect PowerStore volumes using NVMe-OF.
powerstore_ports = []	(List of String) Allowed ports. Comma separated list of PowerStore iSCSI IPs or FC WWNs (ex. 58:cc:f0:98:49:22:07:02) to be used. If option is not set all ports are allowed.
rest_api_call_connect_timeout = 30	(Integer(min=1)) Use this value to specify the connect timeout value (in seconds) for REST API calls to the PowerStore backend.
rest_api_call_read_timeout = 30	(Integer(min=1)) Use this value to specify the read timeout value (in seconds) for REST API calls to the PowerStore backend.

- Description: Dell EMC PowerStore Driver.

```
Version history:
  1.0.0 - Initial version
```

(continues on next page)

(continued from previous page)

```
1.0.1 - Add CHAP support
1.1.0 - Add volume replication v2.1 support
1.1.1 - Add Consistency Groups support
1.1.2 - Fix iSCSI targets not being returned from the REST API call if
        targets are used for multiple purposes
        (iSCSI target, Replication target, etc.)
1.2.0 - Add NVMe-OF support
1.2.1 - Report trim/discard support
1.2.2 - QoS (Quality of Service) support
1.2.3 - Added Cinder active/active support
```

PowerStoreNFSDriver

- Version: 1.0.0
- volume_driver=cinder.volume.drivers.dell_emc.powerstore.nfs.PowerStoreNFSDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/DellEMC_PowerStore_CI
- Driver Configuration Options:

Table 194: **Driver configuration options**

Name = Default Value	(Type) Description
nas_host =	(String) IP address or Hostname of NAS system.
nas_login = admin	(String) User name to connect to NAS system.
nas_mount_options = None	(String) Options used to mount the storage backend file system where Cinder volumes are stored.
nas_password =	(String) Password to connect to NAS system.
nas_private_key =	(String) Filename of private key to use for SSH authentication.
nas_secure_file_operations = auto	(String) Allow network-attached storage systems to operate in a secure environment where root level access is not permitted. If set to False, access is as the root user and insecure. If set to True, access is not as root. If set to auto, a check is done to determine if this is a new installation: True is used if so, otherwise False. Default is auto.
nas_secure_file_permissions = auto	(String) Set more secure file permissions on network-attached storage volume files to restrict broad other/world access. If set to False, volumes are created with open permissions. If set to True, volumes are created with permissions for the cinder user and group (660). If set to auto, a check is done to determine if this is a new installation: True is used if so, otherwise False. Default is auto.
nas_share_path =	(String) Path to the share to use for storing Cinder volumes. For example: /srv/export1 for an NFS server export available at 10.0.5.10:/srv/export1 .
nas_ssh_port = 22	(Port(min=0, max=65535)) SSH port to use to connect to NAS system.
nfs_mount_attempts = 3	(Integer) The number of attempts to mount NFS shares before raising an error. At least one attempt will be made to mount an NFS share, regardless of the value specified.
nfs_mount_options = None	(String) Mount options passed to the NFS client. See the NFS(5) man page for details.
nfs_mount_point_base = $state_path/mnt	(String) Base dir containing mount points for NFS shares.
nfs_qcow2_volumes = False	(Boolean) Create volumes as QCOW2 files rather than raw files.
nfs_shares_config = /etc/cinder/nfs_shares	(String) File with the list of available NFS shares.
nfs_snapshot_support = False	(Boolean) Enable support for snapshots on the NFS driver. Platforms using libvirt <1.2.7 will encounter issues with this feature.
nfs_sparsed_volumes = True	(Boolean) Create volumes as sparsed files which take no space. If set to False volume is created as regular file. In such case volume creation takes a lot of time.

- Description: Dell PowerStore NFS Driver.

```
Version history:
  1.0.0 - Initial version
```

PureFCDriver

- Version: 20.0.fc
- volume_driver=cinder.volume.drivers.pure.PureFCDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/Pure_Storage_CI
- Driver Configuration Options:

Table 195: **Driver configuration options**

Name = Default Value	(Type) Description
driver_ssl_cert_path = None	(String) Can be used to specify a non default path to a CA_BUNDLE file or directory with certificates of trusted CAs, which will be used to validate the backend
driver_ssl_cert_verify = False	(Boolean) If set to True the http client will validate the SSL certificate of the backend endpoint.
max_over_subscription_ratio = 20.0	(String(regex=^(auto|\d*\.\d+|\d+)$)) Representation of the over subscription ratio when thin provisioning is enabled. Default ratio is 20.0, meaning provisioned capacity can be 20 times of the total physical capacity. If the ratio is 10.5, it means provisioned capacity can be 10.5 times of the total physical capacity. A ratio of 1.0 means provisioned capacity cannot exceed the total physical capacity. If ratio is auto, Cinder will automatically calculate the ratio based on the provisioned capacity and the used space. If not set to auto, the ratio has to be a minimum of 1.0.
pure_api_token = None	(String) REST API authorization token.
pure_automatic_max_oversubsc = True	(Boolean) Automatically determine an oversubscription ratio based on the current total data reduction values. If used this calculated value will override the max_over_subscription_ratio config option.
pure_eradicate_on_delete = False	(Boolean) When enabled, all Pure volumes, snapshots, and protection groups will be eradicated at the time of deletion in Cinder. Data will NOT be recoverable after a delete with this set to True! When disabled, volumes and snapshots will go into pending eradication state and can be recovered.
pure_host_personality = None	(String(choices=[aix, esxi, hitachi-vsp, hpux, oracle-vm-server, solaris, vms, None])) Determines how the Purity system tunes the protocol used between the array and the initiator.
pure_iscsi_cidr = 0.0.0.0/0	(String) CIDR of FlashArray iSCSI targets hosts are allowed to connect to. Default will allow connection to any IPv4 address. This parameter now supports IPv6 subnets. Ignored when pure_iscsi_cidr_list is set.
pure_iscsi_cidr_list = None	(List of String) Comma-separated list of CIDR of FlashArray iSCSI targets hosts are allowed to connect to. It supports IPv4 and IPv6 subnets. This parameter supersedes pure_iscsi_cidr.
pure_nvme_cidr = 0.0.0.0/0	(String) CIDR of FlashArray NVMe targets hosts are allowed to connect to. Default will allow connection to any IPv4 address. This parameter now supports IPv6 subnets. Ignored when pure_nvme_cidr_list is set.
pure_nvme_cidr = 0.0.0.0/0	(String) CIDR of FlashArray NVMe targets hosts are allowed to connect to. Default will allow connection to any IPv4 address. This parameter now supports IPv6 subnets. Ignored when pure_nvme_cidr_list is set.
pure_nvme_cidr_list = None	(List of String) Comma-separated list of CIDR of FlashArray NVMe targets hosts are allowed to connect to. It supports IPv4 and IPv6 subnets. This parameter supersedes pure_nvme_cidr.
pure_nvme_cidr_list = None	(List of String) Comma-separated list of CIDR of FlashArray NVMe targets hosts are allowed to connect to. It supports IPv4 and IPv6 subnets. This parameter supersedes pure_nvme_cidr.
pure_nvme_transport = roce	(String(choices=[roce, tcp])) The NVMe transport layer to be used by the NVMe driver.
pure_nvme_transport = roce	(String(choices=[roce, tcp])) The NVMe transport layer to be used by the NVMe driver.
pure_replica_interval_default = 3600	(Integer) Snapshot replication interval in seconds.

- Description: OpenStack Volume Driver to support Pure Storage FlashArray.

This version of the driver enables the use of Fibre Channel for the underlying storage connectivity with the FlashArray. It fully supports the Cinder Fibre Channel Zone Manager.

PureISCSIDriver

- Version: 20.0.iscsi
- volume_driver=cinder.volume.drivers.pure.PureISCSIDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/Pure_Storage_CI
- Driver Configuration Options:

Table 196: **Driver configuration options**

Name = Default Value	(Type) Description
driver_ssl_cert_path = None	(String) Can be used to specify a non default path to a CA_BUNDLE file or directory with certificates of trusted CAs, which will be used to validate the backend
driver_ssl_cert_verify = False	(Boolean) If set to True the http client will validate the SSL certificate of the backend endpoint.
max_over_subscription_ratio = 20.0	(String(regex=^(auto|\d*\.\d+|\d+)$)) Representation of the over subscription ratio when thin provisioning is enabled. Default ratio is 20.0, meaning provisioned capacity can be 20 times of the total physical capacity. If the ratio is 10.5, it means provisioned capacity can be 10.5 times of the total physical capacity. A ratio of 1.0 means provisioned capacity cannot exceed the total physical capacity. If ratio is auto, Cinder will automatically calculate the ratio based on the provisioned capacity and the used space. If not set to auto, the ratio has to be a minimum of 1.0.
pure_api_token = None	(String) REST API authorization token.
pure_automatic_max_oversubsc = True	(Boolean) Automatically determine an oversubscription ratio based on the current total data reduction values. If used this calculated value will override the max_over_subscription_ratio config option.
pure_eradicate_on_delete = False	(Boolean) When enabled, all Pure volumes, snapshots, and protection groups will be eradicated at the time of deletion in Cinder. Data will NOT be recoverable after a delete with this set to True! When disabled, volumes and snapshots will go into pending eradication state and can be recovered.
pure_host_personality = None	(String(choices=[aix, esxi, hitachi-vsp, hpux, oracle-vm-server, solaris, vms, None])) Determines how the Purity system tunes the protocol used between the array and the initiator.
pure_iscsi_cidr = 0.0.0.0/0	(String) CIDR of FlashArray iSCSI targets hosts are allowed to connect to. Default will allow connection to any IPv4 address. This parameter now supports IPv6 subnets. Ignored when pure_iscsi_cidr_list is set.
pure_iscsi_cidr_list = None	(List of String) Comma-separated list of CIDR of FlashArray iSCSI targets hosts are allowed to connect to. It supports IPv4 and IPv6 subnets. This parameter supersedes pure_iscsi_cidr.
pure_nvme_cidr = 0.0.0.0/0	(String) CIDR of FlashArray NVMe targets hosts are allowed to connect to. Default will allow connection to any IPv4 address. This parameter now supports IPv6 subnets. Ignored when pure_nvme_cidr_list is set.
pure_nvme_cidr = 0.0.0.0/0	(String) CIDR of FlashArray NVMe targets hosts are allowed to connect to. Default will allow connection to any IPv4 address. This parameter now supports IPv6 subnets. Ignored when pure_nvme_cidr_list is set.
pure_nvme_cidr_list = None	(List of String) Comma-separated list of CIDR of FlashArray NVMe targets hosts are allowed to connect to. It supports IPv4 and IPv6 subnets. This parameter supersedes pure_nvme_cidr.
pure_nvme_cidr_list = None	(List of String) Comma-separated list of CIDR of FlashArray NVMe targets hosts are allowed to connect to. It supports IPv4 and IPv6 subnets. This parameter supersedes pure_nvme_cidr.
pure_nvme_transport = roce	(String(choices=[roce, tcp])) The NVMe transport layer to be used by the NVMe driver.
pure_nvme_transport = roce	(String(choices=[roce, tcp])) The NVMe transport layer to be used by the NVMe driver.
pure_replica_interval_default = 3600	(Integer) Snapshot replication interval in seconds.

- Description: OpenStack Volume Driver to support Pure Storage FlashArray.

This version of the driver enables the use of iSCSI for the underlying storage connectivity with the FlashArray.

PureNVMEDriver

- Version: 20.0.nvme
- volume_driver=cinder.volume.drivers.pure.PureNVMEDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/Pure_Storage_CI
- Driver Configuration Options:

Table 197: **Driver configuration options**

Name = Default Value	(Type) Description
driver_ssl_cert_path = None	(String) Can be used to specify a non default path to a CA_BUNDLE file or directory with certificates of trusted CAs, which will be used to validate the backend
driver_ssl_cert_verify = False	(Boolean) If set to True the http client will validate the SSL certificate of the backend endpoint.
max_over_subscription_ratio = 20.0	(String(regex=^(auto|\d*\.\d+|\d+)$)) Representation of the over subscription ratio when thin provisioning is enabled. Default ratio is 20.0, meaning provisioned capacity can be 20 times of the total physical capacity. If the ratio is 10.5, it means provisioned capacity can be 10.5 times of the total physical capacity. A ratio of 1.0 means provisioned capacity cannot exceed the total physical capacity. If ratio is auto, Cinder will automatically calculate the ratio based on the provisioned capacity and the used space. If not set to auto, the ratio has to be a minimum of 1.0.
pure_api_token = None	(String) REST API authorization token.
pure_automatic_max_oversubsc = True	(Boolean) Automatically determine an oversubscription ratio based on the current total data reduction values. If used this calculated value will override the max_over_subscription_ratio config option.
pure_eradicate_on_delete = False	(Boolean) When enabled, all Pure volumes, snapshots, and protection groups will be eradicated at the time of deletion in Cinder. Data will NOT be recoverable after a delete with this set to True! When disabled, volumes and snapshots will go into pending eradication state and can be recovered.
pure_host_personality = None	(String(choices=[aix, esxi, hitachi-vsp, hpux, oracle-vm-server, solaris, vms, None])) Determines how the Purity system tunes the protocol used between the array and the initiator.
pure_iscsi_cidr = 0.0.0.0/0	(String) CIDR of FlashArray iSCSI targets hosts are allowed to connect to. Default will allow connection to any IPv4 address. This parameter now supports IPv6 subnets. Ignored when pure_iscsi_cidr_list is set.
pure_iscsi_cidr_list = None	(List of String) Comma-separated list of CIDR of FlashArray iSCSI targets hosts are allowed to connect to. It supports IPv4 and IPv6 subnets. This parameter supersedes pure_iscsi_cidr.
pure_nvme_cidr = 0.0.0.0/0	(String) CIDR of FlashArray NVMe targets hosts are allowed to connect to. Default will allow connection to any IPv4 address. This parameter now supports IPv6 subnets. Ignored when pure_nvme_cidr_list is set.
pure_nvme_cidr = 0.0.0.0/0	(String) CIDR of FlashArray NVMe targets hosts are allowed to connect to. Default will allow connection to any IPv4 address. This parameter now supports IPv6 subnets. Ignored when pure_nvme_cidr_list is set.
pure_nvme_cidr_list = None	(List of String) Comma-separated list of CIDR of FlashArray NVMe targets hosts are allowed to connect to. It supports IPv4 and IPv6 subnets. This parameter supersedes pure_nvme_cidr.
pure_nvme_cidr_list = None	(List of String) Comma-separated list of CIDR of FlashArray NVMe targets hosts are allowed to connect to. It supports IPv4 and IPv6 subnets. This parameter supersedes pure_nvme_cidr.
pure_nvme_transport = roce	(String(choices=[roce, tcp])) The NVMe transport layer to be used by the NVMe driver.
pure_nvme_transport = roce	(String(choices=[roce, tcp])) The NVMe transport layer to be used by the NVMe driver.
pure_replica_interval_default = 3600	(Integer) Snapshot replication interval in seconds.

- Description: OpenStack Volume Driver to support Pure Storage FlashArray.

This version of the driver enables the use of NVMe over different transport types for the underlying storage connectivity with the FlashArray.

RBDDriver

- Version: 1.3.0
- volume_driver=cinder.volume.drivers.rbd.RBDDriver
- Driver Configuration Options:

Table 198: **Driver configuration options**

Name = Default Value	(Type) Description
deferred_deletion_delay = 0	(Integer) Time delay in seconds before a volume is eligible for permanent removal after being tagged for deferred deletion.
deferred_deletion_purge_interval = 60	(Integer) Number of seconds between runs of the periodic task to purge volumes tagged for deletion.
enable_deferred_deletion = False	(Boolean) Enable deferred deletion. Upon deletion, volumes are tagged for deletion but will only be removed asynchronously at a later time.
max_over_subscription_ratio = 20.0	(String(regex=^(auto|\d*\.\d+|\d+)$)) Representation of the over subscription ratio when thin provisioning is enabled. Default ratio is 20.0, meaning provisioned capacity can be 20 times of the total physical capacity. If the ratio is 10.5, it means provisioned capacity can be 10.5 times of the total physical capacity. A ratio of 1.0 means provisioned capacity cannot exceed the total physical capacity. If ratio is auto, Cinder will automatically calculate the ratio based on the provisioned capacity and the used space. If not set to auto, the ratio has to be a minimum of 1.0.
rados_connect_timeout = -1	(Integer) Timeout value (in seconds) used when connecting to ceph cluster. If value < 0, no timeout is set and default librados value is used.
rados_connection_interval = 5	(Integer) Interval value (in seconds) between connection retries to ceph cluster.
rados_connection_retries = 3	(Integer) Number of retries if connection to ceph cluster failed.
rbd_ceph_conf =	(String) Path to the ceph configuration file
rbd_cluster_name = ceph	(String) The name of ceph cluster
rbd_concurrent_flatten_operatic = 3	(Integer(min=0)) Number of flatten operations that will run concurrently on this volume service.
rbd_exclusive_cinder_pool = True	(Boolean) Set to False if the pool is shared with other usages. On exclusive use driver wont query images provisioned size as they will match the value calculated by the Cinder core code for allocated_capacity_gb. This reduces the load on the Ceph cluster as well as on the volume service. On non exclusive use driver will query the Ceph cluster for per image used disk, this is an intensive operation having an independent request for each image.
rbd_flatten_volume_from_snap = False	(Boolean) Flatten volumes created from snapshots to remove dependency from volume to snapshot
rbd_max_clone_depth = 5	(Integer) Maximum number of nested volume clones that are taken before a flatten occurs. Set to 0 to disable cloning. Note: lowering this value will not affect existing volumes whose clone depth exceeds the new value.
rbd_pool = rbd	(String) The RADOS pool where RBD volumes are stored
rbd_secret_uuid = None	(String) The libvirt uuid of the secret for the rbd_user volumes. Defaults to the cluster FSID.
rbd_store_chunk_size = 4	(Integer) Volumes will be chunked into objects of this size (in megabytes).
rbd_user = None	(String) The RADOS client name for accessing RBD volumes - only set when using cephx authentication
replication_connect_timeout = 5	(Integer) Timeout value (in seconds) used when connecting to ceph cluster to do a demotion/promotion of volumes. If value < 0, no timeout is set and default librados value is used.
replication_device = None	(Dict of String) Multi opt of dictionaries to represent a replication target device. This option may be specified multiple times in a single config section to specify multiple replication target devices. Each

- Description: Implements RADOS block device (RBD) volume commands.

Version history:

```
1.3.0 - Added QoS Support
```

RBDISCSIDriver

- Version: 1.0.0
- volume_driver=cinder.volume.drivers.ceph.rbd_iscsi.RBDISCSIDriver
- Driver Configuration Options:

Table 199: **Driver configuration options**

Name = Default Value	(Type) Description
deferred_deletion_delay = 0	(Integer) Time delay in seconds before a volume is eligible for permanent removal after being tagged for deferred deletion.
deferred_deletion_purge_interval = 60	(Integer) Number of seconds between runs of the periodic task to purge volumes tagged for deletion.
driver_ssl_cert_verify = False	(Boolean) If set to True the http client will validate the SSL certificate of the backend endpoint.
enable_deferred_deletion = False	(Boolean) Enable deferred deletion. Upon deletion, volumes are tagged for deletion but will only be removed asynchronously at a later time.
max_over_subscription_ratio = 20.0	(String(regex=^(auto|\d*\.\d+|\d+)$)) Representation of the over subscription ratio when thin provisioning is enabled. Default ratio is 20.0, meaning provisioned capacity can be 20 times of the total physical capacity. If the ratio is 10.5, it means provisioned capacity can be 10.5 times of the total physical capacity. A ratio of 1.0 means provisioned capacity cannot exceed the total physical capacity. If ratio is auto, Cinder will automatically calculate the ratio based on the provisioned capacity and the used space. If not set to auto, the ratio has to be a minimum of 1.0.
rados_connect_timeout = -1	(Integer) Timeout value (in seconds) used when connecting to ceph cluster. If value < 0, no timeout is set and default librados value is used.
rados_connection_interval = 5	(Integer) Interval value (in seconds) between connection retries to ceph cluster.
rados_connection_retries = 3	(Integer) Number of retries if connection to ceph cluster failed.
rbd_ceph_conf =	(String) Path to the ceph configuration file
rbd_cluster_name = ceph	(String) The name of ceph cluster
rbd_concurrent_flatten_operatio = 3	(Integer(min=0)) Number of flatten operations that will run concurrently on this volume service.
rbd_exclusive_cinder_pool = True	(Boolean) Set to False if the pool is shared with other usages. On exclusive use driver wont query images provisioned size as they will match the value calculated by the Cinder core code for allocated_capacity_gb. This reduces the load on the Ceph cluster as well as on the volume service. On non exclusive use driver will query the Ceph cluster for per image used disk, this is an intensive operation having an independent request for each image.
rbd_flatten_volume_from_snap = False	(Boolean) Flatten volumes created from snapshots to remove dependency from volume to snapshot
rbd_iscsi_api_debug = False	(Boolean) Enable client request debugging.
rbd_iscsi_api_password =	(String) The username for the rbd_target_api service
rbd_iscsi_api_url =	(String) The url to the rbd_target_api service
rbd_iscsi_api_user =	(String) The username for the rbd_target_api service
rbd_iscsi_target_iqn = None	(String) The preconfigured target_iqn on the iscsi gateway.
rbd_max_clone_depth = 5	(Integer) Maximum number of nested volume clones that are taken before a flatten occurs. Set to 0 to disable cloning. Note: lowering this value will not affect existing volumes whose clone depth exceeds the new value.
rbd_pool = rbd	(String) The RADOS pool where RBD volumes are stored
rbd_secret_uuid = None	(String) The libvirt uuid of the secret for the rbd_user volumes. Defaults to the cluster FSID.
rbd_store_chunk_size = 4	(Integer) Volumes will be chunked into objects of this size (in megabytes).
rbd_user = None	(String) The RADOS client name for accessing RBD volumes - only

- Description: Implements RADOS block device (RBD) iSCSI volume commands.

RSDDriver

- Version: 1.0.0
- volume_driver=cinder.volume.drivers.rsd.RSDDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/INTEL-RSD-CI
- Driver Configuration Options:

Table 200: **Driver configuration options**

Name = Default Value	(Type) Description
podm_password =	(String) Password of PODM service
podm_url =	(String) URL of PODM service
podm_username =	(String) Username of PODM service

- Description: Openstack driver to perform NVMe-oF volume management in RSD Solution

Version History:
1.0.0: Initial driver

SPDKDriver

- Version: 1.0.0
- volume_driver=cinder.volume.drivers.spdk.SPDKDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/Mellanox_CI
- Description: Executes commands relating to Volumes.

SdsISCSIDriver

- Version: 1.0.0
- volume_driver=cinder.volume.drivers.sandstone.sds_driver.SdsISCSIDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/SandStone_Storage_CI
- Driver Configuration Options:

Table 201: **Driver configuration options**

Name = Default Value	(Type) Description
backend_availability_zone = None	(String) Availability zone for this volume backend. If not set, the storage_availability_zone option value is used as the default for all backends.
chap_password =	(String) Password for specified CHAP account name.
chap_username =	(String) CHAP user name.

continues on next page

Table 201 – continued from previous page

Name = Default Value	(Type) Description
chiscsi_conf = /etc/chelsio-iscsi/chiscsi.conf	(String) Chiscsi (CXT) global defaults configuration file
driver_client_cert = None	(String) The path to the client certificate for verification, if the driver supports it.
driver_client_cert_key = None	(String) The path to the client certificate key for verification, if the driver supports it.
driver_data_namespace = None	(String) Namespace for driver private data values to be saved in.
driver_ssl_cert_path = None	(String) Can be used to specify a non default path to a CA_BUNDLE file or directory with certificates of trusted CAs, which will be used to validate the backend
driver_ssl_cert_verify = False	(Boolean) If set to True the http client will validate the SSL certificate of the backend endpoint.
driver_use_ssl = False	(Boolean) Tell driver to use SSL for connection to backend storage if the driver supports it.
enable_unsupported_driver = False	(Boolean) Set this to True when you want to allow an unsupported driver to start. Drivers that havent maintained a working CI system and testing are marked as unsupported until CI is working again. This also marks a driver as deprecated and may be removed in the next release.
filter_function = None	(String) String representation for an equation that will be used to filter hosts. Only used when the driver filter is set to be used by the Cinder scheduler.
goodness_function = None	(String) String representation for an equation that will be used to determine the goodness of a host. Only used when using the goodness weigher is set to be used by the Cinder scheduler.
iscsi_iotype = fileio	(String(choices=[blockio, fileio, auto])) Sets the behavior of the iSCSI target to either perform blockio or fileio optionally, auto can be set and Cinder will autodetect type of backing device
iscsi_target_flags =	(String) Sets the target-specific flags for the iSCSI target. Only used for tgtadm to specify backing device flags using bsoflags option. The specified string is passed as is to the underlying tool.
iscsi_write_cache = on	(String(choices=[on, off])) Sets the behavior of the iSCSI target to either perform write-back(on) or write-through(off). This parameter is valid if target_helper is set to tgtadm.
max_over_subscription_ratio = 20.0	(String(regex=^(auto|\d*\.\d+|\d+)$)) Representation of the over subscription ratio when thin provisioning is enabled. Default ratio is 20.0, meaning provisioned capacity can be 20 times of the total physical capacity. If the ratio is 10.5, it means provisioned capacity can be 10.5 times of the total physical capacity. A ratio of 1.0 means provisioned capacity cannot exceed the total physical capacity. If ratio is auto, Cinder will automatically calculate the ratio based on the provisioned capacity and the used space. If not set to auto, the ratio has to be a minimum of 1.0.
num_shell_tries = 3	(Integer) Number of times to attempt to run flakey shell commands
num_volume_device_scan_tries = 3	(Integer) The maximum number of times to rescan targets to find volume

continues on next page

Table 201 – continued from previous page

Name = Default Value	(Type) Description
replication_device = None	(Dict of String) Multi opt of dictionaries to represent a replication target device. This option may be specified multiple times in a single config section to specify multiple replication target devices. Each entry takes the standard dict config form: replication_device = target_device_id:<required>,key1:value1,key2:value2
report_discard_supported = False	(Boolean) Report to clients of Cinder that the backend supports discard (aka. trim/unmap). This will not actually change the behavior of the backend or the client directly, it will only notify that it can be used.
reserved_percentage = 0	(Integer(min=0, max=100)) The percentage of backend capacity is reserved
storage_protocol = iSCSI	(String(choices=[iSCSI, FC])) Protocol for transferring data between host and storage back-end.
target_helper = tgtadm	(String(choices=[tgtadm, lioadm, scstadmin, iscsictl, nvmet, spdk-nvmeof, fake])) Target user-land tool to use. tgtadm is default, use lioadm for LIO iSCSI support, scstadmin for SCST target support, iscsictl for Chelsio iSCSI Target, nvmet for NVMEoF support, spdk-nvmeof for SPDK NVMe-oF, or fake for testing.
target_ip_address = $my_ip	(String) The IP address that the iSCSI/NVMEoF daemon is listening on
target_port = 3260	(Port(min=0, max=65535)) The port that the iSCSI/NVMEoF daemon is listening on
target_prefix = iqn.2010-10.org.openstack:	(String) Prefix for iSCSI/NVMEoF volumes
target_protocol = iscsi	(String(choices=[iscsi, iser, nvmet_rdma, nvmet_tcp])) Determines the target protocol for new volumes, created with tgtadm, lioadm and nvmet target helpers. In order to enable RDMA, this parameter should be set with the value iser. The supported iSCSI protocol values are iscsi and iser, in case of nvmet target set to nvmet_rdma or nvmet_tcp.
target_secondary_ip_addresses = []	(List of String) The list of secondary IP addresses of the iSCSI/NVMEoF daemon
trace_flags = None	(List of String) List of options that control which trace info is written to the DEBUG log level to assist developers. Valid values are method and api.
use_chap_auth = False	(Boolean) Option to enable/disable CHAP authentication for targets.
volume_backend_name = None	(String) The backend name for a given driver implementation
volume_clear = zero	(String(choices=[none, zero])) Method used to wipe old volumes
volume_clear_ionice = None	(String) The flag to pass to ionice to alter the i/o priority of the process used to zero a volume after deletion, for example -c3 for idle only priority.
volume_clear_size = 0	(Integer(max=1024)) Size in MiB to wipe at start of old volumes. 1024 MiB at max. 0 => all
volume_copy_blkio_cgroup_name = cinder-volume-copy	(String) The blkio cgroup name to be used to limit bandwidth of volume copy

continues on next page

Table 201 – continued from previous page

Name = Default Value	(Type) Description
volume_copy_bps_limit = 0	(Integer) The upper limit of bandwidth of volume copy. 0 => unlimited
volume_dd_blocksize = 1M	(String) The default block size used when copying/clearing volumes
volumes_dir = $state_path/volumes	(String) Volume configuration file storage directory

- Description: ISCSI driver for SandStone storage arrays.

Version history:

```
1.0.0 - Initial driver
        Provide SandStone storage
        create volume support
        delete volume support
        create snapshot support
        delete snapshot support
        extend volume support
        create volume from snap support
        create cloned volume support
        nova volume-attach support
        nova volume-detach support
```

SolidFireDriver

- Version: 2.2.5
- volume_driver=cinder.volume.drivers.solidfire.SolidFireDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/NetApp_SolidFire_CI
- Driver Configuration Options:

Table 202: **Driver configuration options**

Name = Default Value	(Type) Description
driver_ssl_cert_verify = False	(Boolean) If set to True the http client will validate the SSL certificate of the backend endpoint.
max_over_subscription_ratio = 20.0	(String(regex=^(auto|\d*\.\d+|\d+)$)) Representation of the over subscription ratio when thin provisioning is enabled. Default ratio is 20.0, meaning provisioned capacity can be 20 times of the total physical capacity. If the ratio is 10.5, it means provisioned capacity can be 10.5 times of the total physical capacity. A ratio of 1.0 means provisioned capacity cannot exceed the total physical capacity. If ratio is auto, Cinder will automatically calculate the ratio based on the provisioned capacity and the used space. If not set to auto, the ratio has to be a minimum of 1.0.
replication_device = None	(Dict of String) Multi opt of dictionaries to represent a replication target device. This option may be specified multiple times in a single config section to specify multiple replication target devices. Each entry takes the standard dict config form: replication_device = target_device_id:<required>,key1:value1,key2:value2
reserved_percentage = 0	(Integer(min=0, max=100)) The percentage of backend capacity is reserved
san_ip =	(String) IP address of SAN controller
san_login = admin	(String) Username for SAN controller
san_password =	(String) Password for SAN controller
sf_account_prefix = None	(String) Create SolidFire accounts with this prefix. Any string can be used here, but the string hostname is special and will create a prefix using the cinder node hostname (previous default behavior). The default is NO prefix.
sf_allow_tenant_qos = False	(Boolean) Allow tenants to specify QOS on create
sf_api_port = 443	(Port(min=0, max=65535)) SolidFire API port. Useful if the device api is behind a proxy on a different port.
sf_api_request_timeout = 30	(Integer(min=30)) Sets time in seconds to wait for an api request to complete.
sf_cluster_pairing_timeout = 60	(Integer(min=3)) Sets time in seconds to wait for clusters to complete pairing.
sf_emulate_512 = True	(Boolean) Set 512 byte emulation on volume creation;
sf_enable_vag = False	(Boolean) Utilize volume access groups on a per-tenant basis.
sf_provisioning_calc = maxProvisionedSpace	(String(choices=[maxProvisionedSpace, usedSpace])) Change how SolidFire reports used space and provisioning calculations. If this parameter is set to usedSpace, the driver will report correct values as expected by Cinder thin provisioning.
sf_svip = None	(String) Overrides default cluster SVIP with the one specified. This is required or deployments that have implemented the use of VLANs for iSCSI networks in their cloud.
sf_volume_clone_timeout = 600	(Integer(min=60)) Sets time in seconds to wait for a clone of a volume or snapshot to complete.
sf_volume_create_timeout = 60	(Integer(min=30)) Sets time in seconds to wait for a create volume operation to complete.
sf_volume_pairing_timeout = 3600	(Integer(min=30)) Sets time in seconds to wait for a migrating volume to complete pairing and sync.
sf_volume_prefix = UUID-	(String) Create SolidFire volumes with this prefix. Volume names are of the form <sf_volume_prefix><cinder-volume-id>. The default is to use a prefix of UUID-.

- Description: OpenStack driver to enable SolidFire cluster.

```
Version history:
    1.0 - Initial driver
    1.1 - Refactor, clone support, qos by type and minor bug fixes
    1.2 - Add xfr and retype support
    1.2.1 - Add export/import support
    1.2.2 - Catch VolumeNotFound on accept xfr
    2.0.0 - Move from httplib to requests
    2.0.1 - Implement SolidFire Snapshots
    2.0.2 - Implement secondary account
    2.0.3 - Implement cluster pairing
    2.0.4 - Implement volume replication
    2.0.5 - Try and deal with the stupid retry/clear issues from objects
            and tflow
    2.0.6 - Add a lock decorator around the clone_image method
    2.0.7 - Add scaled IOPS
    2.0.8 - Add active status filter to get volume ops
    2.0.9 - Always purge on delete volume
    2.0.10 - Add response to debug on retryable errors
    2.0.11 - Add ability to failback replicating volumes
    2.0.12 - Fix bug #1744005
    2.0.14 - Fix bug #1782588 qos settings on extend
    2.0.15 - Fix bug #1834013 NetApp SolidFire replication errors
    2.0.16 - Add options for replication mode (Async, Sync and
             SnapshotsOnly)
    2.0.17 - Fix bug #1859653 SolidFire fails to failback when volume
             service is restarted
    2.1.0  - Add Cinder Active/Active support
              - Enable Active/Active support flag
              - Implement Active/Active replication support
    2.2.0  - Add storage assisted volume migration support
    2.2.1  - Fix bug #1891914 fix error on cluster workload rebalancing
             by adding xNotPrimary to the retryable exception list
    2.2.2  - Fix bug #1896112 SolidFire Driver creates duplicate volume
             when API response is lost
    2.2.3  - Fix bug #1942090 SolidFire retype fails due to volume status
             as retyping.
             Fix bug #1932964 SolidFire duplicate volume name exception
             on migration and replication.
    2.2.4  - Fix bug #1934435 fix driver failing with multiple exceptions
             during Element OS upgrade by adding xDBOperationTimeout,
             xDBConnectionLoss, xNoHandler, xSnapshotFailed,
             xRecvTimeout, xDBNoSuchPath, xPermissionDenied to the
             retryable exception list
    2.2.5  - Fix bug #1934459 SolidFire Driver gets into an infinite
             recursion on startup while OS Profiler is enabled
```

StorPoolDriver

- Version: 2.1.0
- volume_driver=cinder.volume.drivers.storpool.StorPoolDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/StorPool_distributed_storage_CI
- Driver Configuration Options:

Table 203: **Driver configuration options**

Name = Default Value	(Type) Description
storpool_replication = 3	(Integer) The default StorPool chain replication value. Used when creating a volume with no specified type if storpool_template is not set. Also used for calculating the apparent free space reported in the stats.
storpool_template = None	(String) The StorPool template for volumes with no type.

- Description: The StorPool block device driver.

Version history:

```
0.1.0   - Initial driver
0.2.0   - Bring the driver up to date with Kilo and Liberty:
          - implement volume retyping and migrations
          - use the driver.*VD ABC metaclasses
          - bugfix: fall back to the configured StorPool template
1.0.0   - Imported into OpenStack Liberty with minor fixes
1.1.0   - Bring the driver up to date with Liberty and Mitaka:
          - drop the CloneableVD and RetypeVD base classes
          - enable faster volume copying by specifying
            sparse_volume_copy=true in the stats report
1.1.1   - Fix the internal _storpool_client_id() method to
          not break on an unknown host name or UUID; thus,
          remove the StorPoolConfigurationMissing exception.
1.1.2   - Bring the driver up to date with Pike: do not
          translate the error messages
1.2.0   - Inherit from VolumeDriver, implement get_pool()
1.2.1   - Implement interface.volumedriver, add CI_WIKI_NAME,
          fix the docstring formatting
1.2.2   - Reintroduce the driver into OpenStack Queens,
          add ignore_errors to the internal _detach_volume() method
1.2.3   - Advertise some more driver capabilities.
2.0.0   - Implement revert_to_snapshot().
2.1.0   - Use the new API client in os-brick to communicate with the
          StorPool API instead of packages `storpool` and
          `storpool.spopenstack`
```

StorwizeSVCFCDriver

- Version: 2.2.6
- volume_driver=cinder.volume.drivers.ibm.storwize_svc.storwize_svc_fc.StorwizeSVCFCDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/IBM_STORAGE_CI
- Driver Configuration Options:

Table 204: **Driver configuration options**

Name = Default Value	(Type) Description
cycle_period_seconds = 300	(Integer(min=60, max=86400)) This defines an optional cycle period that applies to Global Mirror relationships with a cycling mode of multi. A Global Mirror relationship using the multi cycling_mode performs a complete cycle at most once each period. The default is 300 seconds, and the valid seconds are 60-86400.
storwize_peer_pool = None	(String) Specifies the name of the peer pool for hyperswap volume, the peer pool must exist on the other site.
storwize_portset = None	(String) Specifies the name of the portset in which the host is to be created.
storwize_preferred_host_site = {}	(Dict of String) Specifies the site information for host. One WWPN or multi WWPNs used in the host can be specified. For example: storwize_preferred_host_site=site1:wwpn1,site2:wwpn2&wwpn3 or storwize_preferred_host_site=site1:iqn1,site2:iqn2
storwize_san_secondary_ip = None	(String) Specifies secondary management IP or hostname to be used if san_ip is invalid or becomes inaccessible.
storwize_svc_allow_tenant_qos = False	(Boolean) Allow tenants to specify QOS on create
storwize_svc_clean_rate = 50	(Integer(min=0, max=150)) Specifies the Storwize cleaning rate for the mapping. The default rate is 50, and the valid rates are 0-150.
storwize_svc_flashcopy_rate = 50	(Integer(min=1, max=150)) Specifies the Storwize FlashCopy copy rate to be used when creating a full volume copy. The default is rate is 50, and the valid rates are 1-150.
storwize_svc_flashcopy_timeout = 120	(Integer(min=1, max=600)) Maximum number of seconds to wait for FlashCopy to be prepared.
storwize_svc_mirror_pool = None	(String) Specifies the name of the pool in which mirrored copy is stored. Example: pool2
storwize_svc_multihostmap_enable = True	(Boolean) This option no longer has any affect. It is deprecated and will be removed in the next release.
storwize_svc_multipath_enabled = False	(Boolean) Connect with multipath (FC only; iSCSI multipath is controlled by Nova)
storwize_svc_retain_aux_volume = False	(Boolean) Enable or disable retaining of aux volume on secondary storage during delete of the volume on primary storage or moving the primary volume from mirror to non-mirror with replication enabled. This option is valid for Storage Virtualize Family.
storwize_svc_src_child_pool = None	(String) Specifies the name of the source child pool in which global mirror source change volume is stored.
storwize_svc_stretched_cluster_par = None	(String) If operating in stretched cluster mode, specify the name of the pool in which mirrored copies are stored.Example: pool2
storwize_svc_target_child_pool = None	(String) Specifies the name of the target child pool in which global mirror auxiliary change volume is stored.
storwize_svc_vol_autoexpand = True	(Boolean) Storage system autoexpand parameter for volumes (True/False)
storwize_svc_vol_compression = False	(Boolean) Storage system compression option for volumes
storwize_svc_vol_easytier = True	(Boolean) Enable Easy Tier for volumes

- Description: IBM Storwize V7000 and SVC FC volume driver.

Version history:

```
1.0 - Initial driver
1.1 - FC support, create_cloned_volume, volume type support,
      get_volume_stats, minor bug fixes
1.2.0 - Added retype
1.2.1 - Code refactor, improved exception handling
1.2.2 - Fix bug #1274123 (races in host-related functions)
1.2.3 - Fix Fibre Channel connectivity: bug #1279758 (add delim
        to lsfabric, clear unused data from connections, ensure
        matching WWPNs by comparing lower case
1.2.4 - Fix bug #1278035 (async migration/retype)
1.2.5 - Added support for manage_existing (unmanage is inherited)
1.2.6 - Added QoS support in terms of I/O throttling rate
1.3.1 - Added support for volume replication
1.3.2 - Added support for consistency group
1.3.3 - Update driver to use ABC metaclasses
2.0 - Code refactor, split init file and placed shared methods
      for FC and iSCSI within the StorwizeSVCCommonDriver class
2.0.1 - Added support for multiple pools with model update
2.1 - Added replication V2 support to the global/metro mirror
      mode
2.1.1 - Update replication to version 2.1
2.2 - Add CG capability to generic volume groups
2.2.1 - Add vdisk mirror/stretch cluster support
2.2.2 - Add npiv support
2.2.3 - Add replication group support
2.2.4 - Add backup snapshots support
2.2.5 - Add hyperswap support
2.2.6 - Add support for host attachment using portsets
```

StorwizeSVCISCSIDriver

- Version: 2.2.5
- volume_driver=cinder.volume.drivers.ibm.storwize_svc.storwize_svc_iscsi.StorwizeSVCISCSIDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/IBM_STORAGE_CI
- Driver Configuration Options:

Table 205: **Driver configuration options**

Name = Default Value	(Type) Description
cycle_period_seconds = 300	(Integer(min=60, max=86400)) This defines an optional cycle period that applies to Global Mirror relationships with a cycling mode of multi. A Global Mirror relationship using the multi cycling_mode performs a complete cycle at most once each period. The default is 300 seconds, and the valid seconds are 60-86400.
storwize_peer_pool = None	(String) Specifies the name of the peer pool for hyperswap volume, the peer pool must exist on the other site.
storwize_portset = None	(String) Specifies the name of the portset in which the host is to be created.
storwize_preferred_host_site = {}	(Dict of String) Specifies the site information for host. One WWPN or multi WWPNs used in the host can be specified. For example: storwize_preferred_host_site=site1:wwpn1,site2:wwpn2&wwpn3 or storwize_preferred_host_site=site1:iqn1,site2:iqn2
storwize_san_secondary_ip = None	(String) Specifies secondary management IP or hostname to be used if san_ip is invalid or becomes inaccessible.
stor-wize_svc_allow_tenant_qos = False	(Boolean) Allow tenants to specify QOS on create
storwize_svc_clean_rate = 50	(Integer(min=0, max=150)) Specifies the Storwize cleaning rate for the mapping. The default rate is 50, and the valid rates are 0-150.
storwize_svc_flashcopy_rate = 50	(Integer(min=1, max=150)) Specifies the Storwize FlashCopy copy rate to be used when creating a full volume copy. The default is rate is 50, and the valid rates are 1-150.
stor-wize_svc_flashcopy_timeout = 120	(Integer(min=1, max=600)) Maximum number of seconds to wait for FlashCopy to be prepared.
stor-wize_svc_iscsi_chap_enabled = True	(Boolean) Configure CHAP authentication for iSCSI connections (Default: Enabled)
storwize_svc_mirror_pool = None	(String) Specifies the name of the pool in which mirrored copy is stored. Example: pool2
stor-wize_svc_multihostmap_enable = True	(Boolean) This option no longer has any affect. It is deprecated and will be removed in the next release.
stor-wize_svc_retain_aux_volume = False	(Boolean) Enable or disable retaining of aux volume on secondary storage during delete of the volume on primary storage or moving the primary volume from mirror to non-mirror with replication enabled. This option is valid for Storage Virtualize Family.
storwize_svc_src_child_pool = None	(String) Specifies the name of the source child pool in which global mirror source change volume is stored.
stor-wize_svc_stretched_cluster_par = None	(String) If operating in stretched cluster mode, specify the name of the pool in which mirrored copies are stored.Example: pool2
stor-wize_svc_target_child_pool = None	(String) Specifies the name of the target child pool in which global mirror auxiliary change volume is stored.
storwize_svc_vol_autoexpand = True	(Boolean) Storage system autoexpand parameter for volumes (True/False)
stor-wize_svc_vol_compression = False	(Boolean) Storage system compression option for volumes
storwize_svc_vol_easytier = True	(Boolean) Enable Easy Tier for volumes

- Description: IBM Storwize V7000 and SVC iSCSI volume driver.

Version history:

```
1.0 - Initial driver
1.1 - FC support, create_cloned_volume, volume type support,
      get_volume_stats, minor bug fixes
1.2.0 - Added retype
1.2.1 - Code refactor, improved exception handling
1.2.2 - Fix bug #1274123 (races in host-related functions)
1.2.3 - Fix Fibre Channel connectivity: bug #1279758 (add delim
        to lsfabric, clear unused data from connections, ensure
        matching WWPNs by comparing lower case
1.2.4 - Fix bug #1278035 (async migration/retype)
1.2.5 - Added support for manage_existing (unmanage is inherited)
1.2.6 - Added QoS support in terms of I/O throttling rate
1.3.1 - Added support for volume replication
1.3.2 - Added support for consistency group
1.3.3 - Update driver to use ABC metaclasses
2.0 - Code refactor, split init file and placed shared methods
      for FC and iSCSI within the StorwizeSVCCommonDriver class
2.0.1 - Added support for multiple pools with model update
2.1 - Added replication V2 support to the global/metro mirror
      mode
2.1.1 - Update replication to version 2.1
2.2 - Add CG capability to generic volume groups
2.2.1 - Add vdisk mirror/stretch cluster support
2.2.2 - Add replication group support
2.2.3 - Add backup snapshots support
2.2.4 - Add hyperswap support
2.2.5 - Add support for host attachment using portsets
```

SynoISCSIDriver

- Version: 1.0.1
- volume_driver=cinder.volume.drivers.synology.synology_iscsi.SynoISCSIDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/Synology_DSM_CI
- Driver Configuration Options:

Table 206: **Driver configuration options**

Name = Default Value	(Type) Description
chap_password =	(String) Password for specified CHAP account name.
chap_username =	(String) CHAP user name.
driver_use_ssl = False	(Boolean) Tell driver to use SSL for connection to backend storage if the driver supports it.
max_over_subscription_ratio = 20.0	(String(regex=^(auto|\d*\.\d+|\d+)$)) Representation of the over subscription ratio when thin provisioning is enabled. Default ratio is 20.0, meaning provisioned capacity can be 20 times of the total physical capacity. If the ratio is 10.5, it means provisioned capacity can be 10.5 times of the total physical capacity. A ratio of 1.0 means provisioned capacity cannot exceed the total physical capacity. If ratio is auto, Cinder will automatically calculate the ratio based on the provisioned capacity and the used space. If not set to auto, the ratio has to be a minimum of 1.0.
reserved_percentage = 0	(Integer(min=0, max=100)) The percentage of backend capacity is reserved
synology_admin_port = 5000	(Port(min=0, max=65535)) Management port for Synology storage.
synology_device_id = None	(String) Device id for skip one time password check for logging in Synology storage if OTP is enabled.
synology_one_time_pass = None	(String) One time password of administrator for logging in Synology storage if OTP is enabled.
synology_password =	(String) Password of administrator for logging in Synology storage.
synology_pool_name =	(String) Volume on Synology storage to be used for creating lun.
synology_ssl_verify = True	(Boolean) Do certificate validation or not if $driver_use_ssl is True
synology_username = admin	(String) Administrator of Synology storage.
target_ip_address = $my_ip	(String) The IP address that the iSCSI/NVMEoF daemon is listening on
target_port = 3260	(Port(min=0, max=65535)) The port that the iSCSI/NVMEoF daemon is listening on
target_prefix = iqn.2010-10.org.openstack:	(String) Prefix for iSCSI/NVMEoF volumes
target_protocol = iscsi	(String(choices=[iscsi, iser, nvmet_rdma, nvmet_tcp])) Determines the target protocol for new volumes, created with tgtadm, lioadm and nvmet target helpers. In order to enable RDMA, this parameter should be set with the value iser. The supported iSCSI protocol values are iscsi and iser, in case of nvmet target set to nvmet_rdma or nvmet_tcp.
target_secondary_ip_addresses = []	(List of String) The list of secondary IP addresses of the iSCSI/NVMEoF daemon
use_chap_auth = False	(Boolean) Option to enable/disable CHAP authentication for targets.

- Description: OpenStack Cinder drivers for Synology storage.

```
Version history:
    1.0.0 - Initial driver. Provide Cinder minimum features
    1.0.1 - Add support for UC series model
```

TYDSDriver

- Version: 1.0.0
- volume_driver=cinder.volume.drivers.toyou.tyds.tyds.TYDSDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/TOYOU_TYDS_CI
- Driver Configuration Options:

Table 207: **Driver configuration options**

Name = Default Value	(Type) Description
san_ip =	(String) IP address of SAN controller
san_login = admin	(String) Username for SAN controller
san_password =	(String) Password for SAN controller
tyds_clone_progress_interval = 3	(Integer) Interval (in seconds) for retrieving clone progress.
tyds_copy_progress_interval = 3	(Integer) Interval (in seconds) for retrieving copy progress.
tyds_http_port = 80	(Port(min=0, max=65535)) The port that connects to the http api.
tyds_pools = [pool01]	(List of String) The pool name where volumes are stored.
tyds_stripe_size = 4M	(String) Volume stripe size.

- Description: TOYOU distributed storage abstract common class.

```
Version history:
    1.0.0 - Initial TOYOU NetStor TYDS Driver
```

TatlinFCVolumeDriver

- Version: 1.0
- volume_driver=cinder.volume.drivers.yadro.tatlin_fc.TatlinFCVolumeDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/Yadro_Tatlin_Unified_CI
- Driver Configuration Options:

Table 208: **Driver configuration options**

Name = Default Value	(Type) Description
api_port = 443	(Port(min=0, max=65535)) Port to use to access the Tatlin API
auth_method = CHAP	(String) Authentication method for iSCSI (CHAP)
export_ports =	(String) Ports to export Tatlin resource through
host_group =	(String) Tatlin host group name
lba_format = 512e	(String) LBA Format for new volume
max_resource_count = 500	(Integer) Max resource count allowed for Tatlin
pool_max_resource_count = 250	(Integer) Max resource count allowed for single pool
pool_name =	(String) storage pool name
tat_api_retry_count = 10	(Integer) Number of retry on Tatlin API
wait_interval = 30	(Integer) Wait number of seconds before re-checking
wait_retry_count = 15	(Integer) Number of checks for a lengthy operation to finish

- Description: ACCESS Tatlin FC Driver.

Executes commands relating to FC. Supports creation of volumes.

```
API version history:

   1.0 - Initial version.
```

TatlinISCSIVolumeDriver

- Version: 1.1
- volume_driver=cinder.volume.drivers.yadro.tatlin_iscsi.TatlinISCSIVolumeDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/Yadro_Tatlin_Unified_CI
- Driver Configuration Options:

Table 209: **Driver configuration options**

Name = Default Value	(Type) Description
api_port = 443	(Port(min=0, max=65535)) Port to use to access the Tatlin API
auth_method = CHAP	(String) Authentication method for iSCSI (CHAP)
export_ports =	(String) Ports to export Tatlin resource through
host_group =	(String) Tatlin host group name
lba_format = 512e	(String) LBA Format for new volume
max_resource_count = 500	(Integer) Max resource count allowed for Tatlin
pool_max_resource_count = 250	(Integer) Max resource count allowed for single pool
pool_name =	(String) storage pool name
tat_api_retry_count = 10	(Integer) Number of retry on Tatlin API
wait_interval = 30	(Integer) Wait number of seconds before re-checking
wait_retry_count = 15	(Integer) Number of checks for a lengthy operation to finish

- Description: ACCESS Tatlin ISCSI Driver.

Executes commands relating to ISCSI. Supports creation of volumes.

```
API version history:

   1.0 - Initial version.
   1.1 - Common code sharing with FC driver
```

UnityDriver

- Version: 07.02.00
- volume_driver=cinder.volume.drivers.dell_emc.unity.driver.UnityDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/DellEMC_Unity_CI
- Driver Configuration Options:

Table 210: **Driver configuration options**

Name = Default Value	(Type) Description
remove_empty_host = False	(Boolean) To remove the host from Unity when the last LUN is detached from it. By default, it is False.
unity_io_ports = []	(List of String) A comma-separated list of iSCSI or FC ports to be used. Each port can be Unix-style glob expressions.
unity_storage_pool_names = []	(List of String) A comma-separated list of storage pool names to be used.

- Description: Unity Driver.

```
Version history:
  1.0.0 - Initial version
  2.0.0 - Add thin clone support
  3.0.0 - Add IPv6 support
  3.1.0 - Support revert to snapshot API
  4.0.0 - Support remove empty host
  4.2.0 - Support compressed volume
  5.0.0 - Support storage assisted volume migration
  6.0.0 - Support generic group and consistent group
  6.1.0 - Support volume replication
  7.0.0 - Support tiering policy
  7.1.0 - Support consistency group replication
  7.2.0 - Support retype volume
```

VMwareVStorageObjectDriver

- Version: 1.3.0
- volume_driver=cinder.volume.drivers.vmware.fcd.VMwareVStorageObjectDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/VMware_CI
- Driver Configuration Options:

Table 211: **Driver configuration options**

Name = Default Value	(Type) Description
vmware_adapter_type = lsiLogic	(String(choices=[lsiLogic, busLogic, lsiLogicsas, paraVirtual, ide])) Default adapter type to be used for attaching volumes.
vmware_api_retry_count = 10	(Integer) Number of times VMware vCenter server API must be retried upon connection related issues.
vmware_ca_file = None	(String) CA bundle file to use in verifying the vCenter server certificate.
vmware_cluster_name = None	(String) Name of a vCenter compute cluster where volumes should be created.
vmware_connection_pool_size = 10	(Integer) Maximum number of connections in http connection pool.
vmware_datastore_regex = None	(String) Regular expression pattern to match the name of datastores where backend volumes are created.
vmware_enable_volume_stats = False	(Boolean) If true, this enables the fetching of the volume stats from the backend. This has potential performance issues at scale. When False, the driver will not collect ANY stats about the backend.
vmware_host_ip = None	(String) IP address for connecting to VMware vCenter server.
vmware_host_password = None	(String) Password for authenticating with VMware vCenter server.
vmware_host_port = 443	(Port(min=0, max=65535)) Port number for connecting to VMware vCenter server.
vmware_host_username = None	(String) Username for authenticating with VMware vCenter server.
vmware_host_version = None	(String) Optional string specifying the VMware vCenter server version. The driver attempts to retrieve the version from VMware vCenter server. Set this configuration only if you want to override the vCenter server version.
vmware_image_transfer_timeou = 7200	(Integer) Timeout in seconds for VMDK volume transfer between Cinder and Glance.
vmware_insecure = False	(Boolean) If true, the vCenter server certificate is not verified. If false, then the default CA truststore is used for verification. This option is ignored if vmware_ca_file is set.
vmware_lazy_create = True	(Boolean) If true, the backend volume in vCenter server is created lazily when the volume is created without any source. The backend volume is created when the volume is attached, uploaded to image service or during backup.
vmware_max_objects_retrieval = 100	(Integer) Max number of objects to be retrieved per batch. Query results will be obtained in batches from the server and not in one shot. Server may still limit the count to something less than the configured value.
vmware_snapshot_format = template	(String(choices=[template, COW])) Volume snapshot format in vCenter server.
vmware_storage_profile = None	(String) Names of storage profiles to be monitored. Only used when vmware_enable_volume_stats is True.
vmware_task_poll_interval = 2.0	(Float) The interval (in seconds) for polling remote tasks invoked on VMware vCenter server.
vmware_tmp_dir = /tmp	(String) Directory where virtual disks are stored during volume backup and restore.
vmware_volume_folder = Volumes	(String) Name of the vCenter inventory folder that will contain Cinder volumes. This folder will be created under OpenStack/<project_folder>, where project_folder is of format Project (<volume_project_id>).
vmware_wsdl_location = None	(String) Optional VIM service WSDL Location e.g http://<server>/vimService.wsdl. Optional over-ride to default lo-

- Description: Volume driver based on VMware VStorageObject

VMwareVcVmdkDriver

- Version: 3.4.4
- volume_driver=cinder.volume.drivers.vmware.vmdk.VMwareVcVmdkDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/VMware_CI
- Driver Configuration Options:

Table 212: **Driver configuration options**

Name = Default Value	(Type) Description
vmware_adapter_type = lsiLogic	(String(choices=[lsiLogic, busLogic, lsiLogicsas, paraVirtual, ide])) Default adapter type to be used for attaching volumes.
vmware_api_retry_count = 10	(Integer) Number of times VMware vCenter server API must be retried upon connection related issues.
vmware_ca_file = None	(String) CA bundle file to use in verifying the vCenter server certificate.
vmware_cluster_name = None	(String) Name of a vCenter compute cluster where volumes should be created.
vmware_connection_pool_size = 10	(Integer) Maximum number of connections in http connection pool.
vmware_datastore_regex = None	(String) Regular expression pattern to match the name of datastores where backend volumes are created.
vmware_enable_volume_stats = False	(Boolean) If true, this enables the fetching of the volume stats from the backend. This has potential performance issues at scale. When False, the driver will not collect ANY stats about the backend.
vmware_host_ip = None	(String) IP address for connecting to VMware vCenter server.
vmware_host_password = None	(String) Password for authenticating with VMware vCenter server.
vmware_host_port = 443	(Port(min=0, max=65535)) Port number for connecting to VMware vCenter server.
vmware_host_username = None	(String) Username for authenticating with VMware vCenter server.
vmware_host_version = None	(String) Optional string specifying the VMware vCenter server version. The driver attempts to retrieve the version from VMware vCenter server. Set this configuration only if you want to override the vCenter server version.
vmware_image_transfer_timeou = 7200	(Integer) Timeout in seconds for VMDK volume transfer between Cinder and Glance.
vmware_insecure = False	(Boolean) If true, the vCenter server certificate is not verified. If false, then the default CA truststore is used for verification. This option is ignored if vmware_ca_file is set.
vmware_lazy_create = True	(Boolean) If true, the backend volume in vCenter server is created lazily when the volume is created without any source. The backend volume is created when the volume is attached, uploaded to image service or during backup.
vmware_max_objects_retrieval = 100	(Integer) Max number of objects to be retrieved per batch. Query results will be obtained in batches from the server and not in one shot. Server may still limit the count to something less than the configured value.
vmware_snapshot_format = template	(String(choices=[template, COW])) Volume snapshot format in vCenter server.
vmware_storage_profile = None	(String) Names of storage profiles to be monitored. Only used when vmware_enable_volume_stats is True.
vmware_task_poll_interval = 2.0	(Float) The interval (in seconds) for polling remote tasks invoked on VMware vCenter server.
vmware_tmp_dir = /tmp	(String) Directory where virtual disks are stored during volume backup and restore.
vmware_volume_folder = Volumes	(String) Name of the vCenter inventory folder that will contain Cinder volumes. This folder will be created under OpenStack/<project_folder>, where project_folder is of format Project (<volume_project_id>).
vmware_wsdl_location = None	(String) Optional VIM service WSDL Location e.g http://<server>/vimService.wsdl. Optional over-ride to default lo-

- Description: Manage volumes on VMware vCenter server.

VStorageFCDriver

- Version: 1.0.0
- volume_driver=cinder.volume.drivers.nec.v.nec_v_fc.VStorageFCDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/NEC_V_Cinder_CI
- Driver Configuration Options:

Table 213: **Driver configuration options**

Name = Default Value	(Type) Description
chap_password =	(String) Password for specified CHAP account name.
chap_username =	(String) CHAP user name.
driver_ssl_cert_path = None	(String) Can be used to specify a non default path to a CA_BUNDLE file or directory with certificates of trusted CAs, which will be used to validate the backend
driver_ssl_cert_verify = False	(Boolean) If set to True the http client will validate the SSL certificate of the backend endpoint.
enforce_multipath_for_image_xfe = False	(Boolean) If this is set to True, attachment of volumes for image transfer will be aborted when multipathd is not running. Otherwise, it will fallback to single path. This parameter needs to be configured for each backend section or in [backend_defaults] section as a common configuration for all backends.
max_over_subscription_ratio = 20.0	(String(regex=^(auto|\d*\.\d+|\d+)$)) Representation of the over subscription ratio when thin provisioning is enabled. Default ratio is 20.0, meaning provisioned capacity can be 20 times of the total physical capacity. If the ratio is 10.5, it means provisioned capacity can be 10.5 times of the total physical capacity. A ratio of 1.0 means provisioned capacity cannot exceed the total physical capacity. If ratio is auto, Cinder will automatically calculate the ratio based on the provisioned capacity and the used space. If not set to auto, the ratio has to be a minimum of 1.0.
nec_v_async_copy_check_inter = 10	(Integer(min=1, max=600)) Interval in seconds to check asynchronous copying status during a copy pair deletion or data restoration.
nec_v_compute_target_ports = []	(List of String) IDs of the storage ports used to attach volumes to compute nodes. To specify multiple ports, connect them by commas (e.g. CL1-A,CL2-A).
nec_v_copy_check_interval = 3	(Integer(min=1, max=600)) Interval in seconds to check copying status during a volume copy.
nec_v_copy_speed = 3	(Integer(min=1, max=15)) Copy speed of storage system. 1 or 2 indicates low speed, 3 indicates middle speed, and a value between 4 and 15 indicates high speed.
nec_v_discard_zero_page = True	(Boolean) Enable or disable zero page reclamation in a DP-VOL.
nec_v_exec_retry_interval = 5	(Integer) Retry interval in seconds for REST API execution.
nec_v_extend_timeout = 600	(Integer) Maximum wait time in seconds for a volume extention to complete.

continues on next page

Table 213 – continued from previous page

Name = Default Value	(Type) Description
nec_v_group_create = False	(Boolean) If True, the driver will create host groups or iSCSI targets on storage ports as needed.
nec_v_group_delete = False	(Boolean) If True, the driver will delete host groups or iSCSI targets on storage ports as needed.
nec_v_host_mode_options = []	(List of String) Host mode option for host group or iSCSI target
nec_v_ldev_range = None	(String) Range of the LDEV numbers in the format of xxxx-yyyy that can be used by the driver. Values can be in decimal format (e.g. 1000) or in colon-separated hexadecimal format (e.g. 00:03:E8).
nec_v_lock_timeout = 7200	(Integer) Maximum wait time in seconds for storage to be unlocked.
nec_v_lun_retry_interval = 1	(Integer) Retry interval in seconds for REST API adding a LUN.
nec_v_lun_timeout = 50	(Integer) Maximum wait time in seconds for adding a LUN to complete.
nec_v_pools = []	(List of String) Pool number[s] or pool name[s] of the DP pool.
nec_v_rest_another_ldev_mapp = 600	(Integer) Retry time in seconds when new LUN allocation request fails.
nec_v_rest_connect_timeout = 30	(Integer) Maximum wait time in seconds for REST API connection to complete.
nec_v_rest_disable_io_wait = True	(Boolean) It may take some time to detach volume after I/O. This option will allow detaching volume to complete immediately.
nec_v_rest_get_api_response_t = 1800	(Integer) Maximum wait time in seconds for a response against GET method of REST API.
nec_v_rest_job_api_response_t = 1800	(Integer) Maximum wait time in seconds for a response from REST API.
nec_v_rest_keep_session_loop_ = 180	(Integer) Loop interval in seconds for keeping REST API session.
nec_v_rest_server_busy_timeou = 7200	(Integer) Maximum wait time in seconds when REST API returns busy.
nec_v_rest_tcp_keepalive = True	(Boolean) Enables or disables use of REST API tcp keepalive
nec_v_rest_tcp_keepcnt = 4	(Integer) Maximum number of transmissions for TCP keepalive packet.
nec_v_rest_tcp_keepidle = 60	(Integer) Wait time in seconds for sending a first TCP keepalive packet.
nec_v_rest_tcp_keepintvl = 15	(Integer) Interval of transmissions in seconds for TCP keepalive packet.
nec_v_rest_timeout = 30	(Integer) Maximum wait time in seconds for REST API execution to complete.
nec_v_restore_timeout = 86400	(Integer) Maximum wait time in seconds for the restore operation to complete.
nec_v_snap_pool = None	(String) Pool number or pool name of the snapshot pool.
nec_v_state_transition_timeout = 900	(Integer) Maximum wait time in seconds for a volume transition to complete.
nec_v_storage_id = None	(String) Product number of the storage system.
nec_v_target_ports = []	(List of String) IDs of the storage ports used to attach volumes to the controller node. To specify multiple ports, connect them by commas (e.g. CL1-A,CL2-A).

continues on next page

Table 213 – continued from previous page

Name = Default Value	(Type) Description
nec_v_zoning_request = False	(Boolean) If True, the driver will configure FC zoning between the server and the storage system provided that FC zoning manager is enabled.
reserved_percentage = 0	(Integer(min=0, max=100)) The percentage of backend capacity is reserved
san_api_port = None	(Port(min=0, max=65535)) Port to use to access the SAN API
san_ip =	(String) IP address of SAN controller
san_login = admin	(String) Username for SAN controller
san_password =	(String) Password for SAN controller
use_chap_auth = False	(Boolean) Option to enable/disable CHAP authentication for targets.
use_multipath_for_image_xfer = False	(Boolean) Do we attach/detach volumes in cinder using multipath for volume to image and image to volume transfers? This parameter needs to be configured for each backend section or in [backend_defaults] section as a common configuration for all backends.
volume_backend_name = None	(String) The backend name for a given driver implementation
volume_driver = cinder.volume.drivers.lvm.LVMVc	(String) Driver to use for volume creation

- Description: Fibre channel class for NEC Driver.

Version history:

```
1.0.0 - Initial driver.
```

VStorageISCSIDriver

- Version: 1.0.0
- volume_driver=cinder.volume.drivers.nec.v.nec_v_iscsi.VStorageISCSIDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/NEC_V_Cinder_CI
- Driver Configuration Options:

Table 214: **Driver configuration options**

Name = Default Value	(Type) Description
chap_password =	(String) Password for specified CHAP account name.
chap_username =	(String) CHAP user name.
driver_ssl_cert_path = None	(String) Can be used to specify a non default path to a CA_BUNDLE file or directory with certificates of trusted CAs, which will be used to validate the backend
driver_ssl_cert_verify = False	(Boolean) If set to True the http client will validate the SSL certificate of the backend endpoint.
enforce_multipath_for_image_xfe = False	(Boolean) If this is set to True, attachment of volumes for image transfer will be aborted when multipathd is not running. Otherwise, it will fallback to single path. This parameter needs to be configured for each backend section or in [backend_defaults] section as a common configuration for all backends.

continues on next page

Table 214 – continued from previous page

Name = Default Value	(Type) Description
max_over_subscription_ratio = 20.0	(String(regex=^(auto|\d*\.\d+|\d+)$)) Representation of the over subscription ratio when thin provisioning is enabled. Default ratio is 20.0, meaning provisioned capacity can be 20 times of the total physical capacity. If the ratio is 10.5, it means provisioned capacity can be 10.5 times of the total physical capacity. A ratio of 1.0 means provisioned capacity cannot exceed the total physical capacity. If ratio is auto, Cinder will automatically calculate the ratio based on the provisioned capacity and the used space. If not set to auto, the ratio has to be a minimum of 1.0.
nec_v_async_copy_check_inter = 10	(Integer(min=1, max=600)) Interval in seconds to check asynchronous copying status during a copy pair deletion or data restoration.
nec_v_compute_target_ports = []	(List of String) IDs of the storage ports used to attach volumes to compute nodes. To specify multiple ports, connect them by commas (e.g. CL1-A,CL2-A).
nec_v_copy_check_interval = 3	(Integer(min=1, max=600)) Interval in seconds to check copying status during a volume copy.
nec_v_copy_speed = 3	(Integer(min=1, max=15)) Copy speed of storage system. 1 or 2 indicates low speed, 3 indicates middle speed, and a value between 4 and 15 indicates high speed.
nec_v_discard_zero_page = True	(Boolean) Enable or disable zero page reclamation in a DP-VOL.
nec_v_exec_retry_interval = 5	(Integer) Retry interval in seconds for REST API execution.
nec_v_extend_timeout = 600	(Integer) Maximum wait time in seconds for a volume extention to complete.
nec_v_group_create = False	(Boolean) If True, the driver will create host groups or iSCSI targets on storage ports as needed.
nec_v_group_delete = False	(Boolean) If True, the driver will delete host groups or iSCSI targets on storage ports as needed.
nec_v_host_mode_options = []	(List of String) Host mode option for host group or iSCSI target
nec_v_ldev_range = None	(String) Range of the LDEV numbers in the format of xxxx-yyyy that can be used by the driver. Values can be in decimal format (e.g. 1000) or in colon-separated hexadecimal format (e.g. 00:03:E8).
nec_v_lock_timeout = 7200	(Integer) Maximum wait time in seconds for storage to be unlocked.
nec_v_lun_retry_interval = 1	(Integer) Retry interval in seconds for REST API adding a LUN.
nec_v_lun_timeout = 50	(Integer) Maximum wait time in seconds for adding a LUN to complete.
nec_v_pools = []	(List of String) Pool number[s] or pool name[s] of the DP pool.
nec_v_rest_another_ldev_mapp = 600	(Integer) Retry time in seconds when new LUN allocation request fails.
nec_v_rest_connect_timeout = 30	(Integer) Maximum wait time in seconds for REST API connection to complete.
nec_v_rest_disable_io_wait = True	(Boolean) It may take some time to detach volume after I/O. This option will allow detaching volume to complete immediately.
nec_v_rest_get_api_response_t = 1800	(Integer) Maximum wait time in seconds for a response against GET method of REST API.

continues on next page

Table 214 – continued from previous page

Name = Default Value	(Type) Description
nec_v_rest_job_api_response_t = 1800	(Integer) Maximum wait time in seconds for a response from REST API.
nec_v_rest_keep_session_loop_ = 180	(Integer) Loop interval in seconds for keeping REST API session.
nec_v_rest_server_busy_timeou = 7200	(Integer) Maximum wait time in seconds when REST API returns busy.
nec_v_rest_tcp_keepalive = True	(Boolean) Enables or disables use of REST API tcp keepalive
nec_v_rest_tcp_keepcnt = 4	(Integer) Maximum number of transmissions for TCP keepalive packet.
nec_v_rest_tcp_keepidle = 60	(Integer) Wait time in seconds for sending a first TCP keepalive packet.
nec_v_rest_tcp_keepintvl = 15	(Integer) Interval of transmissions in seconds for TCP keepalive packet.
nec_v_rest_timeout = 30	(Integer) Maximum wait time in seconds for REST API execution to complete.
nec_v_restore_timeout = 86400	(Integer) Maximum wait time in seconds for the restore operation to complete.
nec_v_snap_pool = None	(String) Pool number or pool name of the snapshot pool.
nec_v_state_transition_timeout = 900	(Integer) Maximum wait time in seconds for a volume transition to complete.
nec_v_storage_id = None	(String) Product number of the storage system.
nec_v_target_ports = []	(List of String) IDs of the storage ports used to attach volumes to the controller node. To specify multiple ports, connect them by commas (e.g. CL1-A,CL2-A).
reserved_percentage = 0	(Integer(min=0, max=100)) The percentage of backend capacity is reserved
san_api_port = None	(Port(min=0, max=65535)) Port to use to access the SAN API
san_ip =	(String) IP address of SAN controller
san_login = admin	(String) Username for SAN controller
san_password =	(String) Password for SAN controller
use_chap_auth = False	(Boolean) Option to enable/disable CHAP authentication for targets.
use_multipath_for_image_xfer = False	(Boolean) Do we attach/detach volumes in cinder using multipath for volume to image and image to volume transfers? This parameter needs to be configured for each backend section or in [backend_defaults] section as a common configuration for all backends.
volume_backend_name = None	(String) The backend name for a given driver implementation
volume_driver = cinder.volume.drivers.lvm.LVMVc	(String) Driver to use for volume creation

- Description: iSCSI class for NEC Driver.

Version history:

```
1.0.0 - Initial driver.
```

ZadaraVPSAISCSIDriver

- Version: 20.12-24
- volume_driver=cinder.volume.drivers.zadara.zadara.ZadaraVPSAISCSIDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/ZadaraStorage_VPSA_CI
- Driver Configuration Options:

Table 215: **Driver configuration options**

Name = Default Value	(Type) Description
zadara_access_key = None	(String) VPSA access key
zadara_default_snap_policy = False	(Boolean) VPSA - Attach snapshot policy for volumes. If the option is neither configured nor provided as metadata, the VPSA will inherit the default value.
zadara_gen3_vol_compress = False	(Boolean) VPSA - Enable compression for volumes. If the option is neither configured nor provided as metadata, the VPSA will inherit the default value.
zadara_gen3_vol_dedupe = False	(Boolean) VPSA - Enable deduplication for volumes. If the option is neither configured nor provided as metadata, the VPSA will inherit the default value.
zadara_ssl_cert_verify = True	(Boolean) If set to True the http client will validate the SSL certificate of the VPSA endpoint.
zadara_use_iser = True	(Boolean) VPSA - Use ISER instead of iSCSI
zadara_vol_encrypt = False	(Boolean) VPSA - Default encryption policy for volumes. If the option is neither configured nor provided as metadata, the VPSA will inherit the default value.
zadara_vol_name_template = OS_%s	(String) VPSA - Default template for VPSA volume names
zadara_vpsa_host = None	(HostAddress) VPSA - Management Host name or IP address
zadara_vpsa_poolname = None	(String) VPSA - Storage Pool assigned for volumes
zadara_vpsa_port = None	(Port(min=0, max=65535)) VPSA - Port number
zadara_vpsa_use_ssl = False	(Boolean) VPSA - Use SSL connection

- Description: Zadara VPSA iSCSI/iSER volume driver.

```
Version history:
  15.07 - Initial driver
  16.05 - Move from httplib to requests
  19.08 - Add API access key authentication option
  20.01 - Move to json format from xml. Provide manage/unmanage
          volume/snapshot feature
  20.12-01 - Merging with the common code for all the openstack drivers
  20.12-02 - Common code changed as part of fixing
             Zadara github issue #18723
  20.12-03 - Adding the metadata support while creating volume to
             configure vpsa.
  20.12-20 - IPv6 connectivity support for Cinder driver
  20.12-24 - Optimizing get manageable volumes and snapshots
```

Unsupported Drivers

ACCESSIscsiDriver (unsupported)

- Version: 1.0
- volume_driver=cinder.volume.drivers.veritas_access.veritas_iscsi.ACCESSIscsiDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/Veritas_Access_CI
- Driver Configuration Options:

Table 216: **Driver configuration options**

Name = Default Value	(Type) Description
vrts_lun_sparse = True	(Boolean) Create sparse Lun.
vrts_target_config = /etc/cinder/vrts_target.xml	(String) VA config file.

- Description: ACCESS Share Driver.

Executes commands relating to ACCESS ISCSI. Supports creation of volumes on ACCESS.

```
API version history:

  1.0 - Initial version.
```

DPLFCDriver (unsupported)

- Version: 2.0.5
- volume_driver=cinder.volume.drivers.prophetstor.dpl_fc.DPLFCDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/ProphetStor_CI
- Driver Configuration Options:

Table 217: **Driver configuration options**

Name = Default Value	(Type) Description
dpl_pool =	(String) DPL pool uuid in which DPL volumes are stored.
dpl_port = 8357	(Port(min=0, max=65535)) DPL port number.

- Description: <None>

DPLISCSIDriver (unsupported)

- Version: 2.0.5
- volume_driver=cinder.volume.drivers.prophetstor.dpl_iscsi.DPLISCSIDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/ProphetStor_CI
- Driver Configuration Options:

Table 218: **Driver configuration options**

Name = Default Value	(Type) Description
dpl_pool =	(String) DPL pool uuid in which DPL volumes are stored.
dpl_port = 8357	(Port(min=0, max=65535)) DPL port number.

- Description: <None>

FlashSystemFCDriver (unsupported)

- Version: 1.0.12
- volume_driver=cinder.volume.drivers.ibm.flashsystem_fc.FlashSystemFCDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/IBM_STORAGE_CI
- Driver Configuration Options:

Table 219: **Driver configuration options**

Name = Default Value	(Type) Description
flashsystem_connection_protocol = FC	(String) Connection protocol should be FC. (Default is FC.)
flashsystem_multihostmap_enabled = True	(Boolean) Allows vdisk to multi host mapping. (Default is True)

- Description: IBM FlashSystem FC volume driver.

Version history:

```
1.0.0 - Initial driver
1.0.1 - Code clean up
1.0.2 - Add lock into vdisk map/unmap, connection
        initialize/terminate
1.0.3 - Initial driver for iSCSI
1.0.4 - Split Flashsystem driver into common and FC
1.0.5 - Report capability of volume multiattach
1.0.6 - Fix bug #1469581, add I/T mapping check in
        terminate_connection
1.0.7 - Fix bug #1505477, add host name check in
        _find_host_exhaustive for FC
1.0.8 - Fix bug #1572743, multi-attach attribute
        should not be hardcoded, only in iSCSI
1.0.9 - Fix bug #1570574, Cleanup host resource
        leaking, changes only in iSCSI
1.0.10 - Fix bug #1585085, add host name check in
         _find_host_exhaustive for iSCSI
1.0.11 - Update driver to use ABC metaclasses
1.0.12 - Update driver to support Manage/Unmanage
         existing volume
```

FlashSystemISCSIDriver (unsupported)

- Version: 1.0.12
- volume_driver=cinder.volume.drivers.ibm.flashsystem_iscsi.FlashSystemISCSIDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/IBM_STORAGE_CI
- Driver Configuration Options:

Table 220: **Driver configuration options**

Name = Default Value	(Type) Description
flashsystem_connection_protocol = FC	(String) Connection protocol should be FC. (Default is FC.)
flashsystem_multihostmap_enabled = True	(Boolean) Allows vdisk to multi host mapping. (Default is True)

- Description: IBM FlashSystem iSCSI volume driver.

Version history:

```
1.0.0 - Initial driver
1.0.1 - Code clean up
1.0.2 - Add lock into vdisk map/unmap, connection
        initialize/terminate
1.0.3 - Initial driver for iSCSI
1.0.4 - Split Flashsystem driver into common and FC
1.0.5 - Report capability of volume multiattach
1.0.6 - Fix bug #1469581, add I/T mapping check in
        terminate_connection
1.0.7 - Fix bug #1505477, add host name check in
        _find_host_exhaustive for FC
1.0.8 - Fix bug #1572743, multi-attach attribute
        should not be hardcoded, only in iSCSI
1.0.9 - Fix bug #1570574, Cleanup host resource
        leaking, changes only in iSCSI
1.0.10 - Fix bug #1585085, add host name check in
         _find_host_exhaustive for iSCSI
1.0.11 - Update driver to use ABC metaclasses
1.0.12 - Update driver to support Manage/Unmanage
         existing volume
```

QnapISCSIDriver (unsupported)

- Version: 1.2.005
- volume_driver=cinder.volume.drivers.qnap.QnapISCSIDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/QNAP_CI
- Driver Configuration Options:

Table 221: **Driver configuration options**

Name = Default Value	(Type) Description
chap_password =	(String) Password for specified CHAP account name.
chap_username =	(String) CHAP user name.
driver_ssl_cert_verify = False	(Boolean) If set to True the http client will validate the SSL certificate of the backend endpoint.
qnap_management_url = None	(URI) The URL to management QNAP Storage. Driver does not support IPv6 address in URL.
qnap_poolname = None	(String) The pool name in the QNAP Storage
qnap_storage_protocol = iSCSI	(String) Communication protocol to access QNAP storage
reserved_percentage = 0	(Integer(min=0, max=100)) The percentage of backend capacity is reserved
san_login = admin	(String) Username for SAN controller
san_password =	(String) Password for SAN controller
target_ip_address = $my_ip	(String) The IP address that the iSCSI/NVMEoF daemon is listening on
use_chap_auth = False	(Boolean) Option to enable/disable CHAP authentication for targets.

- Description: QNAP iSCSI based cinder driver

```
Version History:
  1.0.0:
        Initial driver (Only iSCSI).
  1.2.001:
        Add supports for Thin Provisioning, SSD Cache, Deduplication,
        Compression and CHAP.
  1.2.002:
        Add support for QES fw 2.0.0.
  1.2.003:
        Add support for QES fw 2.1.0.
  1.2.004:
        Add support for QES fw on TDS series NAS model.
  1.2.005:
        Add support for QTS fw 4.4.0.
```

NOTE: Set driver_ssl_cert_verify as True under backend section to
enable SSL verification.

QuobyteDriver (unsupported)

- Version: 1.1.13
- volume_driver=cinder.volume.drivers.quobyte.QuobyteDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/Quobyte_CI
- Driver Configuration Options:

Table 222: **Driver configuration options**

Name = Default Value	(Type) Description
quobyte_client_cfg = None	(String) Path to a Quobyte Client configuration file.
quobyte_mount_point_base = $state_path/mnt	(String) Base dir containing the mount point for the Quobyte volume.
quobyte_overlay_volumes = False	(Boolean) Create new volumes from the volume_from_snapshot_cache by creating overlay files instead of full copies. This speeds up the creation of volumes from this cache. This feature requires the options quobyte_qcow2_volumes and quobyte_volume_from_snapshot_cache to be set to True. If one of these is set to False this option is ignored.
quobyte_qcow2_volumes = True	(Boolean) Create volumes as QCOW2 files rather than raw files.
quobyte_sparsed_volumes = True	(Boolean) Create volumes as sparse files which take no space. If set to False, volume is created as regular file.
quobyte_volume_from_snapshc = False	(Boolean) Create a cache of volumes from merged snapshots to speed up creation of multiple volumes from a single snapshot.
quobyte_volume_url = None	(String) Quobyte URL to the Quobyte volume using e.g. a DNS SRV record (preferred) or a host list (alternatively) like quobyte://<DIR host1>, <DIR host2>/<volume name>

- Description: Cinder driver for Quobyte USP.

Volumes are stored as files on the mounted Quobyte volume. The hypervisor will expose them as block devices.

Unlike other similar drivers, this driver uses exactly one Quobyte volume because Quobyte USP is a distributed storage system. To add or remove capacity, administrators can add or remove storage servers to/from the volume.

For different types of volumes e.g., SSD vs. rotating disks, use multiple backends in Cinder.

Note: To be compliant with the inherited RemoteFSSnapDriver, Quobyte
volumes are also referred to as shares.

```
Version history:

  1.0   - Initial driver.
  1.1   - Adds optional insecure NAS settings
  1.1.1 - Removes getfattr calls from driver
  1.1.2 - Fixes a bug in the creation of cloned volumes
  1.1.3 - Explicitely mounts Quobyte volumes w/o xattrs
  1.1.4 - Fixes capability to configure redundancy in quobyte_volume_url
  1.1.5 - Enables extension of volumes with snapshots
  1.1.6 - Optimizes volume creation
  1.1.7 - Support fuse subtype based Quobyte mount validation
  1.1.8 - Adds optional snapshot merge caching
  1.1.9 - Support for Qemu >= 2.10.0
  1.1.10 - Adds overlay based volumes for snapshot merge caching
  1.1.11 - NAS secure ownership & permissions are now False by default
  1.1.12 - Ensure the currently configured volume url is always used
```

(continues on next page)

(continued from previous page)

```
1.1.13 - Allow creating volumes from snapshots in state 'backing-up'
```

SCFCDriver (unsupported)

- Version: 4.1.2
- volume_driver=cinder.volume.drivers.dell_emc.sc.storagecenter_fc.SCFCDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/DellEMC_SC_CI
- Driver Configuration Options:

Table 223: **Driver configuration options**

Name = Default Value	(Type) Description
dell_api_async_rest_timeout = 15	(Integer) Dell SC API async call default timeout in seconds.
dell_api_sync_rest_timeout = 30	(Integer) Dell SC API sync call default timeout in seconds.
dell_sc_api_port = 3033	(Port(min=0, max=65535)) Dell API port
dell_sc_server_folder = openstack	(String) Name of the server folder to use on the Storage Center
dell_sc_ssn = 64702	(Integer) Storage Center System Serial Number
dell_sc_verify_cert = False	(Boolean) Enable HTTPS SC certificate verification
dell_sc_volume_folder = openstack	(String) Name of the volume folder to use on the Storage Center
dell_server_os = Red Hat Linux 6.x	(String) Server OS type to use when creating a new server on the Storage Center.
excluded_domain_ip = None	(IPAddress) DEPRECATED: Fault Domain IP to be excluded from iSCSI returns.
excluded_domain_ips = []	(List of IPAddress) Comma separated Fault Domain IPs to be excluded from iSCSI returns.
included_domain_ips = []	(List of IPAddress) Comma separated Fault Domain IPs to be included from iSCSI returns.
secondary_san_ip =	(String) IP address of secondary DSM controller
secondary_san_login = Admin	(String) Secondary DSM user name
secondary_san_password =	(String) Secondary DSM user password name
secondary_sc_api_port = 3033	(Port(min=0, max=65535)) Secondary Dell API port

- Description: Implements commands for Dell Storage Center FC management.

To enable the driver add the following line to the cinder configuration:
volume_driver=cinder.volume.drivers.dell_emc.sc.storagecenter_fc. SCFCDriver

Version history:

```
1.0.0 - Initial driver
1.1.0 - Added extra spec support for Storage Profile selection
1.2.0 - Added consistency group support.
2.0.0 - Switched to inheriting functional objects rather than volume
```

(continues on next page)

(continued from previous page)

```
        driver.
2.1.0 - Added support for ManageableVD.
2.2.0 - Driver retype support for switching volume's Storage Profile
2.3.0 - Added Legacy Port Mode Support
2.3.1 - Updated error handling.
2.4.0 - Added Replication V2 support.
2.4.1 - Updated Replication support to V2.1.
2.5.0 - ManageableSnapshotsVD implemented.
3.0.0 - ProviderID utilized.
3.1.0 - Failback supported.
3.2.0 - Live Volume support.
3.3.0 - Support for a secondary DSM.
3.4.0 - Support for excluding a domain.
3.5.0 - Support for AFO.
3.6.0 - Server type support.
3.7.0 - Support for Data Reduction, Group QOS and Volume QOS.
4.0.0 - Driver moved to dell_emc.
4.1.0 - Timeouts added to rest calls.
4.1.1 - excluded_domain_ips support.
4.1.2 - included_domain_ips IP support.
```

SCISCSIDriver (unsupported)

- Version: 4.1.2
- volume_driver=cinder.volume.drivers.dell_emc.sc.storagecenter_iscsi.SCISCSIDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/DellEMC_SC_CI
- Driver Configuration Options:

Table 224: **Driver configuration options**

Name = Default Value	(Type) Description
dell_api_async_rest_timeout = 15	(Integer) Dell SC API async call default timeout in seconds.
dell_api_sync_rest_timeout = 30	(Integer) Dell SC API sync call default timeout in seconds.
dell_sc_api_port = 3033	(Port(min=0, max=65535)) Dell API port
dell_sc_server_folder = openstack	(String) Name of the server folder to use on the Storage Center
dell_sc_ssn = 64702	(Integer) Storage Center System Serial Number
dell_sc_verify_cert = False	(Boolean) Enable HTTPS SC certificate verification
dell_sc_volume_folder = openstack	(String) Name of the volume folder to use on the Storage Center
dell_server_os = Red Hat Linux 6.x	(String) Server OS type to use when creating a new server on the Storage Center.
excluded_domain_ip = None	(IPAddress) DEPRECATED: Fault Domain IP to be excluded from iSCSI returns.
excluded_domain_ips = []	(List of IPAddress) Comma separated Fault Domain IPs to be excluded from iSCSI returns.
included_domain_ips = []	(List of IPAddress) Comma separated Fault Domain IPs to be included from iSCSI returns.
secondary_san_ip =	(String) IP address of secondary DSM controller
secondary_san_login = Admin	(String) Secondary DSM user name
secondary_san_password =	(String) Secondary DSM user password name
secondary_sc_api_port = 3033	(Port(min=0, max=65535)) Secondary Dell API port

- Description: Implements commands for Dell Storage Center ISCSI management.

To enable the driver add the following line to the cinder configuration:
volume_driver=cinder.volume.drivers.dell_emc.sc. storagecenter_iscsi.SCISCSIDriver

Version history:

```
1.0.0 - Initial driver
1.1.0 - Added extra spec support for Storage Profile selection
1.2.0 - Added consistency group support.
2.0.0 - Switched to inheriting functional objects rather than volume
        driver.
2.1.0 - Added support for ManageableVD.
2.2.0 - Driver retype support for switching volume's Storage Profile.
        Added API 2.2 support.
2.3.0 - Added Legacy Port Mode Support
2.3.1 - Updated error handling.
2.4.0 - Added Replication V2 support.
2.4.1 - Updated Replication support to V2.1.
2.5.0 - ManageableSnapshotsVD implemented.
3.0.0 - ProviderID utilized.
3.1.0 - Failback Supported.
```

(continues on next page)

(continued from previous page)

```
3.2.0 - Live Volume support.
3.3.0 - Support for a secondary DSM.
3.4.0 - Support for excluding a domain.
3.5.0 - Support for AFO.
3.6.0 - Server type support.
3.7.0 - Support for Data Reduction, Group QOS and Volume QOS.
4.0.0 - Driver moved to dell_emc.
4.1.0 - Timeouts added to rest calls.
4.1.1 - excluded_domain_ips support.
4.1.2 - included_domain_ips IP support.
```

VNXDriver (unsupported)

- Version: 14.00.01
- volume_driver=cinder.volume.drivers.dell_emc.vnx.driver.VNXDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/DellEMC_VNX_CI
- Driver Configuration Options:

Table 225: **Driver configuration options**

Name = Default Value	(Type) Description
check_max_pool_luns_threshol = False	(Boolean) DEPRECATED: Report free_capacity_gb as 0 when the limit to maximum number of pool LUNs is reached. By default, the value is False.
default_timeout = 31536000	(Integer) Default timeout for CLI operations in minutes. For example, LUN migration is a typical long running operation, which depends on the LUN size and the load of the array. An upper bound in the specific deployment can be set to avoid unnecessary long wait. By default, it is 365 days long.
destroy_empty_storage_group = False	(Boolean) To destroy storage group when the last LUN is removed from it. By default, the value is False.
force_delete_lun_in_storagegro = True	(Boolean) Delete a LUN even if it is in Storage Groups.
ignore_pool_full_threshold = False	(Boolean) Force LUN creation even if the full threshold of pool is reached. By default, the value is False.
initiator_auto_deregistration = False	(Boolean) Automatically deregister initiators after the related storage group is destroyed. By default, the value is False.
initiator_auto_registration = False	(Boolean) Automatically register initiators. By default, the value is False.
io_port_list = None	(List of String) Comma separated iSCSI or FC ports to be used in Nova or Cinder.
iscsi_initiators = None	(String) Mapping between hostname and its iSCSI initiator IP addresses.
max_luns_per_storage_group = 255	(Integer) Default max number of LUNs in a storage group. By default, the value is 255.
naviseccli_path = None	(String) Naviseccli Path.
storage_vnx_authentication_type = global	(String) VNX authentication scope type. By default, the value is global.
storage_vnx_pool_names = None	(List of String) Comma-separated list of storage pool names to be used.
storage_vnx_security_file_dir = None	(String) Directory path that contains the VNX security file. Make sure the security file is generated first.
vnx_async_migrate = True	(Boolean) Always use asynchronous migration during volume cloning and creating from snapshot. As described in configuration doc, async migration has some constraints. Besides using metadata, customers could use this option to disable async migration. Be aware that *async_migrate* in metadata overrides this option when both are set. By default, the value is True.

- Description: Dell EMC Cinder Driver for VNX using CLI.

```
Version history:
    1.0.0 - Initial driver
    2.0.0 - Thick/thin provisioning, robust enhancement
    3.0.0 - Array-based Backend Support, FC Basic Support,
            Target Port Selection for MPIO,
```

(continues on next page)

(continued from previous page)

```
          Initiator Auto Registration,
          Storage Group Auto Deletion,
          Multiple Authentication Type Support,
          Storage-Assisted Volume Migration,
          SP Toggle for HA
3.0.1 - Security File Support
4.0.0 - Advance LUN Features (Compression Support,
          Deduplication Support, FAST VP Support,
          FAST Cache Support), Storage-assisted Retype,
          External Volume Management, Read-only Volume,
          FC Auto Zoning
4.1.0 - Consistency group support
5.0.0 - Performance enhancement, LUN Number Threshold Support,
          Initiator Auto Deregistration,
          Force Deleting LUN in Storage Groups,
          robust enhancement
5.1.0 - iSCSI multipath enhancement
5.2.0 - Pool-aware scheduler support
5.3.0 - Consistency group modification support
6.0.0 - Over subscription support
          Create consistency group from cgsnapshot support
          Multiple pools support enhancement
          Manage/unmanage volume revise
          White list target ports support
          Snap copy support
          Support efficient non-disruptive backup
7.0.0 - Clone consistency group support
          Replication v2 support(managed)
          Configurable migration rate support
8.0.0 - New VNX Cinder driver
9.0.0 - Use asynchronous migration for cloning
10.0.0 - Extend SMP size before async migration when cloning from an
           image cache volume
10.1.0 - Add QoS support
10.2.0 - Add replication group support
11.0.0 - Fix failure of migration during cloning
12.0.0 - Add `volume revert to snapshot` support
12.1.0 - Adjust max_luns_per_storage_group and
           check_max_pool_luns_threshold
12.1.1 - Fix perf issue when create/delete volume
13.0.0 - Fix bug https://bugs.launchpad.net/cinder/+bug/1817385 to
           make sure sg can be created again after it was destroyed
           under `destroy_empty_stroage_group` setting to `True`
14.0.0 - Fix bug 1794646: failed to delete LUNs from backend due to
           the temporary snapshots on them wasn't deleted.
14.0.1 - Fix bug 1796825, add an option to set default value for
           `async_migrate`.
```

VZStorageDriver (unsupported)

- Version: 1.1
- volume_driver=cinder.volume.drivers.vzstorage.VZStorageDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/Virtuozzo_Storage_CI
- Driver Configuration Options:

Table 226: **Driver configuration options**

Name = Default Value	(Type) Description
vzstorage_default_volume_format = raw	(String) Default format that will be used when creating volumes if no volume format is specified.
vzstorage_mount_options = None	(List of String) Mount options passed to the vzstorage client. See section of the pstorage-mount man page for details.
vzstorage_mount_point_base = $state_path/mnt	(String) Base dir containing mount points for vzstorage shares.
vzstorage_shares_config = /etc/cinder/vzstorage_shares	(String) File with the list of available vzstorage shares.
vzstorage_sparsed_volumes = True	(Boolean) Create volumes as sparsed files which take no space rather than regular files when using raw format, in which case volume creation takes lot of time.
vzstorage_used_ratio = 0.95	(Float) Percent of ACTUAL usage of the underlying volume before no new volumes can be allocated to the volume destination.

- Description: Cinder driver for Virtuozzo Storage.

Creates volumes as files on the mounted vzstorage cluster.

```
Version history:
  1.0     - Initial driver.
  1.1     - Supports vz:volume_format in vendor properties.
```

VeritasCNFSDriver (unsupported)

- Version: 1.0.3
- volume_driver=cinder.volume.drivers.veritas_cnfs.VeritasCNFSDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/Veritas_Access_CI
- Driver Configuration Options:

Table 227: **Driver configuration options**

Name = Default Value	(Type) Description
nas_host =	(String) IP address or Hostname of NAS system.
nas_login = admin	(String) User name to connect to NAS system.
nas_mount_options = None	(String) Options used to mount the storage backend file system where Cinder volumes are stored.
nas_password =	(String) Password to connect to NAS system.
nas_private_key =	(String) Filename of private key to use for SSH authentication.
nas_secure_file_operations = auto	(String) Allow network-attached storage systems to operate in a secure environment where root level access is not permitted. If set to False, access is as the root user and insecure. If set to True, access is not as root. If set to auto, a check is done to determine if this is a new installation: True is used if so, otherwise False. Default is auto.
nas_secure_file_permissions = auto	(String) Set more secure file permissions on network-attached storage volume files to restrict broad other/world access. If set to False, volumes are created with open permissions. If set to True, volumes are created with permissions for the cinder user and group (660). If set to auto, a check is done to determine if this is a new installation: True is used if so, otherwise False. Default is auto.
nas_share_path =	(String) Path to the share to use for storing Cinder volumes. For example: /srv/export1 for an NFS server export available at 10.0.5.10:/srv/export1 .
nas_ssh_port = 22	(Port(min=0, max=65535)) SSH port to use to connect to NAS system.
nfs_mount_attempts = 3	(Integer) The number of attempts to mount NFS shares before raising an error. At least one attempt will be made to mount an NFS share, regardless of the value specified.
nfs_mount_options = None	(String) Mount options passed to the NFS client. See the NFS(5) man page for details.
nfs_mount_point_base = $state_path/mnt	(String) Base dir containing mount points for NFS shares.
nfs_qcow2_volumes = False	(Boolean) Create volumes as QCOW2 files rather than raw files.
nfs_shares_config = /etc/cinder/nfs_shares	(String) File with the list of available NFS shares.
nfs_snapshot_support = False	(Boolean) Enable support for snapshots on the NFS driver. Platforms using libvirt <1.2.7 will encounter issues with this feature.
nfs_sparsed_volumes = True	(Boolean) Create volumes as sparsed files which take no space. If set to False volume is created as regular file. In such case volume creation takes a lot of time.

- Description: Veritas Clustered NFS based cinder driver

```
Version History:

  1.0.0 - Initial driver implementations for Kilo.
  1.0.1 - Liberty release driver not implemented.
          Place holder for Liberty release in case we
          need to support.
  1.0.2 - cinder.interface.volumedriver decorator.
```

(continues on next page)

(continued from previous page)

```
         Mitaka/Newton/Okata Release
1.0.3 - Separate create_cloned_volume() and
         create_volume_from_snapshot () functionality.
         Pike Release
```

Executes commands relating to Volumes.

WindowsISCSIDriver (unsupported)

- Version: 1.0.0
- volume_driver=cinder.volume.drivers.windows.iscsi.WindowsISCSIDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/Microsoft_iSCSI_CI
- Driver Configuration Options:

Table 228: **Driver configuration options**

Name = Default Value	(Type) Description
windows_iscsi_lun_path = C:iSCSIVirtualDisks	(String) Path to store VHD backed volumes

- Description: Executes volume driver commands on Windows Storage server.

WindowsSmbfsDriver (unsupported)

- Version: 1.1.0
- volume_driver=cinder.volume.drivers.windows.smbfs.WindowsSmbfsDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/Cloudbase_Cinder_SMB3_CI
- Driver Configuration Options:

Table 229: **Driver configuration options**

Name = Default Value	(Type) Description
smbfs_default_volume_format = vhd	(String(choices=[vhd, vhdx])) Default format that will be used when creating volumes if no volume format is specified.
smbfs_mount_point_base = C:OpenStack_mnt	(String) Base dir containing mount points for smbfs shares.
smbfs_pool_mappings = { }	(Dict of String) Mappings between share locations and pool names. If not specified, the share names will be used as pool names. Example: //addr/share:pool_name,//addr/share2:pool_name2
smbfs_shares_config = C:OpenStacksmbfs_shares.txt	(String) File with the list of available smbfs shares.

- Description: <None>

XtremIOFCDriver (unsupported)

- Version: 1.0.13
- volume_driver=cinder.volume.drivers.dell_emc.xtremio.XtremIOFCDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/DellEMC_XtremIO_CI
- Driver Configuration Options:

Table 230: **Driver configuration options**

Name = Default Value	(Type) Description
driver_ssl_cert_path = None	(String) Can be used to specify a non default path to a CA_BUNDLE file or directory with certificates of trusted CAs, which will be used to validate the backend
driver_ssl_cert_verify = False	(Boolean) If set to True the http client will validate the SSL certificate of the backend endpoint.
max_over_subscription_ratio = 20.0	(String(regex=^(auto|\d*\.\d+|\d+)$)) Representation of the over subscription ratio when thin provisioning is enabled. Default ratio is 20.0, meaning provisioned capacity can be 20 times of the total physical capacity. If the ratio is 10.5, it means provisioned capacity can be 10.5 times of the total physical capacity. A ratio of 1.0 means provisioned capacity cannot exceed the total physical capacity. If ratio is auto, Cinder will automatically calculate the ratio based on the provisioned capacity and the used space. If not set to auto, the ratio has to be a minimum of 1.0.
reserved_percentage = 0	(Integer(min=0, max=100)) The percentage of backend capacity is reserved
san_ip =	(String) IP address of SAN controller
san_login = admin	(String) Username for SAN controller
san_password =	(String) Password for SAN controller
xtremio_array_busy_retry_cou = 5	(Integer) Number of retries in case array is busy
xtremio_array_busy_retry_inte = 5	(Integer) Interval between retries in case array is busy
xtremio_clean_unused_ig = False	(Boolean) Should the driver remove initiator groups with no volumes after the last connection was terminated. Since the behavior till now was to leave the IG be, we default to False (not deleting IGs without connected volumes); setting this parameter to True will remove any IG after terminating its connection to the last volume.
xtremio_cluster_name =	(String) XMS cluster id in multi-cluster environment
xtremio_ports = []	(List of String) Allowed ports. Comma separated list of XtremIO iSCSI IPs or FC WWNs (ex. 58:cc:f0:98:49:22:07:02) to be used. If option is not set all ports are allowed.
xtremio_volumes_per_glance_c = 100	(Integer) Number of volumes created from each cached glance image

- Description: <None>

XtremIOISCSIDriver (unsupported)

- Version: 1.0.13
- volume_driver=cinder.volume.drivers.dell_emc.xtremio.XtremIOISCSIDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/DellEMC_XtremIO_CI
- Driver Configuration Options:

Table 231: **Driver configuration options**

Name = Default Value	(Type) Description
driver_ssl_cert_path = None	(String) Can be used to specify a non default path to a CA_BUNDLE file or directory with certificates of trusted CAs, which will be used to validate the backend
driver_ssl_cert_verify = False	(Boolean) If set to True the http client will validate the SSL certificate of the backend endpoint.
max_over_subscription_ratio = 20.0	(String(regex=^(auto|\d*\.\d+|\d+)$)) Representation of the over subscription ratio when thin provisioning is enabled. Default ratio is 20.0, meaning provisioned capacity can be 20 times of the total physical capacity. If the ratio is 10.5, it means provisioned capacity can be 10.5 times of the total physical capacity. A ratio of 1.0 means provisioned capacity cannot exceed the total physical capacity. If ratio is auto, Cinder will automatically calculate the ratio based on the provisioned capacity and the used space. If not set to auto, the ratio has to be a minimum of 1.0.
reserved_percentage = 0	(Integer(min=0, max=100)) The percentage of backend capacity is reserved
san_ip =	(String) IP address of SAN controller
san_login = admin	(String) Username for SAN controller
san_password =	(String) Password for SAN controller
xtremio_array_busy_retry_cou = 5	(Integer) Number of retries in case array is busy
xtremio_array_busy_retry_inte = 5	(Integer) Interval between retries in case array is busy
xtremio_clean_unused_ig = False	(Boolean) Should the driver remove initiator groups with no volumes after the last connection was terminated. Since the behavior till now was to leave the IG be, we default to False (not deleting IGs without connected volumes); setting this parameter to True will remove any IG after terminating its connection to the last volume.
xtremio_cluster_name =	(String) XMS cluster id in multi-cluster environment
xtremio_ports = []	(List of String) Allowed ports. Comma separated list of XtremIO iSCSI IPs or FC WWNs (ex. 58:cc:f0:98:49:22:07:02) to be used. If option is not set all ports are allowed.
xtremio_volumes_per_glance_c = 100	(Integer) Number of volumes created from each cached glance image

- Description: Executes commands relating to ISCSI volumes.

We make use of model provider properties as follows:

provider_location
if present, contains the iSCSI target information in the same format as an ietadm discovery i.e.

<ip>:<port>,<portal> <target IQN>

provider_auth
if present, contains a space-separated triple: <auth method> <auth username> <auth password>. *CHAP* is the only auth_method in use at the moment.

Backup Drivers

CephBackupDriver

- backup_driver=cinder.backup.drivers.ceph.CephBackupDriver
- Driver Configuration Options:

Table 232: **Driver configuration options**

Name = Default Value	(Type) Description
backup_ceph_chunk_size = 134217728	(Integer) The chunk size, in bytes, that a backup is broken into before transfer to the Ceph object store.
backup_ceph_conf = /etc/ceph/ceph.conf	(String) Ceph configuration file to use.
backup_ceph_image_journals = False	(Boolean) If True, apply JOURNALING and EXCLUSIVE_LOCK feature bits to the backup RBD objects to allow mirroring
backup_ceph_max_snapshots = 0	(Integer) Number of the most recent snapshots to keep.

0 indicates to keep an unlimited number of snapshots.

Configuring this option can save disk space by only keeping a limited number of snapshots on the source volume storage. However, if a user deletes all incremental backups which still have snapshots on the source storage, the next incremental backup will automatically become a full backup as no common snapshot exists anymore.

- - backup_ceph_pool = backups
  - (String) The Ceph pool where volume backups are stored.
- - backup_ceph_stripe_count = 0
  - (Integer) RBD stripe count to use when creating a backup image.
- - backup_ceph_stripe_unit = 0
  - (Integer) RBD stripe unit to use when creating a backup image.
- - backup_ceph_user = cinder
  - (String) The Ceph user to connect with. Default here is to use the same user as for Cinder volumes. If not using cephx this should be set to None.
- - restore_discard_excess_bytes = True
  - (Boolean) If True, always discard excess bytes when restoring volumes i.e. pad with zeroes.
- Description: Backup Cinder volumes to Ceph Object Store.

This class enables backing up Cinder volumes to a Ceph object store. Backups may be stored in their own pool or even cluster. Store location is defined by the Ceph conf file and service config options supplied.

If the source volume is itself an RBD volume, the backup will be performed using incremental differential backups which *should* give a performance gain.

GlusterfsBackupDriver

- backup_driver=cinder.backup.drivers.glusterfs.GlusterfsBackupDriver
- Driver Configuration Options:

Table 233: **Driver configuration options**

Name = Default Value	(Type) Description
glusterfs_backup_mount_point = $state_path/backup_mount	(String) Base dir containing mount point for gluster share.
glusterfs_backup_share = None	(String) GlusterFS share in <hostname|ipv4addr|ipv6addr>:<gluster_vol_name> format. Eg: 1.2.3.4:backup_vol

- Description: Provides backup, restore and delete using GlusterFS repository.

GoogleBackupDriver

- backup_driver=cinder.backup.drivers.gcs.GoogleBackupDriver
- Driver Configuration Options:

Table 234: **Driver configuration options**

Name = Default Value	(Type) Description
backup_gcs_block_size = 32768	(Integer) The size in bytes that changes are tracked for incremental backups. backup_gcs_object_size has to be multiple of backup_gcs_block_size.
backup_gcs_bucket = None	(String) The GCS bucket to use.
backup_gcs_bucket_location = US	(String) Location of GCS bucket.
backup_gcs_credential_file = None	(String) Absolute path of GCS service account credential file.
backup_gcs_enable_progress_t = True	(Boolean) Enable or Disable the timer to send the periodic progress notifications to Ceilometer when backing up the volume to the GCS backend storage. The default value is True to enable the timer.
backup_gcs_num_retries = 3	(Integer) Number of times to retry.
backup_gcs_object_size = 52428800	(Integer) The size in bytes of GCS backup objects.
backup_gcs_project_id = None	(String) Owner project id for GCS bucket.
backup_gcs_proxy_url = None	(URI) URL for http proxy access.
backup_gcs_reader_chunk_size = 2097152	(Integer) GCS object will be downloaded in chunks of bytes.
backup_gcs_retry_error_codes = [429]	(List of String) List of GCS error codes.
backup_gcs_storage_class = NEARLINE	(String) Storage class of GCS bucket.
backup_gcs_user_agent = gcscinder	(String) Http user-agent string for gcs api.
backup_gcs_writer_chunk_size = 2097152	(Integer) GCS object will be uploaded in chunks of bytes. Pass in a value of -1 if the file is to be uploaded as a single chunk.

- Description: Provides backup, restore and delete of backup objects within GCS.

NFSBackupDriver

- backup_driver=cinder.backup.drivers.nfs.NFSBackupDriver
- Driver Configuration Options:

Table 235: **Driver configuration options**

Name = Default Value	(Type) Description
backup_container = None	(String) Custom directory to use for backups.
backup_enable_progress_timer = True	(Boolean) Enable or Disable the timer to send the periodic progress notifications to Ceilometer when backing up the volume to the backend storage. The default value is True to enable the timer.
backup_file_size = 1999994880	(Integer) The maximum size in bytes of the files used to hold backups. If the volume being backed up exceeds this size, then it will be backed up into multiple files. backup_file_size also determines the buffer size used to build backup files, so should be scaled according to available RAM and number of workers. backup_file_size must be a multiple of backup_sha_block_size_bytes.
backup_posix_path = $state_path/backup	(String) Path specifying where to store backups.
backup_sha_block_size_bytes = 32768	(Integer) The size in bytes that changes are tracked for incremental backups. backup_file_size has to be multiple of backup_sha_block_size_bytes.

- Description: Provides backup, restore and delete using NFS supplied repository.

PosixBackupDriver

- backup_driver=cinder.backup.drivers.posix.PosixBackupDriver
- Driver Configuration Options:

Table 236: **Driver configuration options**

Name = Default Value	(Type) Description
backup_container = None	(String) Custom directory to use for backups.
backup_enable_progress_timer = True	(Boolean) Enable or Disable the timer to send the periodic progress notifications to Ceilometer when backing up the volume to the backend storage. The default value is True to enable the timer.
backup_file_size = 1999994880	(Integer) The maximum size in bytes of the files used to hold backups. If the volume being backed up exceeds this size, then it will be backed up into multiple files. backup_file_size also determines the buffer size used to build backup files, so should be scaled according to available RAM and number of workers. backup_file_size must be a multiple of backup_sha_block_size_bytes.
backup_posix_path = $state_path/backup	(String) Path specifying where to store backups.
backup_sha_block_size_bytes = 32768	(Integer) The size in bytes that changes are tracked for incremental backups. backup_file_size has to be multiple of backup_sha_block_size_bytes.

- Description: Provides backup, restore and delete using a Posix file system.

S3BackupDriver

- backup_driver=cinder.backup.drivers.s3.S3BackupDriver
- Driver Configuration Options:

Table 237: **Driver configuration options**

Name = Default Value	(Type) Description
backup_compression_algorithm = zlib	(String(choices=[none, off, no, zlib, gzip, bz2, bzip2, zstd])) Compression algorithm for backups (none to disable)
backup_s3_block_size = 32768	(Integer) The size in bytes that changes are tracked for incremental backups. backup_s3_object_size has to be multiple of backup_s3_block_size.
backup_s3_ca_cert_file = None	(String) path/to/cert/bundle.pem - A filename of the CA cert bundle to use.
backup_s3_enable_progress_timer = True	(Boolean) Enable or Disable the timer to send the periodic progress notifications to Ceilometer when backing up the volume to the S3 backend storage. The default value is True to enable the timer.
backup_s3_endpoint_url = None	(String) The url where the S3 server is listening.
backup_s3_http_proxy =	(String) Address or host for the http proxy server.
backup_s3_https_proxy =	(String) Address or host for the https proxy server.
backup_s3_max_pool_connections = 10	(Integer) The maximum number of connections to keep in a connection pool.
backup_s3_md5_validation = True	(Boolean) Enable or Disable md5 validation in the s3 backend.
backup_s3_object_size = 52428800	(Integer) The size in bytes of S3 backup objects
backup_s3_retry_max_attempts = 4	(Integer) An integer representing the maximum number of retry attempts that will be made on a single request.
backup_s3_retry_mode = legacy	(String) A string representing the type of retry mode. e.g: legacy, standard, adaptive
backup_s3_sse_customer_algorithm = None	(String) The SSECustomerAlgorithm. backup_s3_sse_customer_key must be set at the same time to enable SSE.
backup_s3_sse_customer_key = None	(String) The SSECustomerKey. backup_s3_sse_customer_algorithm must be set at the same time to enable SSE.
backup_s3_store_access_key = None	(String) The S3 query token access key.
backup_s3_store_bucket = volumebackups	(String) The S3 bucket to be used to store the Cinder backup data.
backup_s3_store_secret_key = None	(String) The S3 query token secret key.
backup_s3_timeout = 60	(Float) The time in seconds till a timeout exception is thrown.
backup_s3_verify_ssl = True	(Boolean) Enable or Disable ssl verify.

- Description: Provides backup, restore and delete of backup objects within S3.

SwiftBackupDriver

- backup_driver=cinder.backup.drivers.swift.SwiftBackupDriver
- Driver Configuration Options:

Table 238: **Driver configuration options**

Name = Default Value	(Type) Description
backup_swift_auth = per_user	(String(choices=[per_user, single_user])) Swift authentication mechanism (per_user or single_user).
backup_swift_auth_insecure = False	(Boolean) Bypass verification of server certificate when making SSL connection to Swift.
backup_swift_auth_url = None	(URI) The URL of the Keystone endpoint
backup_swift_auth_version = 1	(String) Swift authentication version. Specify 1 for auth 1.0, or 2 for auth 2.0 or 3 for auth 3.0
backup_swift_block_size = 32768	(Integer) The size in bytes that changes are tracked for incremental backups. backup_swift_object_size has to be multiple of backup_swift_block_size.
backup_swift_ca_cert_file = None	(String) Location of the CA certificate file to use for swift client requests.
backup_swift_container = volumebackups	(String) The default Swift container to use
backup_swift_create_storage_p = None	(String) The storage policy to use when creating the Swift container. If the container already exists the storage policy cannot be enforced
backup_swift_enable_progress_ = True	(Boolean) Enable or Disable the timer to send the periodic progress notifications to Ceilometer when backing up the volume to the Swift backend storage. The default value is True to enable the timer.
backup_swift_key = None	(String) Swift key for authentication
backup_swift_object_size = 52428800	(Integer) The size in bytes of Swift backup objects
backup_swift_project = None	(String) Swift project/account name. Required when connecting to an auth 3.0 system
backup_swift_project_domain = None	(String) Swift project domain name. Required when connecting to an auth 3.0 system
backup_swift_retry_attempts = 3	(Integer) The number of retries to make for Swift operations
backup_swift_retry_backoff = 2	(Integer) The backoff time in seconds between Swift retries
backup_swift_service_auth = False	(Boolean) Send a X-Service-Token header with service auth credentials. If enabled you also must set the service_user group and enable send_service_user_token.
backup_swift_tenant = None	(String) Swift tenant/account name. Required when connecting to an auth 2.0 system
backup_swift_url = None	(URI) The URL of the Swift endpoint
backup_swift_user = None	(String) Swift user name
backup_swift_user_domain = None	(String) Swift user domain name. Required when connecting to an auth 3.0 system
keystone_catalog_info = identity:Identity Service:publicURL	(String) Info to match when looking for keystone in the service catalog. Format is: separated values of the form: <service_type>:<service_name>:<endpoint_type> - Only used if backup_swift_auth_url is unset
swift_catalog_info = object-store:swift:publicURL	(String) Info to match when looking for swift in the service catalog. Format is: separated values of the form: <service_type>:<service_name>:<endpoint_type> - Only used if backup_swift_url is unset

- Description: Provides backup, restore and delete of backup objects within Swift.

FC Zone Manager Drivers

BrcdFCZoneDriver (unsupported)

- Version: 1.6
- zone_driver=cinder.zonemanager.drivers.brocade.brcd_fc_zone_driver.BrcdFCZoneDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/Brocade_OpenStack_CI
- Driver Configuration Options:

Table 239: **Driver configuration options**

Name = Default Value	(Type) Description
brcd_sb_connector = HTTP	(String) South bound connector for zoning operation
fc_fabric_address =	(String) Management IP of fabric.
fc_fabric_password =	(String) Password for user.
fc_fabric_port = 22	(Port(min=0, max=65535)) Connecting port
fc_fabric_ssh_cert_path =	(String) Local SSH certificate Path.
fc_fabric_user =	(String) Fabric user ID.
fc_southbound_protocol = REST_HTTP	(String(choices=[SSH, HTTP, HTTPS, REST_HTTP, REST_HTTPS])) South bound connector for the fabric.
fc_virtual_fabric_id = None	(String) Virtual Fabric ID.
zone_activate = True	(Boolean) Overridden zoning activation state.
zone_name_prefix = openstack	(String) Overridden zone name prefix.
zoning_policy = initiator-target	(String) Overridden zoning policy.

- Description: Brocade FC zone driver implementation.

OpenStack Fibre Channel zone driver to manage FC zoning in Brocade SAN fabrics.

```
Version history:
  1.0 - Initial Brocade FC zone driver
  1.1 - Implements performance enhancements
  1.2 - Added support for friendly zone name
  1.3 - Added HTTP connector support
  1.4 - Adds support to zone in Virtual Fabrics
  1.5 - Initiator zoning updates through zoneadd/zoneremove
  1.6 - Add REST connector
```

CiscoFCZoneDriver

- Version: 1.1.0
- zone_driver=cinder.zonemanager.drivers.cisco.cisco_fc_zone_driver.CiscoFCZoneDriver
- CI info: https://wiki.openstack.org/wiki/ThirdPartySystems/Cisco_ZM_CI
- Driver Configuration Options:

Table 240: **Driver configuration options**

Name = Default Value	(Type) Description
cisco_sb_connector = cinder.zonemanager.drivers.cisco.c	(String) Southbound connector for zoning operation

- Description: Cisco FC zone driver implementation.

OpenStack Fibre Channel zone driver to manage FC zoning in Cisco SAN fabrics.

```
Version history:
  1.0 - Initial Cisco FC zone driver
  1.1 - Added friendly zone name support
```

General Considerations

Cinder allows you to integrate various storage solutions into your OpenStack cloud. It does this by providing a stable interface for hardware providers to write *drivers* that allow you to take advantage of the various features that their solutions offer.

Supported drivers

In order to make it easier for you to assess the stability and quality of a particular vendors driver, The Cinder team has introduced the concept of a **supported** driver. These are drivers that:

- have an identifiable *driver maintainer*
- are included in the Cinder source code repository
- use the upstream Cinder bug tracking mechanism
- support the Cinder *Required Driver Functions*
- maintain a third-party Continuous Integration system that runs the OpenStack Tempest test suite against their storage devices
  - this must be done for every Cinder commit, and the results must be reported to the OpenStack Gerrit code review interface
  - for details, see Driver Testing

In summary, there are two important aspects to a driver being considered as **supported**:

- the code meets the Cinder driver specifications (so you know it should integrate properly with Cinder)
- the driver code is continually tested against changes to Cinder (so you know that the code actually does integrate properly with Cinder)

The second point is particularly important because changes to Cinder can impact the drivers in two ways:

- A Cinder change may introduce a bug that only affects a particular driver or drivers (this could be because many drivers implement functionality well beyond the Required Driver Functions). With a properly running and reporting third-party CI system, such a bug can be detected at the code review stage.

- A Cinder change may exercise a new code path that exposes a driver bug that had previously gone undetected. A properly running third-party CI system will detect this and alert the driver maintainer that there is a problem.

New Driver CI Requirements

When adding a new driver, the following requirements are made of the driver and its associated 3rd Party CI system:

- CI_WIKI_NAME correct in driver properties
- CI wiki page exists under https://wiki.openstack.org/wiki/ThirdPartySystems
- Email ping to contact in wiki page receives a pong
- Recheck trigger functioning correctly
- CI is responding on the new driver patch
- CI is responding on other cinder patches
- CI is responding on os-brick patches
- CI runs all cinder-tempest-plugin tests
- CI result is accessible

Failure of any one of these requirements will preclude a new driver from being accepted into the Cinder project.

Driver Compliance

The current policy for CI compliance is:

- CIs must report on every patch, whether the code change is in their own driver code or not
- The CI comments must be properly formatted to show up in the CI summary in Gerrit

Non-compliant drivers will be tagged as unsupported if:

- No CI success reporting occurs within a two week span
- The CI is found to not be testing the expected driver (CI runs using the default LVM driver, etc.)
- Other issues are found but failed to be addressed in a timely manner

CI results are reviewed on a regular basis and if found non-compliant, a driver patch is submitted flagging it as unsupported. This can occur at any time during the development cycle. A driver can be returned to supported status as soon as the CI problem is corrected.

We do a final compliance check around the third milestone of each release. If a driver is marked as unsupported, vendors have until the time of the first Release Candidate tag (two weeks after the third milestone) to become compliant, in which case the patch flagging the driver as unsupported can be reverted. Otherwise, the driver will be considered unsupported in the release.

The CI results are currently posted here: http://cinderstats.ivehearditbothways.com/cireport.txt

Unsupported drivers

A driver is marked as unsupported when it is out of compliance.

Such a driver will log a warning message to be logged in the cinder-volume log stating that it is unsupported and deprecated for removal.

In order to use an unsupported driver, an operator must set the configuration option `enable_unsupported_driver=True` in the drivers configuration section of `cinder.conf` or the Cinder service will fail to load.

If the issue is not corrected before the next release, the driver will be eligible for removal from the Cinder code repository per the standard OpenStack deprecation policy.

If the issue *is* corrected before the next release and the team maintaining the driver in question submits a patch marking the driver as supported, that patch is eligible (at the discretion of the cinder stable maintenance team) for backport to the *most recent stable branch*.

Note

The idea behind backporting supported status is that reinstatement should happen very early in the next development cycle after the driver has been marked unsupported. For example, a driver is marked unsupported in the Victoria release but CI issues are addressed early in the Wallaby development cycle; the patch marking the driver may then be proposed to `stable/victoria`. Thus the patch will be included in the first stable release of Victoria, and operators upgrading from Ussuri to this release will not have to change their configuration files.

Note the at the discretion of the cinder stable maintenance team qualification. One reason for this is that the third party CI systems typically run only on changes to the development branch. Thus if a drivers CI is restored early in the development cycle when there have not been many code changes yet, the CI passing in the development branch can be interpreted as a proxy for CI in the most recent stable branch. Obviously, this interpretation becomes increasingly invalid as the development cycle progresses. Further, this interpretation does not extend to older stable branches.

Driver Removal

(Added January 2020)

As stated above, an unsupported driver is eligible for removal during the development cycle following the release in which it was marked unsupported. (For example, a driver marked unsupported in the Ussuri release is eligible for removal during the development cycle leading up to the Victoria release.)

During the Ussuri development cycle, the Cinder team decided that drivers eligible for removal, at the discretion of the team, may remain in the code repository *as long as they continue to pass OpenStack CI testing*. When such a driver blocks the CI check or gate, it will be removed immediately. (This does not violate the OpenStack deprecation policy because such a drivers deprecation period began when it was marked as unsupported.)

Note

Why the at the discretion of the team qualification? Some vendors may announce that they have no intention of continuing to support a driver. In that case, the Cinder team reserves the right to remove the driver as soon as the deprecation period has passed.

Thus, unsupported drivers *may* remain in the code repository for multiple releases following their declaration as unsupported. Operators should therefore take into account the length of time a driver has been marked unsupported when deciding to deploy an unsupported driver. This is because as an unmaintained driver ages, updates and bugfixes to libraries and other software it depends on may cause the driver to fail unit and functional tests, making it subject to immediate removal.

The intent of this policy revision is twofold. First, it gives vendors a longer grace period in which to make the necessary changes to have their drivers reinstated as supported. Second, keeping these drivers in-tree longer should make life easier for operators who have deployed storage backends with drivers that have been marked as unsupported. Operators should keep the above points in mind, however, when deploying such a driver.

Current Cinder Drivers

The Cinder team maintains a page of the current drivers and what exactly they support in the *Driver Support Matrix*.

You may find more details about the current drivers on the *Available Drivers* page.

Additionally, the configuration reference for each driver provides even more information. See *Volume drivers*.

3.3.3 Command-Line Interface Reference

In this section you will find information on Cinders command line utilities.

Cinder Management Commands

These commands are used to manage existing installations. They are designed to be run by operators in an environment where they have direct access to the Cinder database.

cinder-manage

Control and manage OpenStack block storage

Author
openstack-discuss@lists.openstack.org

Copyright
OpenStack Foundation

Manual section
1

Manual group
cloud computing

SYNOPSIS

cinder-manage <category> <action> [<args>]

DESCRIPTION

cinder-manage provides control of cinder database migration, and provides an interface to get information about the current state of cinder. More information about OpenStack Cinder is available at OpenStack Cinder.

OPTIONS

The standard pattern for executing a cinder-manage command is: `cinder-manage <category> <command> [<args>]`

For example, to obtain a list of the cinder services currently running: `cinder-manage service list`

Run without arguments to see a list of available command categories: `cinder-manage`

The categories are listed below, along with detailed descriptions.

You can also run with a category argument such as db to see a list of all commands in that category: `cinder-manage db`

These sections describe the available categories and arguments for cinder-manage.

Cinder Quota

Cinder quotas sometimes run out of sync, and while there are some mechanisms in place in Cinder that, with the proper configuration, try to do a resync of the quotas, they are not perfect and are susceptible to race conditions, so they may result in less than perfect accuracy in refreshed quotas.

The cinder-manage quota commands are meant to help manage these issues while allowing a finer control of when and what quotas are fixed.

Checking if quotas and reservations are correct.

`cinder-manage quota check [-h] [--project-id PROJECT_ID]`

Accepted arguments are:

```
--project-id PROJECT_ID
                        The ID of the project where we want to sync the quotas
                        (defaults to all projects).
```

This command checks quotas and reservations, for a specific project (passing `--project-id`) or for all projects, to see if they are out of sync.

The check will also look for duplicated entries.

One way to use this action in combination with the sync action is to run the check for all projects, take note of those that are out of sync, and the sync them one by one at intervals to allow cinder to operate semi-normally.

Fixing quotas and reservations

`cinder-manage quota sync [-h] [--project-id PROJECT_ID]`

Accepted arguments are:

```
--project-id PROJECT_ID
                        The ID of the project where we want to sync the quotas
                        (defaults to all projects).
```

This command refreshes existing quota usage and reservation count for a specific project or for all projects.

The refresh will also remove duplicated entries.

This operation is best executed when Cinder is not running, but it can be run with cinder services running as well.

A different transaction is used for each projects quota sync, so an action failure will only rollback the current projects changes.

Cinder Db

`cinder-manage db version`

Print the current database version.

`cinder-manage db sync [--bump-versions] [version]`

Sync the database up to the most recent version. This is the standard way to create the db as well.

This command interprets the following options when it is invoked:

version Database version

--bump-versions Update RPC and Objects versions when doing offline upgrades, with this we no longer need to restart the services twice after the upgrade to prevent ServiceTooOld exceptions.

`cinder-manage db purge [<number of days>]`

Purge database entries that are marked as deleted, that are older than the number of days specified.

`cinder-manage db online_data_migrations [--max_count <n>]`

Perform online data migrations for database upgrade between releases in batches.

This command interprets the following options when it is invoked:

```
--max_count     Maximum number of objects to migrate. If not specified, all
                possible migrations will be completed, in batches of 50 at a
                time.
```

Returns exit status 0 if no (further) updates are possible, 1 if the `--max_count` option was used and some updates were completed successfully (even if others generated errors), 2 if some updates generated errors and no other migrations were able to take effect in the last batch attempted, or 127 if invalid input is provided (e.g. non-numeric max-count).

This command should be run after upgrading the database schema. If it exits with partial updates (exit status 1) it should be called again, even if some updates initially generated errors, because some updates may depend on others having completed. If it exits with status 2, intervention is required to resolve the issue causing remaining updates to fail. It should be considered successfully completed only when the exit status is 0.

Cinder Volume

`cinder-manage volume delete <volume_id>`

Delete a volume without first checking that the volume is available.

`cinder-manage volume update_host --currenthost <current host> --newhost <new host>`

Updates the host name of all volumes currently associated with a specified host.

`cinder-manage volume update_service`

When upgrading cinder, new service entries are created in the database as the existing cinder-volume host(s) are upgraded. In some cases, rows in the volumes table keep references to the old service, which can prevent the old services from being deleted when the database is purged. This command makes sure that all volumes have updated service references for all volumes on all cinder-volume hosts.

Cinder Host

`cinder-manage host list [<zone>]`

Displays a list of all physical hosts and their zone. The optional zone argument allows the list to be filtered on the requested zone.

Cinder Service

`cinder-manage service list`

Displays a list of all cinder services and their host, zone, status, state and when the information was last updated.

`cinder-manage service remove <service> <host>`

Removes a specified cinder service from a specified host.

Cinder Backup

`cinder-manage backup list`

Displays a list of all backups (including ones in progress) and the host on which the backup operation is running.

`cinder-manage backup update_backup_host --currenthost <current host> --newhost <new host>`

Updates the host name of all backups currently associated with a specified host.

Cinder Version

`cinder-manage version list`

Displays the codebase version cinder is running upon.

Cinder Config

`cinder-manage config list [<param>]`

Displays the current configuration parameters (options) for Cinder. The optional flag parameter may be used to display the configuration of one parameter.

Cinder Util

`cinder-manage util clean_locks [-h] [--services-offline]`

Clean file locks on the current host that were created and are used by drivers and cinder services for volumes, snapshots, and the backup service on the current host.

Should be run on any host where we are running a Cinder service (API, Scheduler, Volume, Backup) and can be run with the Cinder services running or stopped.

If the services are running it will check existing resources in the Cinder database in order to only remove resources that are no longer present (its safe to delete the files).

For backups, the way to know if we can remove the startup lock is by checking if the PGRP in the file name is currently running cinder-backup.

Deleting locks while the services are offline is faster as theres no need to check the database or the running processes.

Default assumes that services are online, must pass `--services-offline` to specify that they are offline.

The common use case for running the command with `--services-offline` is to be called on startup as a service unit before any cinder service is started. Command will be usually called without the `--services-offline` parameter manually or from a cron job.

Warning

Passing `--services-offline` when the Cinder services are still running breaks the locking mechanism and can lead to undesired behavior in ongoing Cinder operations.

Note

This command doesnt clean DLM locks (except when using file locks), as those dont leave lock leftovers.

FILES

The cinder.conf file contains configuration information in the form of python-gflags.

The cinder-manage.log file logs output from cinder-manage.

SEE ALSO

- OpenStack Cinder

BUGS

- Cinder is hosted on Launchpad so you can view current bugs at Bugs : Cinder

cinder-status

CLI interface for cinder status commands

Author
openstack-discuss@lists.openstack.org

Copyright
OpenStack Foundation

Manual section
1

Manual group
cloud computing

Synopsis

```
cinder-status <category> <command> [<args>]
```

Description

cinder-status is a tool that provides routines for checking the status of a Cinder deployment.

Options

The standard pattern for executing a **cinder-status** command is:

```
cinder-status <category> <command> [<args>]
```

Run without arguments to see a list of available command categories:

```
cinder-status
```

Categories are:

- upgrade

Detailed descriptions are below.

You can also run with a category argument such as upgrade to see a list of all commands in that category:

```
cinder-status upgrade
```

These sections describe the available categories and arguments for **cinder-status**.

Upgrade

cinder-status upgrade check
Performs a release-specific readiness check before restarting services with new code. This command expects to have complete configuration and access to the database. It may also make requests to other services REST API via the Keystone service catalog.

Return Codes

Return code	Description
0	All upgrade readiness checks passed successfully and there is nothing to do.
1	At least one check encountered an issue and requires further investigation. This is considered a warning but the upgrade may be OK.
2	There was an upgrade status check failure that needs to be investigated. This should be considered something that stops an upgrade.
255	An unexpected error occurred.

History of Checks

14.0.0 (Stein)

- Check added to ensure the backup_driver setting is using the full driver class path and not just the module path.
- Checks for the presence of a **policy.json** file have been added to warn if policy changes should be present in a **policy.yaml** file.
- Ensure that correct volume_driver path is used for Windows iSCSI driver.
- Ensure that none of the volume drivers removed in Stein are enabled. Please note that if a driver is in **cinder.conf** but not in the `enabled_drivers` config option this check will not catch the problem. If you have used the CoprHD, ITRI Disco or HGST drivers in the past you should ensure that any data from these backends is transferred to a supported storage array before upgrade.

15.0.0 (Train)

- Check added to make operators aware of new finer-grained configuration options affecting the periodicity of various Cinder tasks. Triggered when the `periodic_interval` option is not set to its default value.
- Added check for use of deprecated `cinder.quota.NestedDbQuotaDriver`.

See Also

- OpenStack Cinder

Bugs

- Cinder bugs are managed at Launchpad

Additional Tools and Information

Manage volumes

A volume is a detachable block storage device, similar to a USB hard drive. You can attach a volume to only one instance. Use the `openstack` client commands to create and manage volumes.

Create a volume

This example creates a `my-new-volume` volume based on an image.

1. List images, and note the ID of the image that you want to use for your volume:

```
$ openstack image list
+--------------------------------------+---------------------------------+
| ID                                   | Name                            |
+--------------------------------------+---------------------------------+
| 8bf4dc2a-bf78-4dd1-aefa-f3347cf638c8 | cirros-0.3.5-x86_64-uec         |
| 9ff9bb2e-3a1d-4d98-acb5-b1d3225aca6c | cirros-0.3.5-x86_64-uec-kernel  |
| 4b227119-68a1-4b28-8505-f94c6ea4c6dc | cirros-0.3.5-x86_64-uec-ramdisk |
+--------------------------------------+---------------------------------+
```

2. List the availability zones, and note the ID of the availability zone in which you want to create your volume:

```
$ openstack availability zone list --volume
+-----------+-------------+
| Zone Name | Zone Status |
+-----------+-------------+
| nova      | available   |
+-----------+-------------+
```

3. Create a volume with 8 gibibytes (GiB) of space, and specify the availability zone and image:

```
$ openstack volume create --image 8bf4dc2a-bf78-4dd1-aefa-f3347cf638c8 \
  --size 8 --availability-zone nova my-new-volume

+------------------------------+--------------------------------------+
| Property                     | Value                                |
+------------------------------+--------------------------------------+
| attachments                  | []                                   |
| availability_zone            | nova                                 |
| bootable                     | false                                |
| consistencygroup_id          | None                                 |
| created_at                   | 2016-09-23T07:52:42.000000           |
| description                  | None                                 |
| encrypted                    | False                                |
| id                           | bab4b0e0-ce3d-4d57-bf57-3c51319f5202 |
| metadata                     | {}                                   |
| multiattach                  | False                                |
| name                         | my-new-volume                        |
| os-vol-tenant-attr:tenant_id | 3f670abbe9b34ca5b81db6e7b540b8d8     |
| replication_status           | disabled                             |
| size                         | 8                                    |
| snapshot_id                  | None                                 |
| source_volid                 | None                                 |
| status                       | creating                             |
| updated_at                   | None                                 |
| user_id                      | fe19e3a9f63f4a14bd4697789247bbc5     |
| volume_type                  | lvmdriver-1                          |
+------------------------------+--------------------------------------+
```

4. To verify that your volume was created successfully, list the available volumes:

```
$ openstack volume list
+--------------------------------------+---------------+-----------+------
↪+-------------+
| ID                                   | Name          |  Status   | Size␣
↪| Attached to |
+--------------------------------------+---------------+-----------+------
↪+-------------+
| bab4b0e0-ce3d-4d57-bf57-3c51319f5202 | my-new-volume | available | 8   ␣
↪|             |
+--------------------------------------+---------------+-----------+------
↪+-------------+
```

If your volume was created successfully, its status is `available`. If its status is `error`, you might have exceeded your quota.

Volume Types

Cinder supports these three ways to specify `volume type` during volume creation.

1. volume_type
2. cinder_img_volume_type (via glance image metadata)
3. default volume type (via project defaults or cinder.conf)

volume-type

User can specify *volume type* when creating a volume.

```
$ openstack volume create --type <volume-type> ...
```

cinder_img_volume_type

If glance image has `cinder_img_volume_type` property, Cinder uses this parameter to specify `volume type` when creating a volume.

Choose glance image which has `cinder_img_volume_type` property and create a volume from the image.

```
$ openstack image list
+----------------------------------------+---------------------------------+----
↪----+
| ID                                     | Name                            |␣
↪Status |
+----------------------------------------+---------------------------------+----
↪----+
| 376bd633-c9c9-4c5d-a588-342f4f66d086 | cirros-0.3.5-x86_64-uec         |␣
↪active |
| 2c20fce7-2e68-45ee-ba8d-beba27a91ab5 | cirros-0.3.5-x86_64-uec-ramdisk |␣
↪active |
| a5752de4-9faf-4c47-acbc-78a5efa7cc6e | cirros-0.3.5-x86_64-uec-kernel  |␣
↪active |
```

(continues on next page)

(continued from previous page)

```
+------------------------------------+---------------------------------+----
↪----+

$ openstack image show 376bd633-c9c9-4c5d-a588-342f4f66d086
+-----------------------+----------------------------------------------------
↪--+
| Field                 | Value                                              ␣
↪  |
+-----------------------+----------------------------------------------------
↪--+
| checksum              | eb9139e4942121f22bbc2afc0400b2a                    ␣
↪  |
| cinder_img_volume_type | nfstype                                           ␣
↪  |
| container_format      | ami                                                ␣
↪  |
| created_at            | 2016-10-13T03:28:55Z                               ␣
↪  |
| disk_format           | ami                                                ␣
↪  |
| file                  | /v2/images/376bd633-c9c9-4c5d-a588-342f4f66d086/
↪file |
| id                    | 376bd633-c9c9-4c5d-a588-342f4f66d086               ␣
↪  |
| min_disk              | 0                                                  ␣
↪  |
| min_ram               | 0                                                  ␣
↪  |
| name                  | cirros-0.3.5-x86_64-uec                            ␣
↪  |
| owner                 | 88ba456e3a884c318394737765e0ef4d                   ␣
↪  |
| properties            | kernel_id='a5752de4-9faf-4c47-acbc-78a5efa7cc6e', ␣
↪  |
|                       | ramdisk_id='2c20fce7-2e68-45ee-ba8d-beba27a91ab5' ␣
↪  |
| protected             | False                                              ␣
↪  |
| schema                | /v2/schemas/image                                  ␣
↪  |
| size                  | 25165824                                           ␣
↪  |
| status                | active                                             ␣
↪  |
| tags                  |                                                    ␣
↪  |
| updated_at            | 2016-10-13T03:28:55Z                               ␣
↪  |
| virtual_size          | None                                               ␣
```

(continues on next page)

(continued from previous page)

```
↪  |
| visibility             | public                                                    ␣
↪  |
+------------------------+-----------------------------------------------------------
↪--+

$ openstack volume create --image 376bd633-c9c9-4c5d-a588-342f4f66d086 \
    --size 1 --availability-zone nova test
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| attachments         | []                                   |
| availability_zone   | nova                                 |
| bootable            | false                                |
| consistencygroup_id | None                                 |
| created_at          | 2016-10-13T06:29:53.688599           |
| description         | None                                 |
| encrypted           | False                                |
| id                  | e6e6a72d-cda7-442c-830f-f306ea6a03d5 |
| multiattach         | False                                |
| name                | test                                 |
| properties          |                                      |
| replication_status  | disabled                             |
| size                | 1                                    |
| snapshot_id         | None                                 |
| source_volid        | None                                 |
| status              | creating                             |
| type                | nfstype                              |
| updated_at          | None                                 |
| user_id             | 33fdc37314914796883706b33e587d51     |
+---------------------+--------------------------------------+
```

default volume type

If above parameters are not set, cinder uses default volume type during volume creation.

The effective default volume type (whether it be project default or default_volume_type) can be checked with the following command:

```
$ openstack volume type list --default
```

There are two ways to set the default volume type:

1) Project specific defaults
2) default_volume_type defined in cinder.conf

Project specific defaults (available since mv 3.62 or higher)

Project specific defaults can be managed using the Default Volume Types API It is set on a per project basis and has a higher priority over default_volume_type defined in cinder.conf

default_volume_type

If the project specific default is not set then default_volume_type configured in cinder.conf is used to create volumes.

Example cinder.conf file configuration.

```
[default]
default_volume_type = lvmdriver-1
```

Attach a volume to an instance

1. Attach your volume to a server, specifying the server ID and the volume ID:

```
$ openstack server add volume 84c6e57d-a6b1-44b6-81eb-fcb36afd31b5 \
  573e024d-5235-49ce-8332-be1576d323f8 --device /dev/vdb
```

2. Show information for your volume:

```
$ openstack volume show 573e024d-5235-49ce-8332-be1576d323f8
```

The output shows that the volume is attached to the server with ID `84c6e57d-a6b1-44b6-81eb-fcb36afd31b5`, is in the nova availability zone, and is bootable.

```
+------------------------------+------------------------------------------
↪-----+
| Field                        | Value                                    
↪     |
+------------------------------+------------------------------------------
↪-----+
| attachments                  | [{u'device': u'/dev/vdb',                
↪     |
|                              |        u'server_id': u'84c6e57d-a        
↪     |
|                              |          u'id': u'573e024d-...           
↪     |
|                              |        u'volume_id': u'573e024d...       
↪     |
| availability_zone            | nova                                     
↪     |
| bootable                     | true                                     
↪     |
| consistencygroup_id          | None                                     
↪     |
| created_at                   | 2016-10-13T06:08:07.000000               
↪     |
| description                  | None                                     
```

(continues on next page)

(continued from previous page)

```
↪      |
| encrypted                    | False                                ␣
↪      |
| id                           | 573e024d-5235-49ce-8332-be1576d323f8 ␣
↪      |
| multiattach                  | False                                ␣
↪      |
| name                         | my-new-volume                        ␣
↪      |
| properties                   |                                      ␣
↪      |
| replication_status           | disabled                             ␣
↪      |
| size                         | 8                                    ␣
↪      |
| snapshot_id                  | None                                 ␣
↪      |
| source_volid                 | None                                 ␣
↪      |
| status                       | in-use                               ␣
↪      |
| type                         | lvmdriver-1                          ␣
↪      |
| updated_at                   | 2016-10-13T06:08:11.000000           ␣
↪      |
| user_id                      | 33fdc37314914796883706b33e587d51     ␣
↪      |
+------------------------------+--------------------------------------
↪-----+
```

Detach a volume from an instance

1. Detach your volume from a server, specifying the server ID and the volume ID:

```
$ openstack server remove volume 84c6e57d-a6b1-44b6-81eb-fcb36afd31b5 \
  573e024d-5235-49ce-8332-be1576d323f8
```

2. Show information for your volume:

```
$ openstack volume show 573e024d-5235-49ce-8332-be1576d323f8
```

The output shows that the volume is no longer attached to the server:

```
+------------------------------+--------------------------------------
↪-----+
| Field                        | Value                                ␣
↪      |
+------------------------------+--------------------------------------
↪-----+
| attachments                  | []                                   ␣
```

(continues on next page)

(continued from previous page)

```
↪       |
| availability_zone            | nova                                   ␣
↪       |
| bootable                     | true                                   ␣
↪       |
| consistencygroup_id          | None                                   ␣
↪       |
| created_at                   | 2016-10-13T06:08:07.000000             ␣
↪       |
| description                  | None                                   ␣
↪       |
| encrypted                    | False                                  ␣
↪       |
| id                           | 573e024d-5235-49ce-8332-be1576d323f8   ␣
↪       |
| multiattach                  | False                                  ␣
↪       |
| name                         | my-new-volume                          ␣
↪       |
| properties                   |                                        ␣
↪       |
| replication_status           | disabled                               ␣
↪       |
| size                         | 8                                      ␣
↪       |
| snapshot_id                  | None                                   ␣
↪       |
| source_volid                 | None                                   ␣
↪       |
| status                       | in-use                                 ␣
↪       |
| type                         | lvmdriver-1                            ␣
↪       |
| updated_at                   | 2016-10-13T06:08:11.000000             ␣
↪       |
| user_id                      | 33fdc37314914796883706b33e587d51       ␣
↪       |
+------------------------------+-----------------------------------------
↪-----+
```

Delete a volume

1. To delete your volume, you must first detach it from the server. To detach the volume from your server and check for the list of existing volumes, see steps 1 and 2 in *Resize_a_volume*.

   Delete the volume using either the volume name or ID:

   ```
   $ openstack volume delete my-new-volume
   ```

   This command does not provide any output.

2. List the volumes again, and note that the status of your volume is `deleting`:

```
$ openstack volume list
+----------------+-----------------+-----------+------+-------------+
|       ID       |   Name          |  Status   | Size | Attached to |
+----------------+-----------------+-----------+------+-------------+
| 573e024d-52... |  my-new-volume  |  deleting |  8   |             |
| bd7cf584-45... | my-bootable-vol | available |  8   |             |
+----------------+-----------------+-----------+------+-------------+
```

When the volume is fully deleted, it disappears from the list of volumes:

```
$ openstack volume list
+----------------+-----------------+-----------+------+-------------+
|       ID       |   Name          |  Status   | Size | Attached to |
+----------------+-----------------+-----------+------+-------------+
| bd7cf584-45... | my-bootable-vol | available |  8   |             |
+----------------+-----------------+-----------+------+-------------+
```

Resize a volume

1. To resize your volume, you must first detach it from the server if the volume driver does not support in-use extend. (See *Extend_attached_volume*.) To detach the volume from your server, pass the server ID and volume ID to the following command:

```
$ openstack server remove volume 84c6e57d-a6b1-44b6-81eb-fcb36afd31b5 
↪573e024d-5235-49ce-8332-be1576d323f8
```

This command does not provide any output.

2. List volumes:

```
$ openstack volume list
+----------------+-----------------+-----------+------+-------------+
|       ID       |   Name          |  Status   | Size | Attached to |
+----------------+-----------------+-----------+------+-------------+
| 573e024d-52... |  my-new-volume  | available |  8   |             |
| bd7cf584-45... | my-bootable-vol | available |  8   |             |
+----------------+-----------------+-----------+------+-------------+
```

Note that the volume is now available.

3. Resize the volume by passing the volume ID and the new size (a value greater than the old one) as parameters:

```
$ openstack volume set 573e024d-5235-49ce-8332-be1576d323f8 --size 10
```

This command does not provide any output. Note: The volume status `reserved` is not a valid state for an extend operation.

Note

When extending an LVM volume with a snapshot, the volume will be deactivated. The reactivation is automatic unless `auto_activation_volume_list` is defined in `lvm.conf`. See `lvm.conf` for more information.

Extend attached volume

Starting from microversion 3.42, it is also possible to extend an attached volume with status `in-use`, depending upon policy settings and the capabilities of the backend storage. Sufficient amount of storage must exist to extend the volume.

1. Resize the volume by passing the microversion,the volume ID, and the new size (a value greater than the old one) as parameters:

   ```
   $ openstack --os-volume-api-version 3.42 volume set 573e024d-5235-49ce-
   ↪8332-be1576d323f8 --size 10
   ```

   This command does not provide any output.

Migrate a volume

As an administrator, you can migrate a volume with its data from one location to another in a manner that is transparent to users and workloads. You can migrate only detached volumes with no snapshots.

Possible use cases for data migration include:

- Bring down a physical storage device for maintenance without disrupting workloads.
- Modify the properties of a volume.
- Free up space in a thinly-provisioned back end.

Migrate a volume with the **openstack volume migrate** command, as shown in the following example:

```
$ openstack volume migrate [-h] --host <host> [--force-host-copy]
                                [--lock-volume] <volume>
```

The arguments for this command are:

host
: The destination host in the format *host@backend-name#pool*.

volume
: The ID of the volume to migrate.

force-host-copy
: Disables any driver optimizations and forces the data to be copied by the host.

lock-volume
: Prevents other processes from aborting the migration.

> **Note**
>
> If the volume has snapshots, the specified host destination cannot accept the volume. If the user is not an administrator, the migration fails.

Transfer a volume

You can transfer a volume from one owner to another by using the **openstack volume transfer request create** command. The volume donor, or original owner, creates a transfer request and sends the created transfer ID and authorization key to the volume recipient. The volume recipient, or new owner, accepts the transfer by using the ID and key.

Starting with the Rocky release, Cinder changes the API behavior for the v2 and v3 API up to microversion 3.55. Snapshots will be transferred with the volume by default. That means if the volume has some snapshots, when a user transfers a volume from one owner to another, then those snapshots will be transferred with the volume as well.

Starting with microversion 3.55 and later, Cinder supports the ability to transfer volume without snapshots. If users dont want to transfer snapshots, they need to specify the new optional argument *no-snapshots*.

Starting with microversion 3.70 and later, Cinder supports the ability to transfer encrypted volumes. Snapshots must be transferred with the volume.

Note

The procedure for volume transfer is intended for projects (both the volume donor and recipient) within the same cloud.

Use cases include:

- Create a custom bootable volume or a volume with a large data set and transfer it to a customer.
- For bulk import of data to the cloud, the data ingress system creates a new Block Storage volume, copies data from the physical device, and transfers device ownership to the end user.

Create a volume transfer request

1. While logged in as the volume donor, list the available volumes:

```
$ openstack volume list
+-----------------+-----------------+-----------+------+-------------+
|       ID        |   Name          |  Status   | Size | Attached to |
+-----------------+-----------------+-----------+------+-------------+
| 72bfce9f-cac... |       None      |   error   |  1   |             |
| a1cdace0-08e... |       None      | available |  1   |             |
+-----------------+-----------------+-----------+------+-------------+
```

2. As the volume donor, request a volume transfer authorization code for a specific volume:

```
$ openstack volume transfer request create [--no-snapshots] <volume>
```

The arguments to be passed are:

`<volume>` Name or ID of volume to transfer.

`--no-snapshots` Transfer the volume without snapshots.

The volume must be in an `available` state or the request will be denied. If the transfer request is valid in the database (that is, it has not expired or been deleted), the volume is placed in an `awaiting-transfer` state. For example:

```
$ openstack volume transfer request create a1cdace0-08e4-4dc7-b9dc-
↪457e9bcfe25f
```

The output shows the volume transfer ID in the `id` row and the authorization key.

```
+------------+--------------------------------------+
| Field      | Value                                |
+------------+--------------------------------------+
| auth_key   | 0a59e53630f051e2                     |
| created_at | 2016-11-03T11:49:40.346181           |
| id         | 34e29364-142b-4c7b-8d98-88f765bf176f |
| name       | None                                 |
| volume_id  | a1cdace0-08e4-4dc7-b9dc-457e9bcfe25f |
+------------+--------------------------------------+
```

Note

Optionally, you can specify a name for the transfer by using the `--name transferName` parameter.

Note

While the `auth_key` property is visible in the output of `openstack volume transfer request create VOLUME_ID`, it will not be available in subsequent `openstack volume transfer request show TRANSFER_ID` command.

1. Send the volume transfer ID and authorization key to the new owner (for example, by email).
2. View pending transfers:

```
$ openstack volume transfer request list
+--------------------------------------+-------------------------------
↪----+------+
|                  ID                  |             Volume            ␣
↪    | Name |
+--------------------------------------+-------------------------------
↪----+------+
| 6e4e9aa4-bed5-4f94-8f76-df43232f44dc | a1cdace0-08e4-4dc7-b9dc-
↪457e9bcfe25f | None |
+--------------------------------------+-------------------------------
↪----+------+
```

3. After the volume recipient, or new owner, accepts the transfer, you can see that the transfer is no longer available:

```
$ openstack volume transfer request list
+----+-----------+------+
| ID | Volume ID | Name |
+----+-----------+------+
+----+-----------+------+
```

Accept a volume transfer request

1. As the volume recipient, you must first obtain the transfer ID and authorization key from the original owner.
2. Accept the request:

```
$ openstack volume transfer request accept transferID authKey
```

For example:

```
$ openstack volume transfer request accept 6e4e9aa4-bed5-4f94-8f76-
↪df43232f44dc b2c8e585cbc68a80
+-----------+--------------------------------------+
|  Property |                Value                 |
+-----------+--------------------------------------+
|     id    | 6e4e9aa4-bed5-4f94-8f76-df43232f44dc |
|    name   |                 None                 |
| volume_id | a1cdace0-08e4-4dc7-b9dc-457e9bcfe25f |
+-----------+--------------------------------------+
```

Note

If you do not have a sufficient quota for the transfer, the transfer is refused.

Delete a volume transfer

1. List available volumes and their statuses:

```
$ openstack volume list
+-----------------+-----------------+-----------------+------+-----------
↪-+
|       ID        |   Name          |      Status     | Size | Attached␣
↪to |
+-----------------+-----------------+-----------------+------+-----------
↪-+
| 72bfce9f-cac... |       None      |      error      |  1   |          ␣
↪ |
| a1cdace0-08e... |       None      |awaiting-transfer|  1   |          ␣
↪ |
+-----------------+-----------------+-----------------+------+-----------
↪-+
```

2. Find the matching transfer ID:

```
$ openstack volume transfer request list
+--------------------------------------+--------------------------------
↪----+------+
|               ID                     |             VolumeID           ␣
↪    | Name |
+--------------------------------------+--------------------------------
```

(continues on next page)

(continued from previous page)

```
↪----+-------+
| a6da6888-7cdf-4291-9c08-8c1f22426b8a | a1cdace0-08e4-4dc7-b9dc-
↪457e9bcfe25f | None |
+--------------------------------------+----------------------------------
↪----+-------+
```

3. Delete the volume:

```
$ openstack volume transfer request delete <transfer>
```

<transfer>
Name or ID of transfer to delete.

For example:

```
$ openstack volume transfer request delete a6da6888-7cdf-4291-9c08-
↪8c1f22426b8a
```

4. Verify that transfer list is now empty and that the volume is again available for transfer:

```
$ openstack volume transfer request list
+----+-----------+------+
| ID | Volume ID | Name |
+----+-----------+------+
+----+-----------+------+
```

```
$ openstack volume list
+-----------------+-----------------+-----------------+------+------------
↪-+
|       ID        |   Name          |      Status     | Size | Attached␣
↪to |
+-----------------+-----------------+-----------------+------+------------
↪-+
| 72bfce9f-cac... |       None      |      error      |  1   |            ␣
↪ |
| a1cdace0-08e... |       None      |    available    |  1   |            ␣
↪ |
+-----------------+-----------------+-----------------+------+------------
↪-+
```

Manage and unmanage a snapshot

A snapshot is a point in time version of a volume. As an administrator, you can manage and unmanage snapshots.

Manage a snapshot

Manage a snapshot with the **`openstack volume snapshot set`** command:

```
$ openstack volume snapshot set [-h]
                                [--name <name>]
                                [--description <description>]
                                [--no-property]
                                [--property <key=value>]
                                [--state <state>]
                                <snapshot>
```

The arguments to be passed are:

--name <name>
New snapshot name

--description <description>
New snapshot description

--no-property
Remove all properties from <snapshot> (specify both no-property and property to remove the current properties before setting new properties.)

--property <key=value>
Property to add or modify for this snapshot (repeat option to set multiple properties)

--state <state>
New snapshot state. (available, error, creating, deleting, or error_deleting) (admin only) (This option simply changes the state of the snapshot in the database with no regard to actual status, exercise caution when using)

<snapshot>
Snapshot to modify (name or ID)

```
$ openstack volume snapshot set my-snapshot-id
```

Unmanage a snapshot

Unmanage a snapshot with the **openstack volume snapshot unset** command:

```
$ openstack volume snapshot unset [-h]
                                  [--property <key>]
                                  <snapshot>
```

The arguments to be passed are:

--property <key>
Property to remove from snapshot (repeat option to remove multiple properties)

<snapshot>
Snapshot to modify (name or ID).

The following example unmanages the `my-snapshot-id` image:

```
$ openstack volume snapshot unset my-snapshot-id
```

Report backend state in service list

Each of the Cinder services report a Status and a State. These are the administrative state and the runtime state, respectively.

To get a listing of all Cinder services and their states, run the command:

```
$ openstack volume service list
+------------------+-------------------+------+---------+-------+------------
↪---------------+
| Binary           | Host              | Zone | Status  | State | Updated At ␣
↪              |
+------------------+-------------------+------+---------+-------+------------
↪---------------+
| cinder-scheduler | tower             | nova | enabled | up    | 2018-03-
↪30T21:16:11.000000 |
| cinder-volume    | tower@lvmdriver-1 | nova | enabled | up    | 2018-03-
↪30T21:16:15.000000 |
| cinder-backup    | tower             | nova | enabled | up    | 2018-03-
↪30T21:16:14.000000 |
+------------------+-------------------+------+---------+-------+------------
↪---------------+
```

Manage quotas

To prevent system capacities from being exhausted without notification, you can set up quotas. Quotas are operational limits. For example, the number of gigabytes allowed for each project can be controlled so that cloud resources are optimized. Quotas can be enforced at both the project and the project-user level.

Using the command-line interface, you can manage quotas for the OpenStack Compute service, the OpenStack Block Storage service, and the OpenStack Networking service.

The cloud operator typically changes default values because a project requires more than ten volumes or 1 TB on a compute node.

Note

To view all projects, run:

```
$ openstack project list
+----------------------------------+----------+
| ID                               | Name     |
+----------------------------------+----------+
| e66d97ac1b704897853412fc8450f7b9 | admin    |
| bf4a37b885fe46bd86e999e50adad1d3 | services |
| 21bd1c7c95234fd28f589b60903606fa | tenant01 |
| f599c5cd1cba4125ae3d7caed08e288c | tenant02 |
+----------------------------------+----------+
```

To display all current users for a project, run:

```
$ openstack user list --project PROJECT_NAME
+----------------------------------+--------+
| ID                               | Name   |
+----------------------------------+--------+
| ea30aa434ab24a139b0e85125ec8a217 | demo00 |
| 4f8113c1d838467cad0c2f337b3dfded | demo01 |
+----------------------------------+--------+
```

Use `openstack quota show PROJECT_NAME` to list all quotas for a project.

Use `openstack quota set PROJECT_NAME --parameters` to set quota values.

Manage Block Storage service quotas

As an administrative user, you can update the OpenStack Block Storage service quotas for a project. You can also update the quota defaults for a new project.

Block Storage quotas

Property name	Defines the number of
gigabytes	Volume gigabytes allowed for each project.
snapshots	Volume snapshots allowed for each project.
volumes	Volumes allowed for each project.

View Block Storage quotas

Administrative users can view Block Storage service quotas.

1. Obtain the project ID:

```
$ PROJECT_ID=$(openstack project show -f value -c id PROJECT_NAME)
```

2. List the default quotas for a project:

```
$ openstack quota show --default $PROJECT_ID
+-----------------------+-------+
| Field                 | Value |
+-----------------------+-------+
| backup-gigabytes      | 1000  |
| backups               | 10    |
| cores                 | 20    |
| fixed-ips             | -1    |
| floating-ips          | 50    |
| gigabytes             | 1000  |
| gigabytes_lvmdriver-1 | -1    |
| health_monitors       | None  |
| injected-file-size    | 10240 |
| injected-files        | 5     |
| injected-path-size    | 255   |
```

(continues on next page)

(continued from previous page)

```
| instances            | 10    |
| key-pairs            | 100   |
| l7_policies          | None  |
| listeners            | None  |
| load_balancers       | None  |
| location             | None  |
| name                 | None  |
| networks             | 10    |
| per-volume-gigabytes | -1    |
| pools                | None  |
| ports                | 50    |
| project              | None  |
| project_id           | None  |
| properties           | 128   |
| ram                  | 51200 |
| rbac_policies        | 10    |
| routers              | 10    |
| secgroup-rules       | 100   |
| secgroups            | 10    |
| server-group-members | 10    |
| server-groups        | 10    |
| snapshots            | 10    |
| snapshots_lvmdriver-1 | -1    |
| subnet_pools         | -1    |
| subnets              | 10    |
| volumes              | 10    |
| volumes_lvmdriver-1  | -1    |
+----------------------+-------+
```

Note

Listing default quotas with the OpenStack command line client will provide all quotas for storage and network services. Previously, the **cinder quota-defaults** command would list only storage quotas. You can use *$PROJECT_ID* or *$PROJECT_NAME* arguments to show Block Storage service quotas. If the *$PROJECT_ID* argument returns errors in locating resources, use *$PROJECT_NAME*.

1. View Block Storage service quotas for a project:

```
$ openstack quota show --volume $PROJECT_ID
+-----------------------+-------+
| Resource              | Limit |
+-----------------------+-------+
| volumes               |    10 |
| snapshots             |    10 |
| gigabytes             |  1000 |
| backups               |    10 |
| volumes_lvmdriver-1   |    -1 |
| gigabytes_lvmdriver-1 |    -1 |
| snapshots_lvmdriver-1 |    -1 |
```

(continues on next page)

(continued from previous page)

```
| volumes___DEFAULT__    |    -1 |
| gigabytes___DEFAULT__  |    -1 |
| snapshots___DEFAULT__  |    -1 |
| groups                 |    10 |
| backup-gigabytes       |  1000 |
| per-volume-gigabytes   |    -1 |
+------------------------+-------+
```

2. Show the current usage of a per-project quota:

```
$ openstack quota show --volume --usage $PROJECT_ID
+------------------------+-------+--------+----------+
| Resource               | Limit | In Use | Reserved |
+------------------------+-------+--------+----------+
| volumes                |    10 |      1 |        0 |
| snapshots              |    10 |      0 |        0 |
| gigabytes              |  1000 |      1 |        0 |
| backups                |    10 |      0 |        0 |
| volumes_lvmdriver-1    |    -1 |      1 |        0 |
| gigabytes_lvmdriver-1  |    -1 |      1 |        0 |
| snapshots_lvmdriver-1  |    -1 |      0 |        0 |
| volumes___DEFAULT__    |    -1 |      0 |        0 |
| gigabytes___DEFAULT__  |    -1 |      0 |        0 |
| snapshots___DEFAULT__  |    -1 |      0 |        0 |
| groups                 |    10 |      0 |        0 |
| backup-gigabytes       |  1000 |      0 |        0 |
| per-volume-gigabytes   |    -1 |      0 |        0 |
+------------------------+-------+--------+----------+
```

Edit and update Block Storage service quotas

Administrative users can edit and update Block Storage service quotas.

1. To update the default quota values for the initial deployment, update the values of the *quota_** config options in the `/etc/cinder/cinder.conf` file. For more information, see the *Block Storage service configuration*.

   Note

   The values of the *quota_** config options are only used at the initial database sync in the initial deployment. If you want to change a default value for a new project, see the following.

   To update a default value for a new project, set `use_default_quota_class = True` (which is the default setting) in the *DEFAULT* section of the `/etc/cinder/cinder.conf` file, and run the command as the following.

   ```
   $ openstack quota set --class default --QUOTA_NAME QUOTA_VALUE
   ```

   Replace `QUOTA_NAME` with the quota that is to be updated, `QUOTA_VALUE` with the required new value.

2. To update Block Storage service quotas for an existing project

```
$ openstack quota set --QUOTA_NAME QUOTA_VALUE PROJECT_ID
```

Replace QUOTA_NAME with the quota that is to be updated, QUOTA_VALUE with the required new value. Use the **openstack quota show** command with PROJECT_ID, which is the required project ID.

For example:

```
$ openstack quota set --volumes 15 $PROJECT_ID
$ openstack quota show $PROJECT_ID
+-----------------------+----------------------------------+
| Field                 | Value                            |
+-----------------------+----------------------------------+
| backup-gigabytes      | 1000                             |
| backups               | 10                               |
| cores                 | 20                               |
| fixed-ips             | -1                               |
| floating-ips          | 29                               |
| gigabytes             | 1000                             |
| gigabytes_lvmdriver-1 | -1                               |
| health_monitors       | None                             |
| injected-file-size    | 10240                            |
| injected-files        | 5                                |
| injected-path-size    | 255                              |
| instances             | 10                               |
| key-pairs             | 100                              |
| l7_policies           | None                             |
| listeners             | None                             |
| load_balancers        | None                             |
| location              | None                             |
| name                  | None                             |
| networks              | 10                               |
| per-volume-gigabytes  | -1                               |
| pools                 | None                             |
| ports                 | 50                               |
| project               | e436339c7f9c476cb3120cf3b9667377 |
| project_id            | None                             |
| properties            | 128                              |
| ram                   | 51200                            |
| rbac_policies         | 10                               |
| routers               | 10                               |
| secgroup-rules        | 100                              |
| secgroups             | 10                               |
| server-group-members  | 10                               |
| server-groups         | 10                               |
| snapshots             | 10                               |
| snapshots_lvmdriver-1 | -1                               |
| subnet_pools          | -1                               |
| subnets               | 10                               |
```

(continues on next page)

(continued from previous page)

```
| volumes                | 15                                 |
| volumes_lvmdriver-1    | -1                                 |
+------------------------+------------------------------------+
```

3. To clear per-project quota limits:

```
$ openstack quota delete --volume $PROJECT_ID
```

Manage Block Storage scheduling

As an administrative user, you have some control over which volume back end your volumes reside on. You can specify affinity or anti-affinity between two volumes. Affinity between volumes means that they are stored on the same back end, whereas anti-affinity means that they are stored on different back ends.

For information on how to set up multiple back ends for Cinder, refer to *Configure multiple-storage back ends*.

Example Usages

1. Create a new volume on the same back end as Volume_A:

```
$ openstack volume create --hint same_host=Volume_A-UUID \
  --size SIZE VOLUME_NAME
```

2. Create a new volume on a different back end than Volume_A:

```
$ openstack volume create --hint different_host=Volume_A-UUID \
  --size SIZE VOLUME_NAME
```

3. Create a new volume on the same back end as Volume_A and Volume_B:

```
$ openstack volume create --hint same_host=Volume_A-UUID \
  --hint same_host=Volume_B-UUID --size SIZE VOLUME_NAME
```

Or:

```
$ openstack volume create --hint same_host="[Volume_A-UUID, \
  Volume_B-UUID]" --size SIZE VOLUME_NAME
```

4. Create a new volume on a different back end than both Volume_A and Volume_B:

```
$ openstack volume create --hint different_host=Volume_A-UUID \
  --hint different_host=Volume_B-UUID --size SIZE VOLUME_NAME
```

Or:

```
$ openstack volume create --hint different_host="[Volume_A-UUID, \
  Volume_B-UUID]" --size SIZE VOLUME_NAME
```

3.4 Additional resources

- Cinder release notes

CHAPTER

FOUR

FOR CONTRIBUTORS

Contributions to Cinder are welcome. There can be a lot of background information needed to get started. This section should help get you started. Please feel free to also ask any questions in the **#openstack-cinder** IRC channel.

4.1 Contributing to Cinder

Contents:

4.1.1 Contributor Guide

In this section you will find information on how to contribute to Cinder. Content includes architectural overviews, tips and tricks for setting up a development environment, and information on Cinders lower level programming APIs.

Getting Started

So You Want to Contribute

For general information on contributing to OpenStack, please check out the contributor guide to get started. It covers all the basics that are common to all OpenStack projects: the accounts you need, the basics of interacting with our Gerrit review system, how we communicate as a community, etc.

Below will cover the more project specific information you need to get started with the Cinder project, which is responsible for the following OpenStack deliverables:

cinder

The OpenStack Block Storage service.

code: https://opendev.org/openstack/cinder

docs: https://cinder.openstack.org

api-ref: https://docs.openstack.org/api-ref/block-storage

Launchpad: https://launchpad.net/cinder

os-brick

Shared library for managing local volume attaches.

code: https://opendev.org/openstack/os-brick

docs: https://docs.openstack.org/os-brick

Launchpad: https://launchpad.net/os-brick

python-cinderclient

Python client library for the OpenStack Block Storage API; includes a CLI shell.

code: https://opendev.org/openstack/python-cinderclient

docs: https://docs.openstack.org/python-cinderclient

Launchpad: https://launchpad.net/python-cinderclient

python-brick-cinderclient-ext

Extends the python-cinderclient library so that it can handle local volume attaches.

code: https://opendev.org/openstack/python-brick-cinderclient-ext

docs: https://docs.openstack.org/python-brick-cinderclient-ext

Launchpad: (doesnt have its own space, uses python-cinderclients)

cinderlib

Library that allows direct usage of Cinder backend drivers without cinder services.

code: https://opendev.org/openstack/cinderlib

docs: https://docs.openstack.org/cinderlib

Launchpad: https://launchpad.net/cinderlib

rbd-iscsi-client

Library that provides a REST client that talks to ceph-iscis rbd-target-api to export rbd images/volumes to an iSCSI initiator.

code: https://opendev.org/openstack/rbd-iscsi-client

docs: https://docs.openstack.org/rbd-iscsi-client

Launchpad: https://launchpad.net/rbd-iscsi-client

cinder-tempest-plugin

Contains additional Cinder tempest-based tests beyond those in the main OpenStack Integration Test Suite (tempest).

code: https://opendev.org/openstack/cinder-tempest-plugin

Launchpad: https://launchpad.net/cinder-tempest-plugin

See the `CONTRIBUTING.rst` file in each code repository for more information about contributing to that specific deliverable. Additionally, you should look over the docs links above; most components have helpful developer information specific to that deliverable. (The main cinder documentation is especially thorough in this regard and you should read through it, particularly *Background Concepts for Cinder* and *Programming HowTos and Tutorials*.)

Communication

IRC

We use IRC *a lot*. You will, too. You can find infomation about what IRC network OpenStack uses for communication (and tips for using IRC) in the Setup IRC section of the main *OpenStack Contributor Guide*.

People working on the Cinder project may be found in the `#openstack-cinder` IRC channel during working hours in their timezone. The channel is logged, so if you ask a question when no one is around, you can check the log to see if its been answered: http://eavesdrop.openstack.org/irclogs/%23openstack-cinder/

weekly meeting

Wednesdays at 14:00 UTC in the `#openstack-meeting-alt` IRC channel. Meetings are logged: http://eavesdrop.openstack.org/meetings/cinder/

More information (including some pointers on meeting etiquette and an ICS file to put the meeting on your calendar) can be found at: http://eavesdrop.openstack.org/#Cinder_Team_Meeting

The meeting agenda for a particular development cycle is kept on an etherpad. You can find a link to the current agenda from the Cinder Meetings wiki page: https://wiki.openstack.org/wiki/CinderMeetings

The last meeting of each month is held simultaneously in videoconference and IRC. Connection information is posted on the meeting agenda.

weekly bug squad meeting
This is a half-hour meeting on Wednesdays at 15:00 UTC (right after the Cinder weekly meeting) in the `#openstack-cinder` IRC channel. At this meeting, led by the Cinder Bug Deputy, we discuss new bugs that have been filed against Cinder project deliverables (and, if theres time, discuss the relevance of old bugs that havent seen any action recently). Info about the meeting is here: http://eavesdrop.openstack.org/#Cinder_Bug_Squad_Meeting

mailing list
We use the openstack-discuss@lists.openstack.org mailing list for asynchronous discussions or to communicate with other OpenStack teams. Use the prefix `[cinder]` in your subject line (its a high-volume list, so most people use email filters).

More information about the mailing list, including how to subscribe and read the archives, can be found at: http://lists.openstack.org/cgi-bin/mailman/listinfo/openstack-discuss

virtual meet-ups
From time to time, the Cinder project will have video meetings to address topics not easily covered by the above methods. These are announced well in advance at the weekly meeting and on the mailing list.

Additionally, the Cinder project has been holding two virtual mid-cycle meetings during each development cycle, roughly at weeks R-18 and R-9. These are used to discuss follow-up issues from the PTG before the spec freeze, and to assess the development status of features and priorities roughly one month before the feature freeze. The exact dates of these are announced at the weekly meeting and on the mailing list.

cinder festival of XS reviews
This is a standing video meeting held the third Friday of each month from 14:00-16:00 UTC in meetpad to review very small patches that havent yet been merged. Its held in video so we can quickly discuss issues and hand reviews back and forth. It is not recorded. Info about the meeting is here: http://eavesdrop.openstack.org/#Cinder_Festival_of_XS_Reviews

physical meet-ups
The Cinder project usually has a presence at the OpenDev/OpenStack Project Team Gathering that takes place at the beginning of each development cycle. Planning happens on an etherpad whose URL is announced at the weekly meetings and on the mailing list.

Contacting the Core Team

The cinder-core team is an active group of contributors who are responsible for directing and maintaining the Cinder project. As a new contributor, your interaction with this group will be mostly through code reviews, because only members of cinder-core can approve a code change to be merged into the code repository.

You can learn more about the role of core reviewers in the OpenStack governance documentation: https://docs.openstack.org/contributors/common/governance.html#core-reviewer

The membership list of cinder-core is maintained in gerrit: https://review.opendev.org/#/admin/groups/83,members

You can also find the members of the cinder-core team at the Cinder weekly meetings.

Project Team Lead

For each development cycle, Cinder project Active Technical Contributors (ATCs) elect a Project Team Lead who is responsible for running the weekly meetings, midcycles, and Cinder sessions at the Project Team Gathering for that cycle (and who is also ultimately responsible for everything else the project does).

- You automatically become an ATC by making a commit to one of the cinder deliverables. Other people who havent made a commit, but have contributed to the project in other ways (for example, making good bug reports) may be recognized as extra-ATCs and obtain voting privileges. If you are such a person, contact the current PTL before the Extra-ATC freeze indicated on the current development cycle schedule (which you can find from the OpenStack Releases homepage .

The current Cinder project Project Team Lead (PTL) is listed in the Cinder project reference maintained by the OpenStack Technical Committee.

All common PTL duties are enumerated in the PTL guide.

Additional responsibilities for the Cinder PTL can be found by reading through the *Managing the Development Cycle* section of the Cinder documentation.

New Feature Planning

The Cinder project uses both specs and blueprints to track new features. Heres a quick rundown of what they are and how the Cinder project uses them.

specs

Exist in the cinder-specs repository. Each spec must have a Launchpad blueprint (see below) associated with it for tracking purposes.

A spec is required for any new Cinder core feature, anything that changes the Block Storage API, or anything that entails a mass change to existing drivers.

The specs repository is: https://opendev.org/openstack/cinder-specs
It contains a `README.rst` file explaining how to file a spec.

You can read rendered specs docs at:
https://specs.openstack.org/openstack/cinder-specs/

blueprints

Exist in Launchpad, where they can be targeted to release milestones.
You file one at https://blueprints.launchpad.net/cinder

Examples of changes that can be covered by a blueprint only are:

- adding a new volume, backup, or target driver; or
- adding support for a defined capability that already exists in the base volume, backup, or target drivers

Feel free to ask in `#openstack-cinder` or at the weekly meeting if you have an idea you want to develop and youre not sure whether it requires a blueprint *and* a spec or simply a blueprint.

The Cinder project observes the following deadlines. For the current development cycle, the dates of each (and a more detailed description) may be found on the release schedule, which you can find from: https://releases.openstack.org/

- spec freeze (all specs must be approved by this date)
- new driver merge deadline
- new target driver merge deadline
- new feature status checkpoint
- driver features declaration
- third-party CI compliance checkpoint

Additionally, the Cinder project observes the OpenStack-wide deadlines, for example, final release of non-client libraries (os-brick), final release for client libraries (python-cinderclient), feature freeze, etc. These are also noted and explained on the release schedule for the current development cycle.

Task Tracking

We track our tasks in Launchpad. See the top of the page for the URL of each Cinder project deliverable.

If youre looking for some smaller, easier work item to pick up and get started on, search for the low-hanging-fruit tag in the Bugs section.

When you start working on a bug, make sure you assign it to yourself. Otherwise someone else may also start working on it, and we dont want to duplicate efforts. Also, if you find a bug in the code and want to post a fix, make sure you file a bug (and assign it to yourself!) just in case someone else comes across the problem in the meantime.

Reporting a Bug

You found an issue and want to make sure we are aware of it? You can do so in the Launchpad space for the affected deliverable:

- cinder: https://bugs.launchpad.net/cinder
- os-brick: https://bugs.launchpad.net/os-brick
- python-cinderclient: https://bugs.launchpad.net/python-cinderclient
- python-brick-cinderclient-ext: same as for python-cinderclient, but tag the bug with brick-cinderclient-ext
- cinderlib: https://bugs.launchpad.net/cinderlib
- cinder-tempest-plugin: https://bugs.launchpad.net/cinder-tempest-plugin

Getting Your Patch Merged

Before your patch can be merged, it must be *reviewed* and *approved*.

The Cinder project policy is that a patch must have two +2s before it can be merged. (Exceptions are documentation changes, which require only a single +2, and specs, for which the PTL may require more than two +2s, depending on the complexity of the proposal.) Only members of the cinder-core team can vote +2 (or -2) on a patch, or approve it.

> **Note**
>
> Although your contribution will require reviews by members of cinder-core, these arent the only people whose reviews matter. Anyone with a gerrit account can post reviews, so you can ask other developers you know to review your code and you can review theirs. (A good way to learn your way around the codebase is to review other peoples patches.)
>
> If youre thinking, Im new at this, how can I possibly provide a helpful review?, take a look at How to Review Changes the OpenStack Way.
>
> There are also some Cinder project specific reviewing guidelines in the *Code Reviews* section of the Cinder Contributor Guide.

Patches lacking unit tests are unlikely to be approved. Check out the *Testing* section of the Cinder Contributors Guide for a discussion of the kinds of testing we do with cinder.

In addition, some changes may require a release note. Any patch that changes functionality, adds functionality, or addresses a significant bug should have a release note. You can find more information about how to write a release note in the *Release notes* section of the Cinder Contributors Guide.

> Keep in mind that the best way to make sure your patches are reviewed in a timely manner is to review other peoples patches. Were engaged in a cooperative enterprise here.

If your patch has a -1 from Zuul, you should fix it right away, because people are unlikely to review a patch that is failing the CI system.

- If its a pep8 issue, the job leaves sufficient information for you to fix the problems yourself.
- If you are failing unit or functional tests, you should look at the failures carefully. These tests guard against regressions, so if your patch causing failures, you need to figure out exactly what is going on.
- The unit, functional, and pep8 tests can all be run locally before you submit your patch for review. By doing so, you can help conserve gate resources.
- Other test failures: we also run integration tests in the gate that run your changes in the context of an OpenStack deployment, where cinder and os-brick interact with users, admins, and other services. Sometimes these tests will fail, and it may not obviously be your patchs fault. Keep in mind, however, that the failure could still be a cinder issue, for which the cinder project (which includes you, as a contributor) is responsible. So please take a few minutes to look over the logs from the failing test job to see if you can identify the issue.
  - If youre not sure how to do this, ask in the `#openstack-cinder` channel (or during open discussion at the weekly cinder meeting), and someone will walk you through the basic process.
  - You can tell Zuul to do a recheck, but first:

  * Make sure you look at the jobs build history, because if the job is failing consistently, its probably due to some particular issue that must be fixed before the job will start passing again. So a recheck in this situation will just waste resources. Check the mailing list or ask in IRC or look at the comments on your patch (sometimes a reviewer will leave a note saying not to recheck until some other patch has merged).
  * When you think a recheck is appropriate, make sure you follow the OpenStack community guidelines for How to Handle Test Failures.

How long it may take for your review to get attention will depend on the current project priorities. For example, the feature freeze is at the third milestone of each development cycle, so feature patches have the highest priority just before M-3. Likewise, once the new driver freeze is in effect, new driver patches are unlikely to receive timely reviews until after the stable branch has been cut (this happens three weeks before release). Similarly, os-brick patches have review priority before the nonclient library release deadline, and cinderclient patches have priority before the client library release each cycle. These dates are clearly noted on the release schedule for the current release, which you can find from https://releases.openstack.org/

You can see whos been doing what with Cinder recently in Stackalytics: https://www.stackalytics.io/report/activity?module=cinder-group

Backporting a Fix

> **tl;dr:** Only propose a cherry pick from a *merged* commit, even if you want to backport the patch to multiple stable branches. Doing them all at once doesnt speed anything up, because the cinder-stable-maint team will **not** approve a backport to branch *n*-1 until the patch has been merged into branch *n*.

From time to time, you may find a bug thats been fixed in master, and youd like to have that fix in the release youre currently using (for example, Wallaby). What you want to do is propose a **backport** of the fix.

Note

The Cinder project observes the OpenStack Stable Branch Policy. Thus, not every change in master is backportable to the stable branches. In particular, features are *never* backportable. A really complicated bugfix may not be backportable if what it fixes is low-occurrence and theres a high risk that it may cause a regression elsewhere in the software.

How can you tell? Ask in the `#openstack-cinder` channel on IRC or during the open discussion part of the weekly Cinder team meeting.

Since we use git for source code version control, backporting is done by *cherry-picking* a change that has already been merged into one branch into another branch. The gerrit web interface makes it really easy to do this. In fact, maybe *too* easy. Here are some guidelines:

- Before you cherry-pick a change, make sure it has already **merged** to master. If the change hasnt merged yet, it may require further revision, and the commit youve cherry-picked wont be the correct commit to backport.
- Backports must be done in *reverse chronological order*. Since OpenStack releases are named alphabetically, this means reverse alphabetical order: `stable/yoga`, `stable/xena`, `stable/wallaby`, etc.

- The cherry-pick must have **merged** into the closest most recent branch before it will be considered for a branch, that is, a cherry-pick to `stable/xena` will **not** be considered until it has merged into `stable/yoga` first.
  - This is because sometimes a backport requires revision along the way. For example, different OpenStack releases support different versions of Python. So if a fix uses a language feature introduced in Python 3.8, it will merge just fine into current master (during zed development), but it will not pass unit tests in `stable/yoga` (which supports Python 3.6). Likewise, if you already cherry-picked the patch from master directly to `stable/xena`, it wont pass tests there either (because xena also supports Python 3.6).

    So its better to follow the policy and wait until the patch is merged into `stable/yoga` *before* you propose a backport to `stable/xena`.
- You can propose backports directly from git instead of using the gerrit web interface, but if you do, you must include the fact that its a cherry-pick in the commit message. Gerrit does this automatically for you *if you cherry-pick from a merged commit* (which is the only kind of commit you should cherry-pick from in Gerrit); git will do it for you if you use the `-x` flag when you do a manual cherry-pick.

  This will keep the history of this backport intact as it goes from branch to branch. We want this information to be in the commit message and to be accurate, because if the fix causes a regression (which is always possible), it will be helpful to the poor sucker who has to fix it to know where this code came from without digging through a bunch of git history.

If you have questions about any of this, or if you have a bug to fix that is only present in one of the stable branches, ask for advice in `#openstack-cinder` on IRC.

Backport CI Testing

Like all code changes, backports should undergo continuous integration testing. This is done automatically by Zuul for changes that affect the main cinder code.

> When a vendor driver patch backport is proposed, we would like to see a clear statement on
> the gerrit review that the patch has been tested in an appropriate environment.

This shouldnt be a big deal because presumably youve done local testing with your backend to ensure that the code works as expected in a stable branch; were simply asking that this be documented on the backport.

A good example of how to document this can be found on https://review.opendev.org/c/openstack/cinder/+/821893/.

Cinder Project Releases

The Cinder project follows the OpenStack 6 month development cycle, at the end of which a new stable branch is created from master, and master becomes the development branch for the next development cycle.

Because many OpenStack consumers dont move as quickly as OpenStack development, we backport appropriate bugfixes from master into the stable branches and create new releases for consumers to use for a while. See the Stable Branches section of the OpenStack Project Team Guide for details about the timelines.

What follows is information about the Cinder project and its releases.

Where Stuff Is

The Cinder Project Deliverables

https://governance.openstack.org/tc/reference/projects/cinder.html#deliverables

The Code Repositories

- https://opendev.org/openstack/cinder
- https://opendev.org/openstack/cinderlib
- https://opendev.org/openstack/os-brick
- https://opendev.org/openstack/python-cinderclient
- https://opendev.org/openstack/python-brick-cinderclient-ext
- https://opendev.org/openstack/rbd-iscsi-client
- https://opendev.org/openstack/cinder-tempest-plugin
- https://opendev.org/openstack/cinder-specs (no releases)

Review Dashboards for Releases

- Patches for releasable stable branches: http://tiny.cc/cinder-releasable-stable
- Patches for nonreleasable stable branches: http://tiny.cc/cinder-em-branches
- Cinder project release patches: http://tiny.cc/cinder-release-patches

All Cinder Project Releases

https://releases.openstack.org/teams/cinder.html

How Stuff Works

Releases from Master

Releases from **master** for *cinder* follow the cycle-with-rc release model.

- The cycle-with-rc model describes projects that produce a single release at the end of the cycle, with one or more release candidates (RC) close to the end of the cycle and optional development milestone betas published on a per-project need.

Releases from **master** for *os-brick, cinderlib, and the clients* follow the cycle-with-intermediary release model.

- The cycle-with-intermediary model describes projects that produce multiple full releases during the development cycle, with a final release to match the end of the cycle.
- os-brick has a deliverable type of library
- python-cinderclient and python-brick-cinderclient-ext have a deliverable type of client-library
- cinderlib has a deliverable type of trailing
  - The final cinderlib release for a cycle must occur no later than 3 months after the coordinated OpenStack release of cinder.

Releases from **master** for *cinder-tempest-plugin* follow the cycle-automatic scheme.

- No stable branches are created.
- Released automatically at the end of each cycle, or on-demand.

Releases from **master** for *rbd-iscsi-client* follow the independent scheme.

- No stable branches are created.
- Released on demand whenever necessary because it has to track ceph development more than openstack development.

For more information about the release models and deliverable types: https://releases.openstack.org/reference/release_models.html

Branching

All Cinder project deliverables (except cinder-tempest-plugin and rbd-iscsi-client) follow the OpenStack stable branch policy. Briefly,

- The stable branches are intended to be a safe source of fixes for high impact bugs and security issues which have been fixed on master since a given release.
- Stable branches are cut from the last release of a given deliverable, at the end of the common 6-month development cycle.

Only members of the cinder-stable-maint gerrit group have +2 powers on patches proposed to stable branches. This is a subset of cinder-core plus the OpenStack-wide stable-maint-core team.

While anyone may propose a release, releases must be approved by the OpenStack Release Managers.

Contributing Documentation to Cinder

Starting with the Pike release, Cinders documentation has been moved from the openstack-manuals repository to the `docs` directory in the Cinder repository. This makes it even more important that Cinder add and maintain good documentation.

Note

Documentation for python-cinderclient and os-brick has undergone the same transition. The information here can be applied for those projects as well.

This page provides guidance on how to provide documentation for those who may not have previously been active writing documentation for OpenStack.

Documentation Content

To keep the documentation consistent across projects, and to maintain quality, please follow the OpenStack Writing style guide.

Using RST

OpenStack documentation uses reStructuredText to write documentation. The files end with a `.rst` extension. The `.rst` files are then processed by Sphinx to build HTML based on the RST files.

> **Note**
>
> Files that are to be included using the `.. include::` directive in an RST file should use the `.inc` extension. If you instead use the `.rst` this will result in the RST file being processed twice during the build and cause Sphinx to generate a warning during the build.

reStructuredText is a powerful language for generating web pages. The documentation team has put together an RST conventions page with information and links related to RST.

Building Cinders Documentation

To build documentation the following command should be used:

```
tox -e docs,pep8
```

When building documentation it is important to also run pep8 as it is easy to introduce pep8 failures when adding documentation. (The tox pep8 job also runs doc8, but currently we do not run doc8 as part of the tox docs job.)

> **Note**
>
> The tox documentation jobs (docs, releasenotes, api-ref) are set up to treat Sphinx warnings as errors. This is because many Sphinx warnings result in improperly formatted pages being generated, so we prefer to fix those right now, instead of waiting for someone to report a docs bug.

During the documentation build a number of things happen:

- All of the RST files under `doc/source` are processed and built.
  - The openstackdocs theme is applied to all of the files so that they will look consistent with all the other OpenStack documentation.
  - The resulting HTML is put into `doc/build/html`.
- Sample files like cinder.conf.sample are generated and put into `doc/source/_static`.
- All of Cinders `.py` files are processed and the docstrings are used to generate the files under `doc/source/contributor/api`

After the build completes the results may be accessed via a web browser in the `doc/build/html` directory structure.

Review and Release Process

Documentation changes go through the same review process as all other changes.

Note

Reviewers can see the resulting web page output by clicking on `openstack-tox-docs` in the Zuul check table on the review, and then look for Artifacts > Docs preview site.

This is also true for the `build-openstack-api-ref` and `build-openstack-releasenotes` check jobs.

Once a patch is approved it is immediately released to the docs.openstack.org website and can be seen under Cinders Documentation Page at https://docs.openstack.org/cinder/latest. When a new release is cut a snapshot of that documentation will be kept at `https://docs.openstack.org/cinder/<release>`. Changes from master can be backported to previous branches if necessary.

Doc Directory Structure

The main location for Cinders documentation is the `doc/source` directory. The top level index file that is seen at https://docs.openstack/org/cinder/latest resides here as well as the `conf.py` file which is used to set a number of parameters for the build of OpenStacks documentation.

Each of the directories under source are for specific kinds of documentation as is documented in the `README` in each directory:

Cinder Administration Documentation (source/admin)

Introduction:

This directory is intended to hold any documentation that relates to how to run or operate Cinder. Previously, this content was in the admin-guide section of openstack-manuals.

The full spec for organization of documentation may be seen in the *OS Manuals Migration Spec <https://specs.openstack.org/openstack/docs-specs/specs/pike/os-manuals-migration.html>*.

Cinder CLI Documentation (source/cli)

Introduction:

This directory is intended to hold any documentation that relates to Cinders Command Line Interface. Note that this directory is intended for basic descriptions of the commands supported, similar to what you would find with a man page. Tutorials or step-by-step guides should go into doc/source/admin or doc/source/user depending on the target audience.

The full spec for organization of documentation may be seen in the *OS Manuals Migration Spec <https://specs.openstack.org/openstack/docs-specs/specs/pike/os-manuals-migration.html>*.

Cinder Configuration Documentation (source/configuration)

Introduction:

This directory is intended to hold any documentation that relates to how to configure Cinder. It is intended that some of this content be automatically generated in the future. At the moment, however, it is not. If you would like to work on this, please use Launchpad Bug #1847600 for tracking purposes. Changes to configuration options for Cinder or its drivers needs to be put under this directory.

The full spec for organization of documentation may be seen in the OS Manuals Migration Spec.

Cinder Contributor Documentation (source/contributor)

Introduction:

This directory is intended to hold any documentation that relates to how to contribute to Cinder or how the project is managed. Some of this content was previous under developer in openstack-manuals. The content of the documentation, however, goes beyond just developers to anyone contributing to the project, thus the change in naming.

The full spec for organization of documentation may be seen in the *OS Manuals Migration Spec <https://specs.openstack.org/openstack/docs-specs/specs/pike/os-manuals-migration.html>*.

Cinder Installation Documentation (source/install)

Introduction:

This directory is intended to hold any installation documentation for Cinder. Documentation that explains how to bring Cinder up to the point that it is ready to use in an OpenStack or standalone environment should be put in this directory.

The full spec for organization of documentation may be seen in the *OS Manuals Migration Spec <https://specs.openstack.org/openstack/docs-specs/specs/pike/os-manuals-migration.html>*.

Cinder Reference Documentation (source/reference)

Introduction:

This directory is intended to hold any reference documentation for Cinder that doesnt fit into install, contributor, configuration, cli, admin, or user categories.

The full spec for organization of documentation may be seen in the *OS Manuals Migration Spec <https://specs.openstack.org/openstack/docs-specs/specs/pike/os-manuals-migration.html>*.

Cinder User Documentation (source/user)

Introduction:

This directory is intended to hold any documentation that helps Cinder end-users. This can include concept guides, tutorials, step-by-step guides for using the CLI, etc. Note that documentation this is focused on administrative actions should go into doc/source/admin.

The full spec for organization of documentation may be seen in the *OS Manuals Migration Spec <https://specs.openstack.org/openstack/docs-specs/specs/pike/os-manuals-migration.html>*.

Finding something to contribute

If you are reading the documentation and notice something incorrect or undocumented, you can directly submit a patch following the advice set out below.

There are also documentation bugs that other people have noticed that you could address:

- https://bugs.launchpad.net/cinder/+bugs?field.tag=doc
- https://bugs.launchpad.net/python-cinderclient/+bugs?field.tag=doc
- https://bugs.launchpad.net/os-brick/+bugs?field.tag=doc

- https://bugs.launchpad.net/cinderlib/+bugs?field.tag=doc

Note

If you dont see a bug listed, you can also try the tag docs or documentation. We tend to use doc as the appropriate tag, but occasionally a bug gets tagged with a variant.

Writing Release Notes

Please follow the format, it will make everyones life easier. Theres even a special section on writing release notes for Cinder drivers.

Release notes

The release notes for a patch should be included in the patch.

If the following applies to the patch, a release note is required:

- Upgrades
  - The deployer needs to take an action when upgrading
  - A new config option is added that the deployer should consider changing from the default
  - A configuration option is deprecated or removed
- Features
  - A new feature or driver is implemented
  - Feature is deprecated or removed
  - Current behavior is changed
- Bugs
  - A security bug is fixed
  - A long-standing or important bug is fixed
- APIs
  - REST API changes

Reviewing release note content

Release notes are user facing. We expect operators to read them (and other people interested in seeing whats in a new release may read them, too). This makes a release note different from a commit message, which is aimed at other developers.

Keep this in mind as you review a release note. Also, since its user facing, something you would think of as a nit in a code comment (for example, bad punctuation or a misspelled word) is not really a nit in a release noteits something that needs to be corrected. This also applies to the format of the release note, which should follow the standards set out later in this document.

In summary, dont feel bad about giving a -1 for a nit in a release note. We dont want to have to go back and fix typos later, especially for a bugfix thats likely to be backported, which would require squashing the typo fix into the backport patch (which is something thats easy to forget). Thus we really want to get release notes right the first time.

Fixing a release note

Of course, even with careful writing and reviewing, a mistake can slip through that isnt noticed until after a release. If that happens, the patch to correct a release note must be proposed *directly to the stable branch in which the release note was introduced*. (Yes, this is completely different from how we handle bugs.)

This is because of how reno scans release notes and determines what release they go with. See Updating Stable Branch Release Notes in the *reno User Guide* for more information.

Bugs

For bug fixes, release notes must include the bug number in Launchpad with a link to it as a RST link like in the following example:

```
---
fixes:
  - |
    `Bug #1889758 <https://bugs.launchpad.net/cinder/+bug/1889758>`_: Fixed
    revert to snapshot not working for non admin users when using the
    snapshot's name.
```

Note the use of the past tense (Fixed) instead of the present tense (Fix). This is because although you are fixing the bug right now in the present, operators will be reading the release notes in the future (at the time of the release), at which time your bug fix will be a thing of the past.

Additionally, keep in mind that when your release note is published, it is mixed in with all the other release notes and wont obviously be connected to your patch. Thus, in order for it to make sense, you may need to repeat information that you already have in your commit message. Thats OK.

Drivers

For release notes related to a specific driver -be it volume, backup, or zone manager- the release note line must start with `<driver-name> driver:`. For example:

```
---
features:
  - |
    RBD driver: Added support for volume manage and unmanage operations.
```

When fixing a driver bug we must not only have the driver name prefix but also the bug number and link:

```
---
fixes:
  - |
    Brocade driver `bug #1866860
    <https://bugs.launchpad.net/cinder/+bug/1889758>`_: Fixed
    ``AttributeError`` when using ``REST_HTTP`` or ``REST_HTTPS`` as the
    ``fc_southbound_protocol`` option and an exception is raised by the
    client.
```

There are times when a bug affects multiple drivers. In such a cases we must list each of the driver as an independent item following above rules:

```
---
fixes:
  - |
    Unity driver `bug #1881108
    <https://bugs.launchpad.net/cinder/+bug/1881108>`_: Fixed leaving
    leftover devices on the host when validation of the attached volume
    fails on some cloning cases and create volume from snapshot.
  - |
    Kaminario driver `bug #1881108
    <https://bugs.launchpad.net/cinder/+bug/1881108>`_:  Fixed leaving
    leftover devices on the host when validation of the attached volume
    fails on some cloning cases and create volume from snapshot.
```

Creating the note

Cinder uses reno to generate release notes. Please read the docs for details. In summary, use

```
$ tox -e venv -- reno new <bug-,bp-,whatever>
```

Then edit the sample file that was created and push it with your change.

To see the results:

```
$ git commit  # Commit the change because reno scans git log.

$ tox -e releasenotes
```

Then look at the generated release notes files in releasenotes/build/html in your favorite browser.

Programming HowTos and Tutorials

Setting Up a Development Environment

This page describes how to setup a working Python development environment that can be used in developing cinder on Ubuntu, Fedora or macOS. These instructions assume youre already familiar with git. Refer to GettingTheCode for additional information.

Following these instructions will allow you to run the cinder unit tests. Running cinder is currently only supported on Linux. Some jobs can be run on macOS, but unfortunately due to some differences in system packages there are known issues with running unit tests.

Virtual environments

Cinder development uses virtualenv to track and manage Python dependencies while in development and testing. This allows you to install all of the Python package dependencies in a virtual environment or virtualenv (a special subdirectory of your cinder directory), instead of installing the packages at the system level.

Note

Virtualenv is useful for running the unit tests, but is not typically used for full integration testing or production usage.

Linux Systems

Note

If you have Ansible and git installed on your system, you may be able to get a working development environment quickly set up by running the following:

```
sudo ansible-pull -U https://github.com/stmcginnis/cinder-dev-setup
```

If that does not work for your system, continue on with the manual steps below.

Install the prerequisite packages.

On Ubuntu20.04-64:

```
sudo apt-get install libssl-dev python3-pip libmysqlclient-dev libpq-dev␣
↪libffi-dev
```

To get a full python3 development environment, the two python3 packages need to be added to the list above:

```
python3-dev python3-pip
```

On Red Hat-based distributions e.g., Fedora/RHEL/CentOS/Scientific Linux (tested on CentOS 6.5 and CentOS 7.3):

```
sudo yum install python-virtualenv openssl-devel python-pip git gcc libffi-
↪devel libxslt-devel mysql-devel postgresql-devel
```

On openSUSE-based distributions (SLES 12, openSUSE 13.1, Factory or Tumbleweed):

```
sudo zypper install gcc git libmysqlclient-devel libopenssl-devel postgresql-
↪devel python-devel python-pip
```

macOS Systems

Install virtualenv:

```
sudo pip install virtualenv
```

Check the version of OpenSSL you have installed:

```
openssl version
```

If you have installed OpenSSL 1.0.0a, which can happen when installing a MacPorts package for OpenSSL, you will see an error when running `cinder.tests.auth_unittest.AuthTestCase.test_209_can_generate_x509`.

The stock version of OpenSSL that ships with Mac OS X 10.6 (OpenSSL 0.9.8l) or later should work fine with cinder.

Getting the code

Grab the code:

```
git clone https://opendev.org/openstack/cinder.git
cd cinder
```

Running unit tests

The preferred way to run the unit tests is using `tox`. It executes tests in isolated environment, by creating separate virtualenv and installing dependencies from the `requirements.txt` and `test-requirements.txt` files, so the only package you install is `tox` itself:

```
sudo pip install tox
```

Run the unit tests by doing:

```
tox -e py3
```

See *Testing* for more details.

Manually installing and using the virtualenv

You can also manually install the virtual environment:

```
tox -e py3 --notest
```

This will install all of the Python packages listed in the `requirements.txt` file into your virtualenv.

To activate the Cinder virtualenv you can run:

```
$ source .tox/py3/bin/activate
```

To exit your virtualenv, just type:

```
$ deactivate
```

Or, if you prefer, you can run commands in the virtualenv on a case by case basis by running:

```
$ tox -e venv -- <your command>
```

Contributing Your Work

Once your work is complete you may wish to contribute it to the project. Cinder uses the Gerrit code review system. For information on how to submit your branch to Gerrit, see GerritWorkflow.

Testing

Cinder contains a few different test suites in the cinder/tests/ directory. The different test suites are Unit Tests, Functional Tests, and Tempest Tests.

Test Types

Unit Tests

Unit tests are tests for individual methods, with at most a small handful of modules involved. Mock should be used to remove any external dependencies.

All significant code changes should have unit test coverage validating the code happy path and any failure paths.

Theres a tox environment defined that will run code coverage tests for you:

```
tox -e cover
```

It will create an HTML code coverage report that you can use a web browser to read locally from the location `./cover/index.html` (relative to the location of your `tox.ini` file). If you are reviewing someone elses patch in Gerrit, we have `cinder-code-coverage` job that generates a coverage report that you can read. From the review page, follow: Zuul Summary tab -> cinder-code-coverage link -> Logs tab. The raw link next to cover will take you to the index page of the report.

Any proposed code change will be automatically rejected by the OpenDev Zuul project gating system[1] if the change causes unit test failures.

Functional Tests

Functional tests validate a code path within Cinder. These tests should validate the interaction of various modules within the project to verify the code is logically correct.

Functional tests run with a database present and may start Cinder services to accept requests. These tests should not need to access an other OpenStack non-Cinder services.

Tempest Tests

The tempest tests in the Cinder tree validate the operational correctness between Cinder and external components such as Nova, Glance, etc. These are integration tests driven via public APIs to verify actual end user usage scenarios.

Running the tests

There are a number of ways to run tests currently, and theres a combination of frameworks used depending on what commands you use. The preferred method is to use tox, which calls ostestr via the tox.ini file.

Unit Tests

To run all unit tests simply run:

```
tox
```

This will create a virtual environment, load all the packages from test-requirements.txt and run all unit tests as well as run flake8 and hacking checks against the code.

You may run individual test targets, for example only py37 tests, by running:

[1] See *Continuous Integration with Zuul*.

```
tox -e py37
```

Note that you can inspect the tox.ini file to get more details on the available options and what the test run does by default.

Functional Tests

To run all functional tests, run:

```
tox -e functional
```

Tempest Tests

Tempest tests in the Cinder tree are plugged in to the normal tempest test execution. To ensure the Cinder tests are picked up when running tempest, run:

```
cd /opt/stack/tempest
tox -e venv-tempest -- pip install (path to the cinder-tempest-plugin␣
↪directory)
tox -e all
```

More information about tempest can be found in the Tempest Documentation.

Database Setup

Some unit and functional tests will use a local database. You can use `tools/test-setup.sh` to set up your local system the same way as its setup in the CI environment.

Running a subset of tests using tox

One common activity is to just run a single test, you can do this with tox simply by specifying to just run py37 tests against a single test:

```
tox -epy37 -- cinder.tests.unit.volume.test_availability_zone.
↪AvailabilityZoneTestCase.test_list_availability_zones_cached
```

Or all tests in the test_volume.py file:

```
tox -epy37 -- cinder.tests.unit.volume.test_volume
```

You may also use regular expressions to run any matching tests:

```
tox -epy37 -- test_volume
```

For more information on these options and details about stestr, please see the stestr documentation.

Gotchas

Running Tests from Shared Folders

If you are running the unit tests from a shared folder, you may see tests start to fail or stop completely as a result of Python lockfile issues. You can get around this by manually setting or updating the following line in `cinder/tests/conf_fixture.py`:

```
CONF['lock_path'].SetDefault('/tmp')
```

Note that you may use any location (not just `/tmp`!) as long as it is not a shared folder.

Assertion types in unit tests

In general, it is best to use the most specific assertion possible in a unit test, to have the strongest validation of code behavior.

For example:

```
self.assertEqual("in-use", volume.status)
```

is preferred over

```
self.assertIsNotNone(volume.status)
```

or

Test methods that implement comparison checks are also generally preferred over writing code into assertEqual() or assertTrue().

```
self.assertGreater(2, volume.size)
```

is preferred over

```
self.assertTrue(2 > volume.size)
```

However, assertFalse() behavior is not obvious in this regard. Since `None` evaluates to `False` in Python, the following check will pass when x is `False` or `None`.

```
self.assertFalse(x)
```

Therefore, it is preferable to use:

```
self.assertEqual(x, False)
```

Debugging

Debugging unit tests

It is possible to attach a debugger to unit tests.

First, modify the test you want to debug by adding the following to the test code itself:

```
import pdb
pdb.set_trace()
```

Then run the unit test with pdb enabled:

```
source .tox/py36/bin/activate

stestr run -n cinder.tests.unit.test_volume_utils

```

(continues on next page)

(continued from previous page)

```
# Or to get a list of tests to run

stestr list test_volume_utils > tests_to_run.txt
stestr run --load-list tests_to_run.txt
```

API Microversions

Background

Cinder uses a framework we called API Microversions for allowing changes to the API while preserving backward compatibility. The basic idea is that a user has to explicitly ask for their request to be treated with a particular version of the API. So breaking changes can be added to the API without breaking users who dont specifically ask for it. This is done with an HTTP header `OpenStack-API-Version` which is a monotonically increasing semantic version number starting from `3.0`.

Each OpenStack service that uses microversions will share this header, so the Volume service will need to prefix the semantic version number with the word `volume`:

```
OpenStack-API-Version: volume 3.0
```

If a user makes a request without specifying a version, they will get the `_MIN_API_VERSION` as defined in `cinder/api/openstack/api_version_request.py`. This value is currently `3.0` and is expected to remain so for quite a long time.

The Nova project was the first to implement microversions. For full details please read Novas Kilo spec for microversions

When do I need a new Microversion?

A microversion is needed when the contract to the user is changed. The user contract covers many kinds of information such as:

- the Request
  - the list of resource URLs which exist on the server

    Example: adding a new shares/{ID}/foo which didnt exist in a previous version of the code
  - the list of query parameters that are valid on URLs

    Example: adding a new parameter `is_yellow` servers/{ID}?is_yellow=True
  - the list of query parameter values for non free form fields

    Example: parameter filter_by takes a small set of constants/enums A, B, C. Adding support for new enum D.
  - new headers accepted on a request
- the Response
  - the list of attributes and data structures returned

    Example: adding a new attribute locked: True/False to the output of shares/{ID}
  - the allowed values of non free form fields

    Example: adding a new allowed `status` to shares/{ID}

- the list of status codes allowed for a particular request

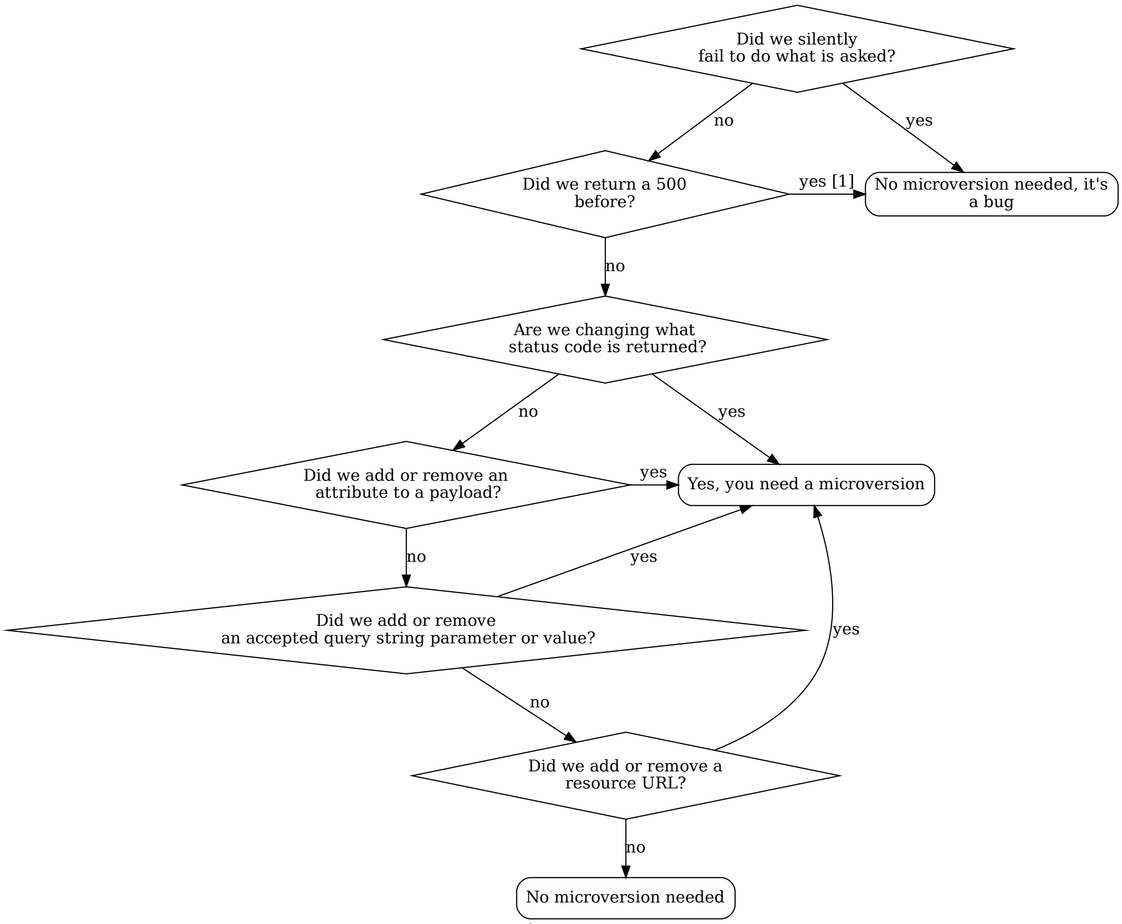
  Example: an API previously could return 200, 400, 403, 404 and the change would make the API now also be allowed to return 409.

- changing a status code on a particular response

  Example: changing the return code of an API from 501 to 400.

- new headers returned on a response

The following flow chart attempts to walk through the process of do we need a microversion.



Do I need a microversion?

If a patch that will require a microversion increment is proposed having similar intention and code with a previously merged patch given the previous merged patch hasnt been released, then the previously merged patch could be modified to include the new patch code under the same microversion.

Footnotes

[1] - When fixing 500 errors that previously caused stack traces, try to map the new error into the existing set of errors that API call could previously return (400 if nothing else is appropriate). Changing the set of allowed status codes from a request is changing the contract, and should be part of a microversion.

The reason why we are so strict on contract is that wed like application writers to be able to know, for sure, what the contract is at every microversion in Cinder. If they do not, they will need to write conditional code in their application to handle ambiguities.

When in doubt, consider application authors. If it would work with no client side changes on both Cinder versions, you probably dont need a microversion. If, on the other hand, there is any ambiguity, a microversion is probably needed.

In Code

In `cinder/api/openstack/wsgi.py` we define an `@api_version` decorator which is intended to be used on top-level Controller methods. It is not appropriate for lower-level methods. Some examples:

Adding a new API method

In the controller class:

```
@wsgi.Controller.api_version("3.4")
def my_api_method(self, req, id):
    ....
```

This method would only be available if the caller had specified an `OpenStack-API-Version` of >= `3.4`. If they had specified a lower version (or not specified it and received the default of `3.1`) the server would respond with `HTTP/404`.

Removing an API method

In the controller class:

```
@wsgi.Controller.api_version("3.1", "3.4")
def my_api_method(self, req, id):
    ....
```

This method would only be available if the caller had specified an `OpenStack-API-Version` of <= `3.4`, and >= `3.1`. If `3.5` or later is specified or if `3.0` or earlier (/v2 or /v1 endpoint), the server will respond with `HTTP/404`

Changing a methods behaviour

In the controller class:

```
@wsgi.Controller.api_version("3.1", "3.3")
def my_api_method(self, req, id):
    .... method_1 ...

@my_api_method.api_version("3.4")
def my_api_method(self, req, id):
    .... method_2 ...
```

If a caller specified `3.1`, `3.2` or `3.3` (or received the default of `3.1`) they would see the result from `method_1`, `3.4` or later `method_2`.

We could use `wsgi.Controller.api_version` decorator on the second `my_api_method` as well, but then we would have to add `# noqa` to that line to avoid failing flake8s `F811` rule. So the recommended approach is to use the `api_version` decorator from the first method that is defined, as illustrated by the example above, and then use `my_api_method` decorator for subsequent api versions of the same method.

The two methods may be different in any kind of semantics (schema validation, return values, response codes, etc.).

A method with only small changes between versions

A method may have only small changes between microversions, in which case you can decorate a private method:

```
@wsgi.Controller.api_version("3.1", "3.4")
def _version_specific_func(self, req, arg1):
    pass

@_version_specific_func.api_version(min_ver="3.5")
def _version_specific_func(self, req, arg1):
    pass

def show(self, req, id):
    .... common stuff ....
    self._version_specific_func(req, "foo")
    .... common stuff ....
```

When not using decorators

When you dont want to use the `@api_version` decorator on a method or you want to change behaviour within a method (say it leads to simpler or simply a lot less code) you can directly test for the requested version with a method as long as you have access to the api request object (commonly called `req`). Every API method has an api_version_request object attached to the req object and that can be used to modify behaviour based on its value:

```
def index(self, req):
    <common code>

    req_version = req.api_version_request
    if req_version.matches("3.1", "3.5"):
        ....stuff....
    elif req_version.matches("3.6", "3.10"):
        ....other stuff....
    elif req_version > api_version_request.APIVersionRequest("3.10"):
        ....more stuff.....

    <common code>
```

The first argument to the matches method is the minimum acceptable version and the second is maximum acceptable version. A specified version can be null:

```
null_version = APIVersionRequest()
```

If the minimum version specified is null then there is no restriction on the minimum version, and likewise if the maximum version is null there is no restriction the maximum version. Alternatively an one sided comparison can be used as in the example above.

Other necessary changes

If you are adding a patch which adds a new microversion, it is necessary to add changes to other places which describe your change:

- Update `REST_API_VERSION_HISTORY` in `cinder/api/openstack/api_version_request.py`
- Update `_MAX_API_VERSION` in `cinder/api/openstack/api_version_request.py`
- Add a verbose description to `cinder/api/openstack/rest_api_version_history.rst`. There should be enough information that it could be used by the docs team for release notes.
- Constants should be used in the code to minimize errors on microversion merge conflicts. Define a constant for the new microversion in the `cinder/api/microversions.py` file and use that in the rest of the code.
- Update the expected versions in affected tests.
- API changes should almost always include a release note announcing the availability of the new API functionality. The description of the API change should indicate which microversion is required for the change, and it should refer to the numerical value of the microversion and not its constant name.
- Update the `version` parameter in api-ref responses here `cinder/api-ref/ source/v3/samples/versions/version-show-response.json` and here `cinder/api-ref/source/v3/samples/versions/versions-response.json` to the latest microversion to avoid functional test failure.
- If the API microversion has changed an endpoint accepted parameters or the values it returns, we need to create the appropriate API samples within the `api-ref/source/v3/samples` tree creating a new `vX.Y` directory with our request and/or response json.
- Update the functional API tests in the `cinder/tests/functional/api_sample_tests` tree to make requests and validate responses with the new microversion. There are multiple convenience methods provided for testing, such as `use_versions` class decorator that allows us to run the same tests with different microversions (each will use their respective json and templates), the `override_mv` method decorator to change the microversion in a single test, and the `common_api_sample` context manager to use the base sample instead of a microversion specific one.
- Update the documentation adding any new parameter to `api-ref/source/v3/parameters.yaml` (remember to add the `min_version`) and then making appropriate changes to the `.inc` file in `api-ref/source/v3/` to reflect new possible return codes, new accepted parameters and their `Request Example (vX.Y)` title and include file, and returned values and their `Response Example (vX.Y)` title and include file.

The Cinder projects policy is that the sample requests and responses should always reflect the *most recent* microversion.

Allocating a microversion

If you are adding a patch which adds a new microversion, it is necessary to allocate the next microversion number. Except under extremely unusual circumstances and this would have been mentioned in the blueprint for the change, the minor number of `_MAX_API_VERSION` will be incremented. This will also be the new microversion number for the API change.

It is possible that multiple microversion patches would be proposed in parallel and the microversions would conflict between patches. This will cause a merge conflict. We dont reserve a microversion for each patch in advance as we dont know the final merge order. Developers may need over time to rebase their patch calculating a new version number as above based on the updated value of `_MAX_API_VERSION`.

Testing Microversioned API Methods

Unit tests for microversions should be put in cinder/tests/unit/api/v3/ . Since all existing functionality is tested in cinder/tests/unit/api/v2, these unit tests are not replicated in /v3, and only new functionality needs to be place in the /v3/directory.

Testing a microversioned API method is very similar to a normal controller method test, you just need to add the `OpenStack-API-Version` header, for example:

```
req = fakes.HTTPRequest.blank('/testable/url/endpoint')
req.headers['OpenStack-API-Version'] = 'volume 3.6'
req.api_version_request = api_version.APIVersionRequest('3.6')

controller = controller.TestableController()

res = controller.index(req)
... assertions about the response ...
```

REST API Version History

Details for each existing microversion change can be found in the *REST API Version History* documentation.

REST API Version History

This documents the changes made to the REST API with every microversion change. The description for each version should be a verbose one which has enough information to be suitable for use in user documentation.

3.0 (Maximum in Mitaka)

The 3.0 Cinder API includes all v2 core APIs existing prior to the introduction of microversions. The /v3 URL is used to call 3.0 APIs. This is the initial version of the Cinder API which supports microversions.

A user can specify a header in the API request:

```
OpenStack-API-Version: volume <version>
```

where <`version`> is any valid api version for this API.

If no version is specified then the API will behave as if version 3.0 was requested.

The only API change in version 3.0 is versions, i.e. GET http://localhost:8786/, which now returns information about 3.0 and later versions and their respective /v3 endpoints.

All other 3.0 APIs are functionally identical to version 2.0.

3.1

Added the parameters `protected` and `visibility` to _volume_upload_image requests.

3.2

Change in return value of GET API request for fetching cinder volume list on the basis of bootable status of volume as filter.

Before V3.2, GET API request to fetch volume list returns non-bootable volumes if bootable filter value is any of the false or False. For any other value provided to this filter, it always returns bootable volume list.

But in V3.2, this behavior is updated. In V3.2, bootable volume list will be returned for any of the T/True/1/true bootable filter values only. Non-bootable volume list will be returned for any of F/False/0/false bootable filter values. But for any other values passed for bootable filter, it will return Invalid input received: bootable={filter value} error.

3.3

Added /messages API.

3.4

Added the filter parameters `glance_metadata` to list/detail volumes requests.

3.5

Added pagination support to /messages API

3.6

Allowed to set empty description and empty name for consistency group in consisgroup-update operation.

3.7

Added `cluster_name` field to service list/detail.

Added /clusters endpoint to list/show/update clusters.

Show endpoint requires the cluster name and optionally the binary as a URL parameter (default is cinder-volume). Returns:

```
{
    "cluster": {
        "created_at": "",
        "disabled_reason": null,
        "last_heartbeat": "",
        "name": "cluster_name",
        "num_down_hosts": 4,
        "num_hosts": 2,
        "state": "up",
        "status": "enabled",
```

(continues on next page)

(continued from previous page)

```
        "updated_at": ""
    }
}
```

Update endpoint allows enabling and disabling a cluster in a similar way to services update endpoint, but in the body we must specify the name and optionally the binary (cinder-volume is the default) and the disabled reason. Returns:

```
{
    "cluster": {
        "name": "cluster_name",
        "state": "up",
        "status": "enabled",
        "disabled_reason": null
    }
}
```

Index and detail accept filtering by *name*, *binary*, *disabled*, *num_hosts* , *num_down_hosts*, and up/down status (*is_up*) as URL parameters.

Index endpoint returns:

```
{
    "clusters": [
        {
            "name": "cluster_name",
            "state": "up",
            "status": "enabled"
        }
    ]
 }
```

Detail endpoint returns:

```
{
    "clusters": [
        {
            "created_at": "",
            "disabled_reason": null,
            "last_heartbeat": "",
            "name": "cluster_name",
            "num_down_hosts": 4,
            "num_hosts": 2,
            "state": "up",
            "status": "enabled",
            "updated_at": ""
        }
    ]
}
```

3.8

Adds the following resources that were previously in extensions:

- os-volume-manage => /v3/<project_id>/manageable_volumes
- os-snapshot-manage => /v3/<project_id>/manageable_snapshots

3.9

Added backup update interface to change name and description. Returns:

```
{
    "backup": {
        "id": "backup_id",
        "name": "backup_name",
        "links": "backup_link"
    }
}
```

3.10

Added the filter parameters `group_id` to list/detail volumes requests.

3.11

Added group types and group specs APIs.

3.12

Added volumes/summary API.

3.13

Added create/delete/update/list/show APIs for generic volume groups.

3.14

Added group snapshots and create group from src APIs.

3.15 (Maximum in Newton)

Added injecting the responses *Etag* header to avoid the lost update problem with volume metadata.

3.16

os-migrate_volume now accepts `cluster` parameter when we want to migrate a volume to a cluster. If we pass the `host` parameter for a volume that is in a cluster, the request will be sent to the cluster as if we had requested that specific cluster. Only `host` or `cluster` can be provided.

Creating a managed volume also supports the cluster parameter.

3.17

os-snapshot-manage and os-volume-manage now support `cluster` parameter on listings (summary and detailed). Both location parameters, `cluster` and `host` are exclusive and only one should be provided.

3.18

Added backup project attribute.

3.19

Added reset status actions reset_status to group snapshot.

3.20

Added reset status actions reset_status to generic volume group.

3.21

Show provider_id in detailed view of a volume for admin.

3.22

Added support to filter snapshot list based on metadata of snapshot.

3.23

Allow passing force parameter to volume delete.

3.24

New API endpoint /workers/cleanup allows triggering cleanup for cinder-volume services. Meant for cleaning ongoing operations from failed nodes.

The cleanup will be performed by other services belonging to the same cluster, so at least one of them must be up to be able to do the cleanup.

Cleanup cannot be triggered during a cloud upgrade.

If no arguments are provided cleanup will try to issue a clean message for all nodes that are down, but we can restrict which nodes we want to be cleaned using parameters `service_id`, `cluster_name`, `host`, `binary`, and `disabled`.

Cleaning specific resources is also possible using `resource_type` and `resource_id` parameters.

We can even force cleanup on nodes that are up with `is_up`, but thats not recommended and should only used if you know what you are doing. For example if you know a specific cinder-volume is down even though its still not being reported as down when listing the services and you know the cluster has at least another service to do the cleanup.

API will return a dictionary with 2 lists, one with services that have been issued a cleanup request (`cleaning` key) and the other with services that cannot be cleaned right now because there is no alternative service to do the cleanup in that cluster (`unavailable` key).

Data returned for each service element in these two lists consist of the `id`, `host`, `binary`, and `cluster_name`. These are not the services that will be performing the cleanup, but the services that will be cleaned up or couldnt be cleaned up.

3.25

Add `volumes` field to group list/detail and group show.

3.26

- New `failover` action equivalent to `failover_host`, but accepting `cluster` parameter as well as the `host` cluster that `failover_host` accepts.
- `freeze` and `thaw` actions accept `cluster` parameter.
- Cluster listing accepts `replication_status`, `frozen` and `active_backend_id` as filters, and returns additional fields for each cluster: `replication_status`, `frozen`, `active_backend_id`.

3.27 (Maximum in Ocata)

Added new attachment APIs. See the API reference for details.

3.28

Add filters support to get_pools

3.29

Add filter, sorter and pagination support in group snapshot.

3.30

Support sort snapshots with name.

3.31

Add support for configure resource query filters.

3.32

Added `set-log` and `get-log` service actions.

3.33

Add `resource_filters` API to retrieve configured resource filters.

3.34

Add like filter support in `volume`, `backup`, `snapshot`, `message`, `attachment`, `group` and `group-snapshot` list APIs.

3.35

Add `volume-type` filter to Get-Pools API.

3.36

Add metadata to volumes/summary response body.

3.37

Support sort backup by name.

3.38

Added enable_replication/disable_replication/failover_replication/ list_replication_targets for replication groups (Tiramisu).

3.39

Add `project_id` admin filters support to limits.

3.40

Add volume revert to its latest snapshot support.

3.41

Add `user_id` field to snapshot list/detail and snapshot show.

3.42

Add ability to extend in-use volume. User should be aware of the whole environment before using this feature because its dependent on several external factors below:

1. nova-compute version - needs to be the latest for Pike.
2. only the libvirt compute driver supports this currently.
3. only iscsi and fibre channel volume types are supported on the nova side currently.

Administrator can disable this ability by updating the `volume:extend_attached_volume` policy rule. Extend of a reserved Volume is NOT allowed.

3.43 (Maximum in Pike)

Support backup CRUD with metadata.

3.44

Support attachment completion. See the API reference for details.

3.45

Add `count` field to volume, backup and snapshot list and detail APIs.

3.46

Support create volume by Nova specific image (0 size image).

3.47

Support create volume from backup.

3.48

Add `shared_targets` and `service_uuid` fields to volume.

3.49

Support report backend storage state in service list.

3.50 (Maximum in Queens)

Services supporting this microversion are capable of volume multiattach. This version does not need to be requested when creating the volume, but can be used as a way to query if the capability exists in the Cinder service.

3.51

Add support for cross AZ backups.

3.52

`RESKEY:availability_zones` is a reserved spec key for AZ volume type, and filter volume type by `extra_specs` is supported now.

3.53

Schema validation support has been added using jsonschema for V2/V3 volume APIs.

- **Create volume API**
  Before 3.53, create volume API used to accept any invalid parameters in the request body like the ones below were passed by python-cinderclient.
  1. user_id
  2. project_id
  3. status
  4. attach_status

  But in 3.53, this behavior is updated. If user passes any invalid parameters to the API which are not documented in api-ref, then it will raise badRequest error.

- **Update volume API**

  Before 3.53, even if user doesnt pass any valid parameters in the request body, the volume was updated. But in 3.53, user will need to pass at least one valid parameter in the request body otherwise it will return 400 error.

3.54

Add `mode` argument to attachment-create.

3.55 (Maximum in Rocky)

Support ability to transfer snapshots along with their parent volume.

3.56

Add `user_id` attribute to response body of list backup with detail and show backup detail APIs.

3.57

Expanded volume transfer record details by adding `source_project_id`, `destination_project_id` and `accepted` fields to `transfer` table and related api (create/show/list detail transfer APIs) responses.

3.58

Add `project_id` attribute to response body of list groups with detail, list group snapshots with detail, show group detail and show group snapshot detail APIs.

3.59 (Maximum in Stein and Train)

Support volume transfer pagination.

3.60 (Maximum in Ussuri)

Users may apply time comparison filters to the volume summary list and volume detail list requests by using the `created_at` or `updated_at` fields. Time must be expressed in ISO 8601 format.

3.61

Add `cluster_name` attribute to response body of volume details for admin in Active/Active HA mode.

3.62 (Maximum in Victoria)

Add support for set, get, and unset a default volume type for a specific project. Setting this default overrides the configured default_volume_type value.

3.63

Includes volume type ID in the volume-show and volume-detail-list JSON responses. Before this microversion, Cinder returns only the volume type name in the volume details.

3.64 (Maximum in Wallaby)

Include the `encryption_key_id` in volume and backup details when the associated volume is encrypted.

3.65

Include a `consumes_quota` field in volume and snapshot details to indicate whether the resource is consuming quota or not. Also, accept a `consumes_quota` filter, which takes a boolean value, in the volume and snapshot list requests. (The default listing behavior is not to use this filter.)

3.66 (Maximum in Xena)

Volume snapshots of in-use volumes can be created without the force flag. Although the force flag is now considered invalid when passed in a volume snapshot request, for backward compatibility, the force flag with a value evaluating to True is silently ignored.

3.67

API URLs no longer need a project_id argument in them. For example, the API route: `https://$(controller)s/volume/v3/$(project_id)s/volumes` is equivalent to `https://$(controller)s/volume/v3/volumes`. When interacting with the cinder service as system or domain scoped users, a project_id should not be specified in the API path.

3.68 (Maximum in Yoga)

Support ability to re-image a volume with a specific image. Specify the `os-reimage` action in the request body.

3.69

Volume field `shared_targets` is a tristate boolean value now, with the following meanings:

- `true`: Do os-brick locking when host iSCSI initiator doesnt support manual scans.
- `false`: Never do locking.
- `null`: Forced locking regardless of the iSCSI initiator.

3.70 (Maximum in Zed, 2023.1 and 2023.2)

Add the ability to transfer encrypted volumes and their snapshots. The feature removes a prior restriction on transferring encrypted volumes. Otherwise, the API request and response schema are unchanged.

3.71 (Maximum in 2024.1 and 2024.2)

Add the `os-extend_volume_completion` volume action, which Nova can use to notify Cinder of success and error when handling a `volume-extended` external server event.

API Races - Conditional Updates

Background

On Cinder API nodes we have to check that requested action can be performed by checking request arguments and involved resources, and only if everything matches required criteria we will proceed with the RPC call to any of the other nodes.

Checking the conditions must be done in a non racy way to ensure that already checked requirements dont change while we check remaining conditions. This is of utter importance, as Cinder uses resource status as a lock to prevent concurrent operations on a resource.

An simple example of this would be extending a volume, where we first check the status:

```
if volume['status'] != 'available':
```

Then update the status:

```
self.update(context, volume, {'status': 'extending'})
```

And finally make the RPC call:

```
self.volume_rpcapi.extend_volume(context, volume, new_size,
                                 reservations)
```

The problem is that this code would allow races, as other request could have already changed the volume status between us getting the value and updating the DB.

There are multiple ways to fix this, such as:

- Using a Distributed Locking Mechanism
- Using DB isolation level
- Using SQL SELECT FOR UPDATE
- USING compare and swap mechanism in SQL query

Our tests showed that the best alternative was compare and swap and we decided to call this mechanism Conditional Update as it seemed more appropriate.

Conditional Update

Conditional Update is the mechanism we use in Cinder to prevent races when updating the DB. In essence it is the SQL equivalent of an `UPDATE ... FROM ... WHERE;` clause.

It is implemented as an abstraction layer on top of SQLAlchemy ORM engine in our DB api layer and exposed for consumption in Cinders Persistent Versioned Objects through the `conditional_update` method so it can be used from any Versioned Object instance that has persistence (Volume, Snapshot, Backup).

Method signature is:

```
def conditional_update(self, values, expected_values=None, filters=(),
                       save_all=False, session=None, reflect_changes=True,
                       order=None):
```

values
: Dictionary of key-value pairs with changes that we want to make to the resource in the DB.

expected_values
: Dictionary with conditions that must be met for the update to be executed.

Condition `field.id == resource.id` is implicit and there is no need to add it to the conditions.

If no `expected_values` argument is provided update will only go through if no field in the DB has changed. Dirty fields from the Versioned Object are excluded as we dont know their original value.

filters
Additional SQLAlchemy filters can be provided for more complex conditions.

save_all
By default we will only be updating the DB with values provided in the `values` argument, but we can explicitly say that we also want to save objects current dirty fields.

session
A SQLAlchemy session can be provided, although it is unlikely to be needed.

reflect_changes
On a successful update we will also update Versioned Object instance to reflect these changes, but we can prevent this instance update passing False on this argument.

order
Specific order of fields in which to update the values.

Return Value
Well return the number of changed rows. So well get a 0 value if the conditional update has not been successful instead of an exception.

Basic Usage

- **Simple match**

  The most basic example is doing a simple match, for example for a `volume` variable that contains a Versioned Object Volume class instance we may want to change the `status` to deleting and update the `terminated_at` field with current UTC time only if current `status` is available and the volume is not in a consistency group.

```
values={'status': 'deleting',
        'terminated_at': timeutils.utcnow()}
expected_values = {'status': 'available',
                   'consistencygroup_id': None}

volume.conditional_update(values, expected_values)
```

- **Iterable match**

  Conditions can contain not only single values, but also iterables, and the conditional update mechanism will correctly handle the presence of None values in the range, unlike SQL `IN` clause that doesnt support `NULL` values.

```
values={'status': 'deleting',
        'terminated_at': timeutils.utcnow()}
expected_values={
    'status': ('available', 'error', 'error_restoring' 'error_extending'),
    'migration_status': (None, 'deleting', 'error', 'success'),
    'consistencygroup_id': None
```

(continues on next page)

(continued from previous page)

```
}

volume.conditional_update(values, expected_values)
```

- **Exclusion**

  In some cases well need to set conditions on what is *not* in the DB record instead of what is in, for that we will use the exclusion mechanism provided by the `Not` class in all persistent objects. This class accepts single values as well as iterables.

```
values={'status': 'deleting',
        'terminated_at': timeutils.utcnow()}
expected_values={
    'attach_status': volume.Not('attached'),
    'status': ('available', 'error', 'error_restoring' 'error_extending'),
    'migration_status': (None, 'deleting', 'error', 'success'),
    'consistencygroup_id': None
}

volume.conditional_update(values, expected_values)
```

- **Filters**

  We can use complex filters in the conditions, but these must be SQLAlchemy queries/conditions and as the rest of the DB methods must be properly abstracted from the API.

  Therefore we will create the method in cinder/db/sqlalchemy/api.py:

```
def volume_has_snapshots_filter():
    return sql.exists().where(
        and_(models.Volume.id == models.Snapshot.volume_id,
             ~models.Snapshot.deleted))
```

  Then expose this filter through the cinder/db/api.py:

```
def volume_has_snapshots_filter():
    return IMPL.volume_has_snapshots_filter()
```

  And finally used in the API (notice how we are negating the filter at the API):

```
filters = [~db.volume_has_snapshots_filter()]
values={'status': 'deleting',
        'terminated_at': timeutils.utcnow()}
expected_values={
    'attach_status': volume.Not('attached'),
    'status': ('available', 'error', 'error_restoring' 'error_extending'),
    'migration_status': (None, 'deleting', 'error', 'success'),
    'consistencygroup_id': None
}

volume.conditional_update(values, expected_values, filters)
```

Returning Errors

The most important downside of using conditional updates to remove API races is the inherent uncertainty of the cause of failure resulting in more generic error messages.

When we use the *conditional_update* method well use returned value to determine the success of the operation, as a value of 0 indicates that no rows have been updated and the conditions were not met. But we dont know which one, or which ones, were the cause of the failure.

There are 2 approaches to this issue:

- On failure we go one by one checking the conditions and return the first one that fails.
- We return a generic error message indicating all conditions that must be met for the operation to succeed.

It was decided that we would go with the second approach, because even though the first approach was closer to what we already had and would give a better user experience, it had considerable implications such as:

- More code was needed to do individual checks making operations considerable longer and less readable. This was greatly alleviated using helper methods to return the errors.
- Higher number of DB queries required to determine failure cause.
- Since there could be races because DB contents could be changed between the failed update and the follow up queries that checked the values for the specific error, a loop would be needed to make sure that either the conditional update succeeds or one of the condition checks fails.
- Having such a loop means that a small error in the code could lead to an endless loop in a production environment. This coding error could be an incorrect conditional update filter that would always fail or a missing or incorrect condition that checked for the specific issue to return the error.

A simple example of a generic error can be found in *begin_detaching* code:

```
@wrap_check_policy
def begin_detaching(self, context, volume):
    # If we are in the middle of a volume migration, we don't want the
    # user to see that the volume is 'detaching'. Having
    # 'migration_status' set will have the same effect internally.
    expected = {'status': 'in-use',
                'attach_status': 'attached',
                'migration_status': self.AVAILABLE_MIGRATION_STATUS}

    result = volume.conditional_update({'status': 'detaching'}, expected)

    if not (result or self._is_volume_migrating(volume)):
        msg = _("Unable to detach volume. Volume status must be 'in-use' "
                "and attach_status must be 'attached' to detach.")
        LOG.error(msg)
        raise exception.InvalidVolume(reason=msg)
```

Building filters on the API

SQLAlchemy filters created as mentioned above can create very powerful and complex conditions, but sometimes we may require a condition that, while more complex than the basic match and not match on the resource fields, its still quite simple. For those cases we can create filters directly on the API using the `model` field provided in Versioned Objects.

This `model` field is a reference to the ORM model that allows us to reference ORM fields.

Well use as an example changing the `status` field of a backup to restoring if the backup status is available and the volume where we are going to restore the backup is also in available state.

Joining of tables is implicit when using a model different from the one used for the Versioned Object instance.

- **As expected_values**

  Since this is a matching case we can use `expected_values` argument to make the condition:

```
values = {'status': 'restoring'}
expected_values={'status': 'available',
                 objects.Volume.model.id: volume.id,
                 objects.Volume.model.status: 'available'}
```

- **As filters**

  We can also use the `filters` argument to achieve the same results:

```
filters = [objects.Volume.model.id == volume.id,
           objects.Volume.model.status == 'available']
```

- **Other filters**

  If we are not doing a match for the condition the only available option will be to use `filters` argument. For example if we want to do a check on the volume size against the backup size:

```
filters = [objects.Volume.model.id == volume.id,
           objects.Volume.model.size >= backup.model.size]
```

Using DB fields for assignment

- **Using non modified fields**

  Similar to the way we use the fields to specify conditions, we can also use them to set values in the DB.

  For example when we disable a service we want to keep existing `updated_at` field value:

```
values = {'disabled': True,
          'updated_at': service.model.updated_at}
```

- **Using modified field**

  In some cases we may need to use a DB field that we are also updating, for example when we are updating the `status` but we also want to keep the old value in the `previous_status` field.

```
values = {'status': 'retyping',
          'previous_status': volume.model.status}
```

Conditional update mechanism takes into account that MySQL does not follow SQL language specs and adjusts the query creation accordingly.

- **Together with filters**

  Using DB fields for assignment together with using them for values can give us advanced functionality like for example increasing a quota value based on current value and making sure we dont exceed our quota limits.

```
values = {'in_use': quota.model.in_use + volume.size}
filters = [quota.model.in_use <= max_usage - volume.size]
```

Conditional value setting

Under certain circumstances you may not know what value should be set in the DB because it depends on another field or on another condition. For those cases we can use the `Case` class present in our persistent Versioned Objects which implements the SQL CASE clause.

The idea is simple, using `Case` class we can say which values to set in a field based on conditions and also set a default value if none of the conditions are True.

Conditions must be SQLAlchemy conditions, so well need to use fields from the `model` attribute.

For example setting the status to maintenance during migration if current status is available and leaving it as it was if its not can be done using the following:

```
values = {
    'status': volume.Case(
        [
            (volume.model.status == 'available', 'maintenance')
        ],
        else_=volume.model.status)
}
```

reflect_changes considerations

As weve already mentioned `conditional_update` method will update Versioned Object instance with provided values if the row in the DB has been updated, and in most cases this is OK since we can set the values directly because we are using simple values, but there are cases where we dont know what value we should set in the instance, and is in those cases where the default `reflect_changes` value of True has performance implications.

There are 2 cases where Versioned Object `conditional_update` method doesnt know the value it has to set on the Versioned Object instance, and they are when we use a field for assignment and when we are using the `Case` class, since in both cases the DB is the one deciding the value that will be set.

In those cases `conditional_update` will have to retrieve the value from the DB using `get_by_id` method, and this has a performance impact and therefore should be avoided when possible.

So the recommendation is to set `reflect_changes` to False when using `Case` class or using fields in the `values` argument if we dont care about the stored value.

Limitations

We can only use functionality that works on **all** supported DBs, and thats why we dont allow multi table updates and will raise ProgrammingError exception even when the code is running against a DB engine that supports this functionality.

This way we make sure that we dont inadvertently add a multi table update that works on MySQL but will surely fail on PostgreSQL.

MySQL DB engine also has some limitations that we should be aware of when creating our filters.

One that is very common is when we are trying to check if there is a row that matches a specific criteria in the same table that we are updating. For example, when deleting a Consistency Group we want to check that it is not being used as the source for a Consistency Group that is in the process of being created.

The straightforward way of doing this is using the core exists expression and use an alias to differentiate general query fields and the exists subquery. Code would look like this:

```
def cg_creating_from_src(cg_id):
   model = aliased(models.ConsistencyGroup)
   return sql.exists().where(and_(
       ~model.deleted,
       model.status == 'creating',
       conditions.append(model.source_cgid == cg_id)))
```

While this will work in SQLite and PostgreSQL, it will not work on MySQL and an error will be raised when the query is executed: You cant specify target table consistencygroups for update in FROM clause.

To solve this we have 2 options:

- Create a specific query for MySQL engines using an update with a left self join, which is a feature only available in MySQL.
- Use a trick -using a select subquery- that will work on all DBs.

Considering that its always better to have only 1 way of doing things and that SQLAlchemy doesnt support MySQLs non standard behavior we should generate these filters using the select subquery method like this:

```
def cg_creating_from_src(cg_id):
   subq = sql.select(models.ConsistencyGroup).where(
       and_(
           ~model.deleted,
           model.status == 'creating'
       )
   ).alias('cg2')

   return sql.exists([subq]).where(subq.c.source_cgid == cgid)
```

Considerations for new ORM & Versioned Objects

Conditional update mechanism works using generic methods for getting an object from the DB as well as determining the model for a specific Versioned Object instance for field binding.

These generic methods rely on some naming rules for Versioned Object classes, ORM classes, and get methods, so when we are creating a new ORM class and adding the matching Versioned Object and access methods we must be careful to follow these rules or at least specify exceptions if we have a good reason not to follow these conventions.

Rules:

- Versioned Object class name must be the same as the ORM class
- Get method name must be ORM class converted to snake format with postfix _get. For example, for `Volume` ORM class expected method is `volume_get`, and for an imaginary `MyORMClass` it would be `my_orm_class_get`.
- Get method must receive the `context` as the first argument and the `id` as the second one, although it may accept more optional arguments.

We should avoid diverging from these rules whenever is possible, but there are cases where this is not possible, for example `BackupImport` Versioned Object that really uses `Backup` ORM class. For cases such as this we have a way to set exceptions both for the generic get method and the model for a Versioned Object.

To add exceptions for the get method we have to add a new entry to `GET_EXCEPTIONS` dictionary mapping in `cinder.db.sqlalchemy.api._get_get_method`.

And for determining the model for the Versioned Object we have to add a new entry to `VO_TO_MODEL_EXCEPTIONS` dictionary mapping in `cinder.db.sqlalchemy.api.get_model_for_versioned_object`.

Adding a Method to the OpenStack API

The interface is a mostly RESTful API. REST stands for Representational State Transfer and provides an architecture style for distributed systems using HTTP for transport. Figure out a way to express your request and response in terms of resources that are being created, modified, read, or destroyed.

Routing

To map URLs to controllers+actions, OpenStack uses the Routes package, a clone of Rails routes for Python implementations. See http://routes.groovie.org/ for more information.

URLs are mapped to action methods on controller classes in `cinder/api/openstack/__init__/ApiRouter.__init__` .

See http://routes.readthedocs.io/en/latest/ for all syntax, but youll probably just need these two:

- mapper.connect() lets you map a single URL to a single action on a controller.
- mapper.resource() connects many standard URLs to actions on a controller.

Controllers and actions

Controllers live in `cinder/api/openstack`, and inherit from cinder.wsgi.Controller.

See `cinder/api/v3/volumes.py` for an example.

Action methods take parameters that are sucked out of the URL by mapper.connect() or .resource(). The first two parameters are self and the WebOb request, from which you can get the req.environ, req.body, req.headers, etc.

Serialization

Actions return a dictionary, and wsgi.Controller serializes that to JSON or XML based on the requests content-type.

Errors

There will be occasions when you will want to return a REST error response to the caller and there are multiple valid ways to do this:

- If you are at the controller level you can use a `faults.Fault` instance to indicate the error. You can either return the `Fault` instance as the result of the action, or raise it, depending on whats more convenient: `raise faults.Fault(webob.exc.HTTPBadRequest(explanation=msg))`.
- If you are raising an exception our WSGI middleware exception handler is smart enough to recognize webob exceptions as well, so you dont really need to wrap the exceptions in a `Fault` class and you can just let the middleware add it for you: `raise webob.exc.HTTPBadRequest(explanation=msg)`.
- While most errors require an explicit webob exception there are some Cinder exceptions (`NotFound` and `Invalid`) that are so common that they are directly handled by the middleware and dont need us to convert them, we can just raise them at any point in the API service and they will return the appropriate REST error to the caller. So any `NotFound` exception, or child class, will return a 404 error, and any `Invalid` exception, or child class, will return a 400 error.

Drivers

Cinder exposes an API to users to interact with different storage backend solutions. The following are standards across all drivers for Cinder services to properly interact with a driver.

Basic attributes

There are some basic attributes that all drivers classes should have:

- VERSION: Driver version in string format. No naming convention is imposed, although semantic versioning is recommended.
- CI_WIKI_NAME: Must be the exact name of the ThirdPartySystems wiki page. This is used by our tooling system to associate jobs to drivers and track their CI reporting status correctly.

The tooling system will also use the name and docstring of the driver class.

Configuration options

Each driver requires different configuration options set in the cinder.conf file to operate, and due to the complexities of the Object Oriented programming mechanisms (inheritance, composition, overwriting, etc.) once your driver defines its parameters in the code Cinder has no automated way of telling which configuration options are relevant to your driver.

In order to assist operators and installation tools we recommend reporting the relevant options:

- For operators: In the documentation under `doc/source/configuration/block-storage`.
- For operators and installers: Through the `get_driver_options` static method returning that returns a list of all the Oslo Config parameters.

Minimum Features

Minimum features are enforced to avoid having a grid of what features are supported by which drivers and which releases. Cinder Core requires that all drivers implement the following minimum features.

Core Functionality

- Volume Create/Delete
- Volume Attach/Detach
- Snapshot Create/Delete
- Create Volume from Snapshot
- Get Volume Stats
- Copy Image to Volume
- Copy Volume to Image
- Clone Volume
- Extend Volume

Security Requirements

- Drivers must delete volumes in a way where volumes deleted from the backend will not leak data into new volumes when they are created. Cinder operates in multi-tenant environments and this is critical to ensure data safety.
- Drivers should support secure TLS/SSL communication between the cinder volume service and the backend as configured by the driver_ssl_cert_verify and driver_ssl_cert_path options in cinder.conf.
- Drivers should use standard Python libraries to handle encryption-related functionality, and not contain custom implementations of encryption code.

Volume Stats

Volume stats are used by the different schedulers for the drivers to provide a report on their current state of the backend. The following should be provided by a driver.

- driver_version
- free_capacity_gb
- storage_protocol
- total_capacity_gb
- vendor_name
- volume_backend_name

NOTE: If the driver is unable to provide a value for free_capacity_gb or total_capacity_gb, keywords can be provided instead. Please use unknown if the backend cannot report the value or infinite if the backend has no upper limit. But, it is recommended to report real values as the Cinder scheduler assigns lowest weight to any storage backend reporting unknown or infinite.

NOTE: By default, Cinder assumes that the driver supports attached volume extending. If it doesnt, it should report online_extend_support=False. Otherwise the scheduler will attempt to perform the operation, and may leave the volume in error_extending state.

Value of `storage_protocol` is a single string representing the transport protocol used by the storage. Existing protocols are present in `cinder.common.constants` and should be used by drivers instead of string literals.

Variant values only exist for older drivers that were already reporting those values. New drivers must use non variant versions.

The `storage_protocol` can be used by operators using the `cinder get-pools --detail` command, by volume types in their extra specs, and by the filter and goodness functions.

We must not mistake the value of the `storage_protocol` with the identifier of the os-brick connector, which is returned by the `initialize_connection` driver method in the `driver_volume_type` dictionary key. In some cases they may have the same value, but they are different things.

Feature Enforcement

All concrete driver implementations should use the `cinder.interface.volumedriver` decorator on the driver class:

```
@interface.volumedriver
class LVMVolumeDriver(driver.VolumeDriver):
```

This will register the driver and allow automated compliance tests to run against and verify the compliance of the driver against the required interface to support the *Core Functionality* listed above.

Running `tox -e compliance` will verify all registered drivers comply to this interface. This can be used during development to perform self checks along the way. Any missing method calls will be identified by the compliance tests.

The details for the required volume driver interfaces can be found in the `cinder/interface/volume_*_driver.py` source.

New Driver Review Checklist

There are some common issues caught during the review of new driver patches that can easily be avoided. New driver maintainers should review the *New Driver Review Checklist* for some things to watch out for.

New Driver Review Checklist

Reviewers can use this list for some common things to watch for when doing new driver reviews. This list is by no means exhaustive, but does try to capture some of things that have been found in past reviews.

Note

Feel free to propose additional items to help make this a more complete list.

Review Checklist

- Driver Code
  - Passing all gate tests
  - Driver keeps all configuration in `cinder.conf` and not in separate vendor specific config file.
    - xml files for configs are forbidden
- Common gotchas
  - Code should use `volume.name_id` instead of `volume.id`.
  - Handles detach where `connector == None` for force detach
  - Create from snapshot and clone properly account for new volume size being larger than original volume size
  - Volume not found in delete calls should return success
  - Ensure proper code format w/ pep8 (`tox -e pep8`), but start here first: https://docs.openstack.org/hacking/latest/user/hacking.html
    - `tox -e fast8` can be used as a quick check only against modified files
  - Unit tests included for all but trivial code in driver
    - Make sure theres an `__init__.py` file in the directory containing the test files or they wont be discovered by stestr when running the generic `tox -e pyXX` command to run unit tests.
    - Use the results of the `cinder-code-coverage` job or run `tox -e cover` locally to see a test coverage report.
  - All source code files contain Apache 2 copyright header
    - Stating copyright for vendor is optional
    - Dont attribute copyright to the OpenStack Foundation
  - Run `tox -e compliance` to make sure all required interfaces are implemented.
  - Required in driver:
    - Concrete driver implementation has decorator `@interface.volumedriver`
    - `VERSION` constant defined in driver class
    - `CI_WIKI_NAME` constant defined in driver class
    - well documented version history in the comment block for the main driver class.
    - Support *minimum driver features*.
    - Meet release deadline(s)
      - By Milestone 2 of the current development cycle, the driver should have working third party CI and no code review issues.
      - You can find the exact date on the current release schedule, which you can find from https://releases.openstack.org/index.html
  - Driver does not add unnecessary new config options

    * For example, adding vendor_username instead of using the common san_login
  - Driver reports all options it uses in get_driver_options() method
    * This is necessary for cinderlib/emberCSI use of the driver
    * The response should include any common config options (see above) in addition to driver-specific options
    * See https://review.opendev.org/c/openstack/cinder/+/770807/ for an example of how to do this
  - If the driver is a subclass of an existing driver, verify that it implements its own `_update_volume_stats()` function to override any capabilities of the parent driver that the child driver may not have. For example, the parent driver may support multiattach, while this may not be the case (or may not yet be verified) for the child driver.
  - Driver specific exceptions inherit from `VolumeDriverException` or `VolumeBackendAPIException`
    * Exceptions should be defined with driver code
  - Logging level is appropriate for content
    * General tracing should be at debug level
    * Things operators should be aware of should be at Info level
    * Issues that are of concern but may not have an impact on actual operation should be warning
    * Issues operators need to take action on or should definitely know about should be ERROR
    * Messages about a failure should include the snapshot or volume in question.
  - All exception messages that could be raised to users should be marked for translation with _()
  - Cryptography
    * Drivers must not use md5 for any security-related purpose. (In fact, drivers should avoid using it at all, because some security audits only allow a yes/no checkbox for md5 use but thats up to the vendor.)
      · If md5 *is* being used for a non security-related purpose, the code must use oslo.utils and not call hashlib directly to access md5. Heres an example of how to do this: https://review.opendev.org/c/openstack/os-brick/+/756151
    * Any cryptography done by a driver should be implemented by using a well-respected cryptographic library. *Under no circumstances should a driver implement its own cryptographic functions.*

      If the library is already in OpenStack global requirements, then it is well-respected; otherwise, you will find out if its well-respected when you apply for it to be added to global requirements (see next item).
  - Any additional libraries needed for a driver must be added to the global requirements.
    * https://wiki.openstack.org/wiki/Requirements#Adding_a_Requirement_to_an_OpenStack_Project

        * Pypi installable libraries should be added to driver section in setup.cfg
        * Binary dependencies need to be OSI licensed and added to bindep.txt
    - Third Party CI checks
        * Responds correctly to recheck from run-<CI Name>
        * Tempest run console log available
        * `cinder.conf` and all cinder service logs available
        * LVM driver is not being configured in `local.conf/cinder.conf`
        * Only the driver in question should be in `cinder.conf` and enabled
            · `default_volume_type` and `enabled_backends` in `cinder.conf`, OR
            · `CINDER_DEFAULT_VOLUME_TYPE` and `CINDER_ENABLED_BACKENDS` in `local.conf`, OR
            · `TEMPEST_VOLUME_DRIVER` and `TEMPEST_VOLUME_VENDER` in `local.conf`
        * specify correct patch for each CI run
            · `CINDER_BRANCH` in `local.conf`, OR
            · `git fetch https://review.opendev.org/openstack/cinder refs/changes/56/657856/2 && git checkout cherry-pick` (https://wiki.openstack.org/wiki/Cinder/tested-3rdParty-drivers)
    - CI runs `tox -e all -- *volume*`
        * Any skipped tests need to be clearly documented why they are being skipped including the plan for getting rid of the need to skip them.
        * https://opendev.org/openstack/cinder-tempest-plugin needs to be installed so those tempest tests run as well.
        * `tox | tempest` with `--subunit` helps generate HTML output (https://docs.openstack.org/os-testr/latest/user/subunit2html.html)
        * `tox | tempest` with `--concurrency=<n>` for specifying <n> number of test runners
    - CI must run Cinder services using Python 3. More specifically:
        * At the Ussuri Virtual Mid-Cycle meeting (session 2, 16 March 2020), the Cinder team agreed that new Third-Party CI systems should:
            · ideally, test using *all* of the cycle Python runtimes
            · otherwise, test using at least one of the cycle runtimes
        * The current Python runtimes are determined by the OpenStack Technical Committee. See Tested Runtimes in the OpenStack governance documents.
    - CI does not report failures or exception due to the CI operation and not due to test failures due to code changes.
    - *optional, but highly recommended:* CI only runs on third party CI recheck trigger or on successful +1 from Zuul.
    - CI only runs on patches to the master branch unless they are intentionally set up to be able to properly run stable branch testing.

- Included with driver patch
  - Release note stating something like New volume driver added for Blah blah blah storage
    - See Reno usage information here: https://docs.openstack.org/reno/latest/user/usage.html
    - Make sure that the release note is in the correct subdirectory, namely, `releasenotes/notes/` in the repository root directory. It should *not* be located in the drivers section of the code tree.
  - Driver added to `doc/source/reference/support-matrix.ini` and `doc/source/reference/support-matrix.rst`
  - Driver configuration information added under `doc/source/configuration/block-storage/drivers`
  - Update `cinder/opts.py` including the new driver library options using the command `tox -e genopts`

Driver Development Documentations

The LVM driver is our reference for all new driver implementations. The information below can provide additional documentation for the methods that volume drivers need to implement.

Volume ID

Drivers should always get a volumes ID using the `name_id` attribute instead of the `id` attribute.

A Cinder volume may have two different UUIDs, a user facing one, and one the driver should use.

When a volume is created these two are the same, but when doing a generic migration (create new volume, then copying data) they will be different if we were unable to rename the new volume in the final migration steps.

So the volume will have been created using the new volumes UUID and the driver will have to look for it using that UUID, but the user on the other hand will keep referencing the volume with the original UUID.

Base Driver Interface

The methods documented below are the minimum required interface for a volume driver to support. All methods from this interface must be implemented in order to be an official Cinder volume driver.

Core backend volume driver interface.

All backend drivers should support this interface as a bare minimum, but some methods (marked as optional in their description) can rely on the default implementation.

class VolumeDriverCore

Core backend driver required interface.

after_volume_copy(*context*, *src_vol*, *dest_vol*, *remote=None*)

Driver-specific actions executed after copying a volume.

This method will be called after _copy_volume_data during volume migration.

Parameters

- `context` Context
- `src_volume` Source volume in the copy operation.
- `dest_volume` Destination volume in the copy operation.
- `remote` Whether the copy operation is local.

Returns
There is no return value for this method.

`before_volume_copy`(*context*, *src_vol*, *dest_vol*, *remote=None*)
Driver-specific actions executed before copying a volume.

This method will be called before _copy_volume_data during volume migration.

Parameters

- `context` Context
- `src_volume` Source volume in the copy operation.
- `dest_volume` Destination volume in the copy operation.
- `remote` Whether the copy operation is local.

Returns
There is no return value for this method.

`check_for_setup_error`()
Validate there are no issues with the driver configuration.

Called after do_setup(). Driver initialization can occur there or in this call, but must be complete by the time this returns.

If this method raises an exception, the driver will be left in an uninitialized state by the volume manager, which means that it will not be sent requests for volume operations.

This method typically checks things like whether the configured credentials can be used to log in the storage backend, and whether any external dependencies are present and working.

Raises

- *VolumeBackendAPIException* in case of setup error.
- *InvalidConfigurationValue* raise this if you detect a problem during a configuration check

`clone_image`(*context*, *volume*, *image_location*, *image_meta*, *image_service*)
Create a volume efficiently from an existing image.

Drivers that, always or under some circumstances, can efficiently create a volume from a Glance image can implement this method to be given a chance to try to do the volume creation as efficiently as possible.

If the driver cannot do it efficiently on a specific call it can return `(None, False)` to let Cinder try other mechanisms.

This method is optional and most drivers wont need to implement it and can leverage the default driver implementation that returns `(None, False)` to indicate that this optimization is not possible on this driver.

Examples where drivers can do this optimization:

- When images are stored on the same storage system and the driver can locate them and efficiently create a volume. For example the RBD driver can efficiently create a volume if the image is stored on the same Ceph cluster and the image format is `raw`. Another example is the GPFS driver.
- When volumes are locally accessible and accessing them that way is more efficient than going through the remote connection mechanism. For example in the GPFS driver if the cloning feature doesnt work it will copy the file without using os-brick to connect to the volume.

Parameters

- **context** Security/policy info for the request.
- **volume** The volume to create, as an OVO instance. Drivers should use attributes to access its values instead of using the dictionary compatibility interface it provides.
- **image_location** Tuple with (`direct_url`, `locations`) from the image metadata fields. `direct_url`, when present, is a string whose format depends on the image services external storage in use. Any, or both, tuple positions can be None, depending on the image service configuration. `locations`, when present, is a list of dictionaries where the value of the `url` key contains the direct urls (including the one from `direct_url`).
- **image_meta** Dictionary containing information about the image, including basic attributes and custom properties. Some transformations have been applied, such as converting timestamps (from `created_at`, `updated_at`, and `deleted_at`) to datetimes, and deserializing JSON values from `block_device_mapping` and `mappings` keys if present. Base properties, as per the images schema, will be stored on the base dictionary and the rest will be stored under the `properties` key. An important field to check in this method is the `disk_format` (e.g. raw, qcow2).
- **image_service** The image service to use (`GlanceImageService` instance). Can fetch image data directly using it.

Returns

Tuple of (model_update, boolean) where the boolean specifies whether the clone occurred.

`copy_image_to_volume`(*context*, *volume*, *image_service*, *image_id*, *disable_sparse=False*)

Fetch the image from image_service and write it to the volume.

Parameters

- **context** Security/policy info for the request.
- **volume** The volume to create.
- **image_service** The image service to use.
- **image_id** The image identifier.
- **disable_sparse** Enable or disable sparse copy. Default=False.

Returns

Model updates.

copy_volume_to_image(*context*, *volume*, *image_service*, *image_meta*)

Copy the volume to the specified image.

Parameters

- **context** Security/policy info for the request.
- **volume** The volume to copy.
- **image_service** The image service to use.
- **image_meta** Information about the image.

Returns

Model updates.

create_snapshot(*snapshot*)

Creates a snapshot.

Parameters

snapshot Information for the snapshot to be created.

create_volume(*volume*)

Create a new volume on the backend.

This method is responsible only for storage allocation on the backend. It should not export a LUN or actually make this storage available for use, this is done in a later call.

TODO(smcginnis): Add example data structure of volume object.

Parameters

volume Volume object containing specifics to create.

Returns

(Optional) dict of database updates for the new volume.

Raises

VolumeBackendAPIException if creation failed.

create_volume_from_snapshot(*volume*, *snapshot*)

Creates a volume from a snapshot.

If volume_type extra specs includes replication: <is> True the driver needs to create a volume replica (secondary), and setup replication between the newly created volume and the secondary volume.

An optional larger size for the new volume can be specified. Drivers should check this value and create or expand the new volume to match.

Parameters

- **volume** The volume to be created.
- **snapshot** The snapshot from which to create the volume.

Returns

A dict of database updates for the new volume.

delete_snapshot(*snapshot*)

Deletes a snapshot.

Parameters
`snapshot` The snapshot to delete.

`delete_volume`(*volume*)
Delete a volume from the backend.

If the driver can talk to the backend and detects that the volume is no longer present, this call should succeed and allow Cinder to complete the process of deleting the volume.

It is imperative that this operation ensures that the data from the deleted volume cannot leak into new volumes when they are created, as new volumes are likely to belong to a different tenant/project.

Parameters
`volume` The volume to delete.

Raises
`VolumeIsBusy` if the volume is still attached or has snapshots. VolumeBackendAPIException on error.

`do_setup`(*context*)
Any initialization the volume driver needs to do while starting.

Called once by the manager after the driver is loaded. Can be used to set up clients, check licenses, set up protocol specific helpers, etc.

If you choose to raise an exception here, the setup is considered failed already and the check_for_setup_error() will not be called.

Parameters
`context` The admin context of type context.RequestContext.

Raises

- *`InvalidConfigurationValue`* raise this if you detect a problem during a configuration check
- *`VolumeDriverException`* raise this or one of its more specific subclasses if you detect setup problems other than invalid configuration

`extend_volume`(*volume*, *new_size*)
Extend the size of a volume.

Parameters

- `volume` The volume to extend.
- `new_size` The new desired size of the volume.

Note that if the volume backend doesnt support extending an in-use volume, the driver should report online_extend_support=False.

`get_volume_stats`(*refresh=False*)
Collects volume backend stats.

The get_volume_stats method is used by the volume manager to collect information from the driver instance related to information about the driver, available and used space, and driver/backend capabilities.

stats are stored in self._stats field, which could be updated in _update_volume_stats method.

It returns a dict with the following required fields:

- **volume_backend_name**
  This is an identifier for the backend taken from cinder.conf. Useful when using multi-backend.
- **vendor_name**
  Vendor/author of the driver who serves as the contact for the drivers development and support.
- **driver_version**
  The driver version is logged at cinder-volume startup and is useful for tying volume service logs to a specific release of the code. There are currently no rules for how or when this is updated, but it tends to follow typical major.minor.revision ideas.
- **storage_protocol**
  The protocol used to connect to the storage, this should be a short string such as: iSCSI, FC, NFS, ceph, etc. Available protocols are present in cinder.common.constants and they must be used instead of string literals. Variant values only exist for older drivers that were already reporting those values. New drivers must use non variant versions. In some cases this may be the same value as the driver_volume_type returned by the initialize_connection method, but they are not the same thing, since this one is meant to be used by the scheduler, while the latter is the os-brick connector identifier used in the factory method.
- **total_capacity_gb**
  The total capacity in gigabytes (GiB) of the storage backend being used to store Cinder volumes. Use keyword unknown if the backend cannot report the value or infinite if there is no upper limit. But, it is recommended to report real values as the Cinder scheduler assigns lowest weight to any storage backend reporting unknown or infinite.
- **free_capacity_gb**
  The free capacity in gigabytes (GiB). Use keyword unknown if the backend cannot report the value or infinite if there is no upper limit. But, it is recommended to report real values as the Cinder scheduler assigns lowest weight to any storage backend reporting unknown or infinite.

And the following optional fields:

- **reserved_percentage (integer)**
  Percentage of backend capacity which is not used by the scheduler.
- **location_info (string)**
  Driver-specific information used by the driver and storage backend to correlate Cinder volumes and backend LUNs/files.
- **QoS_support (Boolean)**
  Whether the backend supports quality of service.
- **provisioned_capacity_gb**
  The total provisioned capacity on the storage backend, in gigabytes (GiB), including space consumed by any user other than Cinder itself.
- **max_over_subscription_ratio**
  The maximum amount a backend can be over subscribed.

- **thin_provisioning_support (Boolean)**
  Whether the backend is capable of allocating thinly provisioned volumes.
- **thick_provisioning_support (Boolean)**
  Whether the backend is capable of allocating thick provisioned volumes. (Typically True.)
- **total_volumes (integer)**
  Total number of volumes on the storage backend. This can be used in custom driver filter functions.
- **filter_function (string)**
  A custom function used by the scheduler to determine whether a volume should be allocated to this backend or not. Example:

  capabilities.total_volumes < 10
- **goodness_function (string)**
  Similar to filter_function, but used to weigh multiple volume backends. Example:

  capabilities.capacity_utilization < 0.6 ? 100 : 25
- **multiattach (Boolean)**
  Whether the backend supports multiattach or not. Defaults to False.
- **sparse_copy_volume (Boolean)**
  Whether copies performed by the volume manager for operations such as migration should attempt to preserve sparseness.
- **online_extend_support (Boolean)**
  Whether the backend supports in-use volume extend or not. Defaults to True.
- **clone_across_pools (Boolean)**
  Whether the backend supports cloning a volume across different pools. Defaults to False.

The returned dict may also contain a list, pools, which has a similar dict for each pool being used with the backend.

Parameters
`refresh` Whether to discard any cached values and force a full refresh of stats.

Returns
dict of appropriate values (see above).

`init_capabilities()`

Fetch and merge capabilities of the driver.

Do not override this, implement _init_vendor_properties instead.

`initialize_connection`(*volume*, *connector*, *initiator_data=None*)

Allow connection to connector and return connection info.

Parameters

- `volume` The volume to be attached.
- `connector` Dictionary containing information about what is being connected to.

- **initiator_data** (Optional) A dictionary of driver_initiator_data objects with key-value pairs that have been saved for this initiator by a driver in previous initialize_connection calls.

Returns
A dictionary of connection information. This can optionally include a initiator_updates field.

The initiator_updates field must be a dictionary containing a set_values and/or remove_values field. The set_values field must be a dictionary of key-value pairs to be set/updated in the db. The remove_values field must be a list of keys, previously set with set_values, that will be deleted from the db.

May be called multiple times to get connection information after a volume has already been attached.

initialized()

Getter for drivers initialized status.

Do not implement this in a driver. Rely on the default implementation.

migrate_volume(*context*, *volume*, *host*)

Migrate the volume to the specified host.

Parameters

- **context** Context
- **volume** A dictionary describing the volume to migrate
- **host** A dictionary describing the host to migrate to, where host[host] is its name, and host[capabilities] is a dictionary of its reported capabilities.

Returns
Tuple of (model_update, boolean) where the boolean specifies whether the migration occurred.

retype(*context*, *volume*, *new_type*, *diff*, *host*)

Change the type of a volume.

This operation occurs on the same backend and the return value indicates whether it was successful. If migration is required to satisfy a retype, that will be handled by the volume manager.

Parameters

- **context** Context
- **volume** The volume to retype
- **new_type** The target type for the volume
- **diff** The differences between the two types
- **host** The host that contains this volume

Returns
Tuple of (boolean, model_update) where the boolean specifies whether the retype occurred.

`set_initialized()`
Mark driver as initialized.

Do not implement this in a driver. Rely on the default implementation.

`set_throttle()`
Hook for initialization of cinder.volume.throttle.

This has not been necessary to re-implement or override in any drivers thus far. The generic implementation does nothing unless explicitly enabled.

`supported()`
Getter for drivers supported status.

Do not implement this in a driver. Rely on the default implementation.

`terminate_connection`(*volume*, *connector*)
Remove access to a volume.

Note: If `connector` is `None`, then all connections to the volume should be terminated.

Parameters

- **volume** The volume to remove.
- **connector** The Dictionary containing information about the connection. This is optional when doing a force-detach and can be None.

`update_migrated_volume`(*context*, *volume*, *new_volume*, *original_volume_status*)
Return model update for migrated volume.

Each driver implementing this method needs to be responsible for the values of _name_id and provider_location. If None is returned or either key is not set, it means the volume table does not need to change the value(s) for the key(s). The return format is {_name_id: value, provider_location: value}.

Parameters

- **context** Context
- **volume** The original volume that was migrated to this backend
- **new_volume** The migration volume object that was created on this backend as part of the migration process
- **original_volume_status** The status of the original volume

Returns
model_update to update DB with any needed changes

`update_provider_info`(*volumes*, *snapshots*)
Get provider info updates from driver.

This retrieves a list of volumes and a list of snapshots that changed their providers thanks to the initialization of the host, so that Cinder can update this information in the volume database.

This is only implemented by drivers where such migration is possible.

Parameters

- **volumes** List of Cinder volumes to check for updates

- **snapshots** List of Cinder snapshots to check for updates

Returns
tuple (volume_updates, snapshot_updates)

where volume updates {id: uuid, provider_id: <provider-id>} and snapshot updates {id: uuid, provider_id: <provider-id>}

Manage/Unmanage Support

An optional feature a volume backend can support is the ability to manage existing volumes or unmanage volumes - keep the volume on the storage backend but no longer manage it through Cinder.

To support this functionality, volume drivers must implement these methods:

Manage/unmanage existing volume driver interface.

class VolumeListManageableDriver

Interface to support listing manageable snapshots and volumes.

get_manageable_snapshots(*cinder_snapshots*, *marker*, *limit*, *offset*, *sort_keys*, *sort_dirs*)

List snapshots on the backend available for management by Cinder.

Returns a list of dictionaries, each specifying a snapshot in the host, with the following keys:

- reference (dictionary): The reference for a snapshot, which can be passed to manage_existing_snapshot.
- size (int): The size of the snapshot according to the storage backend, rounded up to the nearest GB.
- safe_to_manage (boolean): Whether or not this snapshot is safe to manage according to the storage backend. For example, is the snapshot in use or invalid for any reason.
- reason_not_safe (string): If safe_to_manage is False, the reason why.
- cinder_id (string): If already managed, provide the Cinder ID.
- extra_info (string): Any extra information to return to the user
- source_reference (string): Similar to reference, but for the snapshots source volume.

Parameters

- **cinder_snapshots** A list of snapshots in this host that Cinder currently manages, used to determine if a snapshot is manageable or not.
- **marker** The last item of the previous page; we return the next results after this value (after sorting)
- **limit** Maximum number of items to return
- **offset** Number of items to skip after marker
- **sort_keys** List of keys to sort results by (valid keys are identifier and size)
- **sort_dirs** List of directions to sort by, corresponding to sort_keys (valid directions are asc and desc)

`get_manageable_volumes`(*cinder_volumes*, *marker*, *limit*, *offset*, *sort_keys*, *sort_dirs*)

List volumes on the backend available for management by Cinder.

Returns a list of dictionaries, each specifying a volume in the host, with the following keys:

- reference (dictionary): The reference for a volume, which can be passed to manage_existing.
- size (int): The size of the volume according to the storage backend, rounded up to the nearest GB.
- safe_to_manage (boolean): Whether or not this volume is safe to manage according to the storage backend. For example, is the volume in use or invalid for any reason.
- reason_not_safe (string): If safe_to_manage is False, the reason why.
- cinder_id (string): If already managed, provide the Cinder ID.
- extra_info (string): Any extra information to return to the user

Parameters

- **cinder_volumes** A list of volumes in this host that Cinder currently manages, used to determine if a volume is manageable or not.
- **marker** The last item of the previous page; we return the next results after this value (after sorting)
- **limit** Maximum number of items to return
- **offset** Number of items to skip after marker
- **sort_keys** List of keys to sort results by (valid keys are identifier and size)
- **sort_dirs** List of directions to sort by, corresponding to sort_keys (valid directions are asc and desc)

`class VolumeManagementDriver`

Interface for drivers that support managing existing volumes.

`manage_existing`(*volume*, *existing_ref*)

Brings an existing backend storage object under Cinder management.

existing_ref is passed straight through from the API requests manage_existing_ref value, and it is up to the driver how this should be interpreted. It should be sufficient to identify a storage object that the driver should somehow associate with the newly-created cinder volume structure.

There are two ways to do this:

1. Rename the backend storage object so that it matches the, volume[name] which is how drivers traditionally map between a cinder volume and the associated backend storage object.
2. Place some metadata on the volume, or somewhere in the backend, that allows other driver requests (e.g. delete, clone, attach, detach) to locate the backend storage object when required.

If the existing_ref doesnt make sense, or doesnt refer to an existing backend storage object, raise a ManageExistingInvalidReference exception.

The volume may have a volume_type, and the driver can inspect that and compare against the properties of the referenced backend storage object. If they are incompatible, raise a ManageExistingVolumeTypeMismatch, specifying a reason for the failure.

Parameters

- `volume` Cinder volume to manage
- `existing_ref` Dictionary with keys source-id, source-name with driver-specific values to identify a backend storage object.

Raises

- *ManageExistingInvalidReference* If the existing_ref doesnt make sense, or doesnt refer to an existing backend storage object.
- *ManageExistingVolumeTypeMismatch* If there is a mismatch between the volume type and the properties of the existing backend storage object.

`manage_existing_get_size`(*volume*, *existing_ref*)

Return size of volume to be managed by manage_existing.

When calculating the size, round up to the next GB.

Parameters

- `volume` Cinder volume to manage
- `existing_ref` Dictionary with keys source-id, source-name with driver-specific values to identify a backend storage object.

Raises

ManageExistingInvalidReference If the existing_ref doesnt make sense, or doesnt refer to an existing backend storage object.

`unmanage`(*volume*)

Removes the specified volume from Cinder management.

Does not delete the underlying backend storage object.

For most drivers, this will not need to do anything. However, some drivers might use this call as an opportunity to clean up any Cinder-specific configuration that they have associated with the backend storage object.

Parameters

`volume` Cinder volume to unmanage

Manage/Unmanage Snapshot Support

In addition to the ability to manage and unmanage volumes, Cinder backend drivers may also support managing and unmanaging volume snapshots. These additional methods must be implemented to support these operations.

Manage/unmanage existing volume snapshots driver interface.

`class VolumeSnapshotManagementDriver`

Interface for drivers that support managing existing snapshots.

manage_existing_snapshot(*snapshot*, *existing_ref*)

Brings an existing backend storage object under Cinder management.

existing_ref is passed straight through from the API requests manage_existing_ref value, and it is up to the driver how this should be interpreted. It should be sufficient to identify a storage object that the driver should somehow associate with the newly-created cinder snapshot structure.

There are two ways to do this:

1. Rename the backend storage object so that it matches the snapshot[name] which is how drivers traditionally map between a cinder snapshot and the associated backend storage object.
2. Place some metadata on the snapshot, or somewhere in the backend, that allows other driver requests (e.g. delete) to locate the backend storage object when required.

Parameters

- **snapshot** The snapshot to manage.
- **existing_ref** Dictionary with keys source-id, source-name with driver-specific values to identify a backend storage object.

Raises

ManageExistingInvalidReference If the existing_ref doesnt make sense, or doesnt refer to an existing backend storage object.

manage_existing_snapshot_get_size(*snapshot*, *existing_ref*)

Return size of snapshot to be managed by manage_existing.

When calculating the size, round up to the next GB.

Parameters

- **snapshot** The snapshot to manage.
- **existing_ref** Dictionary with keys source-id, source-name with driver-specific values to identify a backend storage object.

Raises

ManageExistingInvalidReference If the existing_ref doesnt make sense, or doesnt refer to an existing backend storage object.

unmanage_snapshot(*snapshot*)

Removes the specified snapshot from Cinder management.

Does not delete the underlying backend storage object.

For most drivers, this will not need to do anything. However, some drivers might use this call as an opportunity to clean up any Cinder-specific configuration that they have associated with the backend storage object.

Parameters

snapshot The snapshot to unmanage.

Volume Consistency Groups

Some storage backends support the ability to group volumes and create write consistent snapshots across the group. In order to support these operations, the following interface must be implemented by the driver.

Consistency group volume driver interface.

class VolumeConsistencyGroupDriver

Interface for drivers that support consistency groups.

create_cgsnapshot(*context*, *cgsnapshot*, *snapshots*)

Creates a cgsnapshot.

Parameters

- **context** the context of the caller.
- **cgsnapshot** the dictionary of the cgsnapshot to be created.
- **snapshots** a list of snapshot dictionaries in the cgsnapshot.

Returns

model_update, snapshots_model_update

param snapshots is retrieved directly from the db. It is a list of cinder.db.sqlalchemy.models.Snapshot to be precise. It cannot be assigned to snapshots_model_update. snapshots_model_update is a list of dictionaries. It has to be built by the driver. An entry will be in this format: {id: xxx, status: xxx, }. model_update will be in this format: {status: xxx, }.

The driver should populate snapshots_model_update and model_update and return them.

The manager will check snapshots_model_update and update db accordingly for each snapshot. If the driver successfully deleted some snapshots but failed to delete others, it should set statuses of the snapshots accordingly so that the manager can update db correctly.

If the status in any entry of snapshots_model_update is error, the status in model_update will be set to the same if it is not already error.

If the status in model_update is error, the manager will raise an exception and the status of cgsnapshot will be set to error in the db. If snapshots_model_update is not returned by the driver, the manager will set the status of every snapshot to error in the except block.

If the driver raises an exception during the operation, it will be caught by the try-except block in the manager and the statuses of cgsnapshot and all snapshots will be set to error.

For a successful operation, the driver can either build the model_update and snapshots_model_update and return them or return None, None. The statuses of cgsnapshot and all snapshots will be set to available at the end of the manager function.

create_consistencygroup(*context*, *group*)

Creates a consistencygroup.

Parameters

- **context** the context of the caller.
- **group** the dictionary of the consistency group to be created.

Returns
model_update

model_update will be in this format: {status: xxx, }.

If the status in model_update is error, the manager will throw an exception and it will be caught in the try-except block in the manager. If the driver throws an exception, the manager will also catch it in the try-except block. The group status in the db will be changed to error.

For a successful operation, the driver can either build the model_update and return it or return None. The group status will be set to available.

`create_consistencygroup_from_src`(*context*, *group*, *volumes*, *cgsnapshot=None*, *snapshots=None*, *source_cg=None*, *source_vols=None*)

Creates a consistencygroup from source.

Parameters

- **context** the context of the caller.
- **group** the dictionary of the consistency group to be created.
- **volumes** a list of volume dictionaries in the group.
- **cgsnapshot** the dictionary of the cgsnapshot as source.
- **snapshots** a list of snapshot dictionaries in the cgsnapshot.
- **source_cg** the dictionary of a consistency group as source.
- **source_vols** a list of volume dictionaries in the source_cg.

Returns
model_update, volumes_model_update

The source can be cgsnapshot or a source cg.

param volumes is retrieved directly from the db. It is a list of cinder.db.sqlalchemy.models.Volume to be precise. It cannot be assigned to volumes_model_update. volumes_model_update is a list of dictionaries. It has to be built by the driver. An entry will be in this format: {id: xxx, status: xxx, }. model_update will be in this format: {status: xxx, }.

To be consistent with other volume operations, the manager will assume the operation is successful if no exception is thrown by the driver. For a successful operation, the driver can either build the model_update and volumes_model_update and return them or return None, None.

`delete_cgsnapshot`(*context*, *cgsnapshot*, *snapshots*)

Deletes a cgsnapshot.

Parameters

- **context** the context of the caller.
- **cgsnapshot** the dictionary of the cgsnapshot to be deleted.
- **snapshots** a list of snapshot dictionaries in the cgsnapshot.

Returns
model_update, snapshots_model_update

param snapshots is retrieved directly from the db. It is a list of cinder.db.sqlalchemy.models.Snapshot to be precise. It cannot be assigned to snapshots_model_update. snapshots_model_update is a list of dictionaries. It has to be built by the driver. An entry will be in this format: {id: xxx, status: xxx, }. model_update will be in this format: {status: xxx, }.

The driver should populate snapshots_model_update and model_update and return them.

The manager will check snapshots_model_update and update db accordingly for each snapshot. If the driver successfully deleted some snapshots but failed to delete others, it should set statuses of the snapshots accordingly so that the manager can update db correctly.

If the status in any entry of snapshots_model_update is error_deleting or error, the status in model_update will be set to the same if it is not already error_deleting or error.

If the status in model_update is error_deleting or error, the manager will raise an exception and the status of cgsnapshot will be set to error in the db. If snapshots_model_update is not returned by the driver, the manager will set the status of every snapshot to error in the except block.

If the driver raises an exception during the operation, it will be caught by the try-except block in the manager and the statuses of cgsnapshot and all snapshots will be set to error.

For a successful operation, the driver can either build the model_update and snapshots_model_update and return them or return None, None. The statuses of cgsnapshot and all snapshots will be set to deleted after the manager deletes them from db.

`delete_consistencygroup`(*context*, *group*, *volumes*)

Deletes a consistency group.

Parameters

- **context** the context of the caller.
- **group** the dictionary of the consistency group to be deleted.
- **volumes** a list of volume dictionaries in the group.

Returns

model_update, volumes_model_update

param volumes is retrieved directly from the db. It is a list of cinder.db.sqlalchemy.models.Volume to be precise. It cannot be assigned to volumes_model_update. volumes_model_update is a list of dictionaries. It has to be built by the driver. An entry will be in this format: {id: xxx, status: xxx, }. model_update will be in this format: {status: xxx, }.

The driver should populate volumes_model_update and model_update and return them.

The manager will check volumes_model_update and update db accordingly for each volume. If the driver successfully deleted some volumes but failed to delete others, it should set statuses of the volumes accordingly so that the manager can update db correctly.

If the status in any entry of volumes_model_update is error_deleting or error, the status in model_update will be set to the same if it is not already error_deleting or error.

If the status in model_update is error_deleting or error, the manager will raise an exception and the status of the group will be set to error in the db. If volumes_model_update is not returned by the driver, the manager will set the status of every volume in the group to error in the except block.

If the driver raises an exception during the operation, it will be caught by the try-except block in the manager. The statuses of the group and all volumes in it will be set to error.

For a successful operation, the driver can either build the model_update and volumes_model_update and return them or return None, None. The statuses of the group and all volumes will be set to deleted after the manager deletes them from db.

`update_consistencygroup`(*context*, *group*, *add_volumes=None*, *remove_volumes=None*)

Updates a consistency group.

Parameters

- **context** the context of the caller.
- **group** the dictionary of the consistency group to be updated.
- **add_volumes** a list of volume dictionaries to be added.
- **remove_volumes** a list of volume dictionaries to be removed.

Returns

model_update, add_volumes_update, remove_volumes_update

model_update is a dictionary that the driver wants the manager to update upon a successful return. If None is returned, the manager will set the status to available.

add_volumes_update and remove_volumes_update are lists of dictionaries that the driver wants the manager to update upon a successful return. Note that each entry requires a {id: xxx} so that the correct volume entry can be updated. If None is returned, the volume will remain its original status. Also note that you cannot directly assign add_volumes to add_volumes_update as add_volumes is a list of cinder.db.sqlalchemy.models.Volume objects and cannot be used for db update directly. Same with remove_volumes.

If the driver throws an exception, the status of the group as well as those of the volumes to be added/removed will be set to error.

Generic Volume Groups

The generic volume groups feature provides the ability to manage a group of volumes together. Because this feature is implemented at the manager level, every driver gets this feature by default. If a driver wants to override the default behavior to support additional functionalities such as consistent group snapshot, the following interface must be implemented by the driver. Once every driver supporting volume consistency groups has added the consistent group snapshot capability to generic volume groups, we no longer need the volume consistency groups interface listed above.

Generic volume group volume driver interface.

class VolumeGroupDriver

Interface for drivers that support groups.

`create_group`(*context*, *group*)

Creates a group.

Parameters

- **context** the context of the caller.
- **group** the Group object to be created.

Returns
model_update

model_update will be in this format: {status: xxx, }.

If the status in model_update is error, the manager will throw an exception and it will be caught in the try-except block in the manager. If the driver throws an exception, the manager will also catch it in the try-except block. The group status in the db will be changed to error.

For a successful operation, the driver can either build the model_update and return it or return None. The group status will be set to available.

`create_group_from_src`(*context*, *group*, *volumes*, *group_snapshot=None*, *snapshots=None*, *source_group=None*, *source_vols=None*)

Creates a group from source.

Parameters

- `context` the context of the caller.
- `group` the Group object to be created.
- `volumes` a list of Volume objects in the group.
- `group_snapshot` the GroupSnapshot object as source.
- `snapshots` a list of Snapshot objects in the group_snapshot.
- `source_group` a Group object as source.
- `source_vols` a list of Volume objects in the source_group.

Returns
model_update, volumes_model_update

The source can be group_snapshot or a source group.

param volumes is a list of objects retrieved from the db. It cannot be assigned to volumes_model_update. volumes_model_update is a list of dictionaries. It has to be built by the driver. An entry will be in this format: {id: xxx, status: xxx, }. model_update will be in this format: {status: xxx, }.

To be consistent with other volume operations, the manager will assume the operation is successful if no exception is thrown by the driver. For a successful operation, the driver can either build the model_update and volumes_model_update and return them or return None, None.

`create_group_snapshot`(*context*, *group_snapshot*, *snapshots*)

Creates a group_snapshot.

Parameters

- `context` the context of the caller.
- `group_snapshot` the GroupSnapshot object to be created.
- `snapshots` a list of Snapshot objects in the group_snapshot.

Returns
model_update, snapshots_model_update

param snapshots is a list of Snapshot objects. It cannot be assigned to snapshots_model_update. snapshots_model_update is a list of dictionaries. It has to be built by the driver. An entry will be in this format: {id: xxx, status: xxx, }. model_update will be in this format: {status: xxx, }.

The driver should populate snapshots_model_update and model_update and return them.

The manager will check snapshots_model_update and update db accordingly for each snapshot. If the driver successfully deleted some snapshots but failed to delete others, it should set statuses of the snapshots accordingly so that the manager can update db correctly.

If the status in any entry of snapshots_model_update is error, the status in model_update will be set to the same if it is not already error.

If the status in model_update is error, the manager will raise an exception and the status of group_snapshot will be set to error in the db. If snapshots_model_update is not returned by the driver, the manager will set the status of every snapshot to error in the except block.

If the driver raises an exception during the operation, it will be caught by the try-except block in the manager and the statuses of group_snapshot and all snapshots will be set to error.

For a successful operation, the driver can either build the model_update and snapshots_model_update and return them or return None, None. The statuses of group_snapshot and all snapshots will be set to available at the end of the manager function.

`delete_group`(*context*, *group*, *volumes*)

Deletes a group.

Parameters

- `context` the context of the caller.
- `group` the Group object to be deleted.
- `volumes` a list of Volume objects in the group.

Returns

model_update, volumes_model_update

param volumes is a list of objects retrieved from the db. It cannot be assigned to volumes_model_update. volumes_model_update is a list of dictionaries. It has to be built by the driver. An entry will be in this format: {id: xxx, status: xxx, }. model_update will be in this format: {status: xxx, }.

The driver should populate volumes_model_update and model_update and return them.

The manager will check volumes_model_update and update db accordingly for each volume. If the driver successfully deleted some volumes but failed to delete others, it should set statuses of the volumes accordingly so that the manager can update db correctly.

If the status in any entry of volumes_model_update is error_deleting or error, the status in model_update will be set to the same if it is not already error_deleting or error.

If the status in model_update is error_deleting or error, the manager will raise an exception and the status of the group will be set to error in the db. If volumes_model_update is not returned by the driver, the manager will set the status of every volume in the group to error in the except block.

If the driver raises an exception during the operation, it will be caught by the try-except block in the manager. The statuses of the group and all volumes in it will be set to error.

For a successful operation, the driver can either build the model_update and volumes_model_update and return them or return None, None. The statuses of the group and all volumes will be set to deleted after the manager deletes them from db.

`delete_group_snapshot`(*context*, *group_snapshot*, *snapshots*)

Deletes a group_snapshot.

Parameters

- `context` the context of the caller.
- `group_snapshot` the GroupSnapshot object to be deleted.
- `snapshots` a list of Snapshot objects in the group_snapshot.

Returns

model_update, snapshots_model_update

param snapshots is a list of objects. It cannot be assigned to snapshots_model_update. snapshots_model_update is a list of of dictionaries. It has to be built by the driver. An entry will be in this format: {id: xxx, status: xxx, }. model_update will be in this format: {status: xxx, }.

The driver should populate snapshots_model_update and model_update and return them.

The manager will check snapshots_model_update and update db accordingly for each snapshot. If the driver successfully deleted some snapshots but failed to delete others, it should set statuses of the snapshots accordingly so that the manager can update db correctly.

If the status in any entry of snapshots_model_update is error_deleting or error, the status in model_update will be set to the same if it is not already error_deleting or error.

If the status in model_update is error_deleting or error, the manager will raise an exception and the status of group_snapshot will be set to error in the db. If snapshots_model_update is not returned by the driver, the manager will set the status of every snapshot to error in the except block.

If the driver raises an exception during the operation, it will be caught by the try-except block in the manager and the statuses of group_snapshot and all snapshots will be set to error.

For a successful operation, the driver can either build the model_update and snapshots_model_update and return them or return None, None. The statuses of group_snapshot and all snapshots will be set to deleted after the manager deletes them from db.

`update_group`(*context*, *group*, *add_volumes=None*, *remove_volumes=None*)

Updates a group.

Parameters

- `context` the context of the caller.
- `group` the Group object to be updated.
- `add_volumes` a list of Volume objects to be added.
- `remove_volumes` a list of Volume objects to be removed.

Returns

model_update, add_volumes_update, remove_volumes_update model_update is a dictionary that the driver wants the manager to update upon a successful return. If None is returned, the manager will set the status to available.

add_volumes_update and remove_volumes_update are lists of dictionaries that the driver wants the manager to update upon a successful return. Note that each entry requires a {id: xxx} so that the correct volume entry can be updated. If None is returned, the volume will remain its original status. Also note that you cannot directly assign add_volumes to add_volumes_update as add_volumes is a list of volume objects and cannot be used for db update directly. Same with remove_volumes.

If the driver throws an exception, the status of the group as well as those of the volumes to be added/removed will be set to error.

Revert To Snapshot

Some storage backends support the ability to revert a volume to the last snapshot. To support snapshot revert, the following interface must be implemented by the driver.

Revert to snapshot capable volume driver interface.

class VolumeSnapshotRevertDriver

Interface for drivers that support revert to snapshot.

revert_to_snapshot(*context*, *volume*, *snapshot*)

Revert volume to snapshot.

Note: the revert process should not change the volumes current size, that means if the driver shrank the volume during the process, it should extend the volume internally.

Parameters

- **context** the context of the caller.
- **volume** The volume to be reverted.
- **snapshot** The snapshot used for reverting.

High Availability

In this guide well go over design and programming considerations related to high availability in Cinder.

The document aims to provide a single point of truth in all matters related to Cinders high availability.

Cinder developers must always have these aspects present during the design and programming of the Cinder core code, as well as the drivers code.

Most topics will focus on Active-Active deployments. Some topics covering node and process concurrency will also apply to Active-Passive deployments.

Overview

There are 4 services that must be considered when looking at a highly available Cinder deployment: API, Scheduler, Volume, Backup.

Each of these services has its own challenges and mechanisms to support concurrent and multi node code execution.

This document provides a general overview of Cinder aspects related to high availability, together with implementation details. Given the breadth and depth required to properly explain them all, it will fall short in some places. It will provide external references to expand on some of the topics hoping to help better understand them.

Some of the topics that will be covered are:

- Job distribution.
- Message queues.
- Threading model.
- Versioned Objects used for rolling upgrades.
- Heartbeat system.
- Mechanism used to clean up out of service cluster nodes.
- Mutual exclusion mechanisms used in Cinder.

Its good to keep in mind that Cinder threading model is based on eventlets green threads. Some Cinder and driver code may use native threads to prevent thread blocking, but thats not the general rule.

Throughout the document well be referring to clustered and non clustered Volume services. This distinction is not based on the number of services running, but on their configurations.

A non clustered Volume service is one that will be deployed as Active-Passive and has not been included in a Cinder cluster.

On the other hand, a clustered Volume service is one that can be deployed as Active-Active because it is part of a Cinder cluster. We consider a Volume service to be clustered even when there is only one node in the cluster.

Job distribution

Cinder uses RPC calls to pass jobs to Scheduler, Volume, and Backup services. A message broker is used for the transport layer on the RPC calls and parameters.

Job distribution is handled by the message broker using message queues. The different services, except the API, listen on specific message queues for RPC calls.

Based on the maximum number of nodes that will connect, we can differentiate two types of message queues: those with a single listener and those with multiple listeners.

We use single listener queues to send RPC calls to a specific service in a node. For example, when the API calls a non clustered Volume service to create a snapshot.

Message queues having multiple listeners are used in operations such as:

- Creating any volume. Call made from the API to the Scheduler.
- Creating a volume in a clustered Volume service. Call made from the Scheduler to the Volume service.
- Attaching a volume in a clustered Volume service. Call made from the API to the Volume service.

Regardless of the number of listeners, all the above mentioned RPC calls are unicast calls. The caller will place the request in a queue in the message broker and a single node will retrieve it and execute the call.

There are other kinds of RPC calls, those where we broadcast a single RPC call to multiple nodes. The best example of this type of call is the Volume service capabilities report sent to all the Schedulers.

Message queues are fair queues and are used to distribute jobs in a round robin fashion. Single target RPC calls made to message queues with multiple listeners are distributed in round robin. So sending three request to a cluster of 3 Schedulers will send one request to each one.

Distribution is content and workload agnostic. A node could be receiving all the quick and easy jobs while another one gets all the heavy lifting and its ongoing workload keeps increasing.

Cinders job distribution mechanism allows fine grained control over who to send RPC calls. Even on clustered Volume services we can still access individual nodes within the cluster. So developers must pay attention to where they want to send RPC calls and ask themselves: Is the target a clustered service? Is the RPC call intended for *any* node running the service? Is it for a *specific* node? For *all* nodes?

The code in charge of deciding the target message queue, therefore the recipient, is in the *rpcapi.py* files. Each service has its own file with the RPC calls: *volume/rpcapi.py*, *scheduler/rpcapi.py*, and *backup/rpcapi.py*.

For RPC calls the different *rcpapi.py* files ultimately use the *_get_cctxt* method from the *cinder.rpc.RPCAPI* class.

For a detailed description on the issue, ramifications, and solutions, please refer to the Cinder Volume Job Distribution.

The RabbitMQ tutorials are a good way to understand message brokers general topics.

Heartbeats

Cinder services, with the exception of API services, have a periodic heartbeat to indicate they are up and running.

When services are having health issues, they may decide to stop reporting heartbeats, even if they are running. This happens during initialization if the driver cannot be setup correctly.

The database is used to report service heartbeats. Fields *report_count* and *updated_at*, in the *services* table, keep a heartbeat counter and the last time the counter was updated.

There will be multiple database entries for Cinder Volume services running multiple backends. One per backend.

Using a date-time to mark the moment of the last heartbeat makes the system time relevant for Cinders operation. A significant difference in system times on our nodes could cause issues in a Cinder deployment.

All services report and expect the *updated_at* field to be UTC.

To determine if a service is up, we check the time of the last heartbeat to confirm that its not older than *service_down_time* seconds. Default value for *service_down_time* configuration option is 60 seconds.

Cinder uses method *is_up*, from the *Service* and *Cluster* Versioned Object, to ensure consistency in the calculations across the whole code base.

Heartbeat frequency in Cinder services is determined by the *report_interval* configuration option. The default is 10 seconds, allowing network and database interruptions.

Cinder protects itself against some incorrect configurations. If *report_interval* is greater or equal than *service_down_time*, Cinder will log a warning and use a service down time of two and a half times the configured *report_interval*.

Note

It is of utter importance having the same *service_down_time* and *report_interval* configuration options in all your nodes.

In each services section well expand this topic with specific information only relevant to that service.

Cleanup

Power outages, hardware failures, unintended reboots, and software errors. These are all events that could make a Cinder service unexpectedly halt its execution.

A running Cinder service is usually carrying out actions on resources. So when the service dies unexpectedly, it will abruptly stop those operations. Stopped operations in this way leaves resources in transitioning states. For example a volume could be left in a *deleting* or *creating* status. If left alone resources will remain in this state forever, as the service in charge of transitioning them to a rest status (*available*, *error*, *deleted*) is no longer running.

Existing reset-status operations allow operators to forcefully change the state of a resource. But these state resets are not recommended except in very specific cases and when we really know what we are doing.

Cleanup mechanisms are tasked with services recovery after an abrupt stop of the service. They are the recommended way to resolve stuck transitioning states caused by sudden service stop.

There are multiple cleanup mechanisms in Cinder, but in essence they all follow the same logic. Based on the resource type and its status the mechanism determines the best cleanup action that will transition the state to a rest state.

Some actions require a resource going through several services. In this case deciding the cleanup action may also require taking into account where the resource was being processed.

Cinder has two types of cleanup mechanisms:

- On node startup: Happen on Scheduler, Volume, and Backup services.
- Upon user request. User requested cleanups can only be triggered on Scheduler and Volume nodes.

When a node starts it will do a cleanup, but only for the resources that were left in a transitioning state when the service stopped. It will never touch resources from other services in the cluster.

Node startup cleanup is slightly different on services supporting user requested cleanups -Scheduler and Volume- than on Backup services. Backup cleanups will be covered in the services section.

For services supporting user requested cleanups we can differentiate the following tasks:

- Tracking transitioning resources: Using workers table and Cleanable Versioned Objects methods.
- Defining when a resource must be cleaned if service dies: Done in Cleanable Versioned Objects.
- Defining how a resource must be cleaned: Done in the service manager.

Note

All Volume services can accept cleanup requests, doesnt matter if they are clustered or not. This will provide a better alternative to the reset-state mechanism to handle resources stuck in a transitioning state.

Workers table

For Cinder Volume managed resources -Volumes and Snapshots- we used to establish a one-to-one relationship between a resource and the volume service managing it. A resource would belong to a node if the resources *host* field matched that of the running Cinder Volume service.

Snapshots must always be managed by the same service as the volume they originate from, so they dont have a *host* field in the database. In this case the parent volumes *host* is used to determine who owns the resource.

Cinder-Volume services can be clustered, so we no longer have a one-to-one owner relationship. On clustered services we use the *cluster_name* database field instead of the *host* to determine ownership. Now we have a one-to-many ownership relationship.

When a clustered service abruptly stops running, any of the nodes from the same cluster can cleanup the resources it was working on. There is no longer a need to restart the service to get the resources cleaned by the node startup cleanup process.

We keep track of the resources our Cinder services are working on in the *workers* table. Only resources that can be cleaned are tracked. This table stores the resource type and id, the status that should be cleared on service failure, the service that is working on it, etc. And well be updating this table as the resources move from service to service.

Worker entries are not passed as RPC parameters, so we dont need a Versioned Object class to represent them. We only have the *Worker* ORM class to represent database entries.

Following subsections will cover implementation details required to develop new cleanup resources and states. For a detailed description on the issue, ramifications, and overall solution, please refer to the Cleanup spec.

Tracking resources

Resources supporting cleanup using the workers table must inherit from the *CinderCleanableObject* Versioned Object class.

This class provides helper methods and the general interface used by Cinder for the cleanup mechanism. This interface is conceptually split in three tasks:

- Manage workers table on the database.
- Defining what states must be cleaned.
- Defining how to clean resources.

Among methods provided by the *CinderCleanableObject* class the most important ones are:

- *is_cleanable*: Checks if the resource, given its current status, is cleanable.
- *create_worker*: Create a worker entry on the API service.
- *set_worker*: Create or update worker entry.
- *unset_worker*: Remove an entry from the database. This is a real delete, not a soft-delete.
- *set_workers*: Function decorator to create or update worker entries.

Inheriting classes must define *_is_cleanable* method to define which resource states can be cleaned up.

Earlier we mentioned how cleanup depends on a resources current state. But it also depends under what version the services are running. With rolling updates we can have a service running under an earlier pinned version for compatibility purposes. A version X service could have a resource that it would consider cleanable, but its pinned to version X-1, where it was not considered cleanable. To avoid breaking things, the resource should be considered as non cleanable until the service version is unpinned.

Implementation of *_is_cleanable* method must take them both into account. The state, and the version.

Volumes implementation is a good example, as workers table was not supported before version 1.6:

```
@staticmethod
def _is_cleanable(status, obj_version):
    if obj_version and obj_version < 1.6:
        return False
    return status in ('creating', 'deleting', 'uploading', 'downloading')
```

Tracking states in the workers table starts by calling the *create_worker* method on the API node. This is best done on the different *rpcapi.py* files.

For example, a create volume operation will go from the API service to the Scheduler service, so well add it in *cinder/scheduler/rpcapi.py*:

```
def create_volume(self, ctxt, volume, snapshot_id=None, image_id=None,
                  request_spec=None, filter_properties=None,
                  backup_id=None):
    volume.create_worker()
```

But if we are deleting a volume or creating a snapshot the API will call the Volume service directly, so changes should go in *cinder/scheduler/rpcapi.py*:

```
def delete_volume(self, ctxt, volume, unmanage_only=False, cascade=False):
    volume.create_worker()
```

Once we receive the call on the other sides manager we have to call the *set_worker* method. To facilitate this task we have the *set_workers* decorator that will automatically call *set_worker* for any cleanable versioned object that is in a cleanable state.

For the create volume on the Scheduler service:

```
@objects.Volume.set_workers
@append_operation_type()
def create_volume(self, context, volume, snapshot_id=None, image_id=None,
                  request_spec=None, filter_properties=None,
                  backup_id=None):
```

And then again for the create volume on the Volume service:

```
@objects.Volume.set_workers
def create_volume(self, context, volume, request_spec=None,
                  filter_properties=None, allow_reschedule=True):
```

In these examples we are using the *set_workers* method from the *Volume* Versioned Object class. But we could be using it from any other class as it is a *staticmethod* that is not overwritten by any of the classes.

Using the *set_workers* decorator will cover most of our use cases, but sometimes we may have to call the *set_worker* method ourselves. Thats the case when transitioning from *creating* state to *downloading*. The *worker* database entry was created with the *creating* state and the working service was updated when the Volume service received the RPC call. But once we change the status to *creating* the worker and the resource status dont match, so the cleanup mechanism will ignore the resource.

To solve this we add another worker update in the *save* method from the *Volume* Versioned Object class:

```
def save(self):

    ...

    if updates.get('status') == 'downloading':
        self.set_worker()
```

Actions on resource cleanup

Weve seen how to track cleanable resources in the *workers* table. Now well cover how to define the actions used to cleanup a resource.

Services using the *workers* table inherit from the *CleanableManager* class and must implement the *_do_cleanup* method.

This method receives a versioned object to clean and indicates whether we should keep the *workers* table entry. On asynchronous cleanup tasks method must return *True* and take care of removing the worker entry on completion.

Simplified version of the cleanup of the Volume service, illustrating synchronous and asynchronous cleanups and how we can do a synchronous cleanup and take care ourselves of the *workers* entry:

```
def _do_cleanup(self, ctxt, vo_resource):
    if isinstance(vo_resource, objects.Volume):
        if vo_resource.status == 'downloading':
            self.driver.clear_download(ctxt, vo_resource)

        elif vo_resource.status == 'deleting':
            if CONF.volume_service_inithost_offload:
                self._add_to_threadpool(self.delete_volume, ctxt,
                                        vo_resource, cascade=True)
            else:
                self.delete_volume(ctxt, vo_resource, cascade=True)
            return True

    if vo_resource.status in ('creating', 'downloading'):
        vo_resource.status = 'error'
        vo_resource.save()
```

When the volume is *downloading* we dont return anything, so the caller receives *None*, which evaluates to not keep the row entry. When the status is *deleting* we call *delete_volume* synchronously or asynchronously. The *delete_volume* has the *set_workers* decorator, that calls *unset_worker* once the decorated method has successfully finished. So when calling *delete_volume* we must ask the caller of *_do_cleanup* to not try to remove the *workers* entry.

Cleaning resources

We may not have a *Worker* Versioned Object because we didnt need it, but we have a *CleanupRequest* Versioned Object to specify resources for cleanup.

Resources will be cleaned when a node starts up and on user request. In both cases well use the *CleanupRequest* that contains a filtering of what needs to be cleaned up.

The *CleanupRequest* can be considered as a filter on the *workers* table to determine what needs to be cleaned.

Managers for services using the *workers* table must support the startup cleanup mechanism. Support for this mechanism is provided via the *init_host* method in the *CleanableManager* class. So managers inheriting from *CleanableManager* must make sure they call this *init_host* method. This can be done using *CleanableManager* as the first inherited class and using *super* to call the parents *init_host* method, or by calling the class method directly: *cleanableManager.init_host(self,)*.

*CleanableManager*s *init_host* method will create a *CleanupRequest* for the current service before calling its *do_cleanup* method with it before returning. Thus cleaning up all transitioning resources from the service.

For user requested cleanups, the API generates a *CleanupRequest* object using the requests parameters and calls the schedulers *work_cleanup* RPC with it.

The Scheduler receives the *work_cleanup* RPC call and uses the *CleanupRequest* to filter services that match the request. With this list of services the Scheduler sends an individual cleanup request for each of the services. This way we can spread the cleanup work if we have multiple services to cleanup.

The Scheduler checks the service to clean to know where it must send the clean request. Scheduler service cleanup can be performed by any Scheduler, so we send it to the scheduler queue where all Schedulers are listening. In the worst case it will come back to us if there is no other Scheduler running at the time.

For the Volume service well be sending it to the cluster message queue if its a clustered service, or to a single node if its non clustered. But unlike with the Scheduler, we cant be sure that there is a service to do the cleanup, so we check if the service or cluster is up before sending the request.

After sending all the cleanup requests, the Scheduler will return a list of services that have received a cleanup request, and all the services that didnt because they were down.

Mutual exclusion

In Cinder, as many other concurrent and parallel systems, there are critical sections. Code sections that share a common resource that can only be accessed by one of them at a time.

Resources can be anything, not only Cinder resources such as Volumes and Snapshots, and they can be local or remote. Examples of resources are libraries, command line tools, storage target groups, etc.

Exclusion scopes can be per process, per node, or global.

We have four mutual exclusion mechanisms available during Cinder development:

- Database locking using resource states.
- Process locks.
- Node locks.
- Global locks.

For performance reasons we must always try to avoid using any mutual exclusion mechanism. If avoiding them is not possible, we should try to use the narrowest scope possible and reduce the critical section as much as possible. Locks by decreasing order of preference are: process locks, node locks, global locks, database locks.

Status based locking

Many Cinder operations are inherently exclusive and the Cinder core code ensures that drivers will not receive contradictory or incompatible calls. For example, you cannot clone a volume if its being created. And you shouldnt delete the source volume of an ongoing snapshot.

To prevent these from happening Cinder API services use resource status fields to check for incompatibilities preventing operations from getting through.

There are exceptions to this rule, for example the force delete operation that ignores the status of a resource.

We should also be aware that administrators can forcefully change the status of a resource and then call the API, bypassing the check that prevents multiple operations from being requested to the drivers.

Resource locking using states is expanded upon in the *Race prevention* subsection in the *Cinder-API* section.

Process locks

Cinder services are multi-threaded -not really since we use greenthreads-, so the narrowest possible scope of locking is among the threads of a single process.

Some cases where we may want to use this type of locking are when we share arrays or dictionaries between the different threads within the process, and when we use a Python or C library that doesnt properly handle concurrency and we have to be careful with how we call its methods.

To use this locking in Cinder we must use the *synchronized* method in *cinder.utils*. This method in turn uses the *synchronized* method from *oslo_concurrency.lockutils* with the *cinder-* prefix for all the locks to avoid conflict with other OpenStack services.

The only required parameter for this usage is the name of the lock. The name parameter provided for these locks must be a literal string value. There is no kind of templating support.

Example from *cinder/volume/throttling.py*:

```
@utils.synchronized('BlkioCgroup')
def _inc_device(self, srcdev, dstdev):
```

Note

When developing a driver, and considering which type of lock to use, we must remember that Cinder is a multi backend service. So the same driver can be running multiple times on different processes in the same node.

Node locks

Sometimes we want to define the whole node as the scope of the lock. Our critical section requires that only one thread in the whole node is using the resource. This inter process lock ensures that no matter how many processes and backends want to access the same resource, only one will access it at a time. All others will have to wait.

These locks are useful when:

- We want to ensure theres only one ongoing call to a command line program. Thats the case of the *cinder-rtstool* command in *cinder/volume/targets/lio.py*, and the *nvmetcli* command in *cinder/volume/targets/nvmet.py*.
- Common initialization in all processes in the node. This is the case of the backup service cleanup code. The backup service can run multiple processes simultaneously for the same backend, but only one of them can run the cleanup code on start.
- Drivers not supporting Active-Active configurations. Any operation that should only be performed by one driver at a time. For example creating target groups for a node.

This type of lock use the same method as the *Process locks*, *synchronized* method from *cinder.utils*. Here we need to pass two parameters, the name of the lock, and *external=True* to make sure that file locks are being used.

The name parameter provided for these locks must be a literal string value. There is no kind of templating support.

Example from *cinder/volume/targets/lio.py*:

```
@staticmethod
@utils.synchronized('lioadm', external=True)
def _execute(*args, **kwargs):
```

Example from *cinder/backup/manager.py*:

```
@utils.synchronized('backup-pgid-%s' % os.getpgrp(),
                    external=True, delay=0.1)
def _cleanup_incomplete_backup_operations(self, ctxt):
```

> **Warning**
>
> These are not fair locks. Order in which the lock is acquired by callers may differ from request order. Starvation is possible, so dont choose a generic lock name for all your locks and try to create a unique name for each locking domain.

Drivers that use node locks based on volumes should implement method `clean_volume_file_locks` and if they use locks based on the snapshots they should also implement `clean_snapshot_file_locks` and use method `synchronized_remove` from `cinder.utils`.

Example for a driver that used `cinder.utils.synchronized`:

```
def my_operation(self, volume):
    @utils.synchronized('my-driver-lock' + volume.id)
    def method():
```

(continues on next page)

(continued from previous page)

```
        pass

    method()

@classmethod
def clean_volume_file_locks(cls, volume_id):
    utils.synchronized_remove('my-driver-lock-' + volume_id)
```

Global locks

Global locks, also known as distributed locks in Cinder, provide mutual exclusion in the global scope of the Cinder services.

They allow you to have a lock regardless of the backend, for example to prevent deleting a volume that is being cloned, or making sure that your driver is only creating a Target group at a time, in the whole Cinder deployment, to avoid race conditions.

Global locking functionality is provided by the *synchronized* decorator from *cinder.coordination*.

Attention

Optional *blocking* and *coordinator* arguments to the *synchronized* decorator are **keyword** arguments only and cannot be passed as positional arguments.

This method is more advanced than the one used for the *Process locks* and the *Node locks*, as the name supports templates. For the template we have all the method parameters as well as *f_name* that represents that name of the method being decorated. Templates must use Pythons Format Specification Mini-Language.

Using brackets we can access the function name *{f_name}*, an attribute of a parameter *{volume.id}*, a key in a dictonary *{snapshot[name]}*, etc.

Up to date information on the method can be found in the synchronized methods documentation.

Example from the delete volume operation in *cinder/volume/manager.py*. We use the *id* attribute of the *volume* parameter, and the function name to form the lock name:

```
@coordination.synchronized('{volume.id}-{f_name}')
@objects.Volume.set_workers
def delete_volume(self, context, volume, unmanage_only=False,
                  cascade=False):
```

Example from create snapshot in *cinder/volume/drivers/nfs.py*, where we use an attribute from *self*, and a recursive reference in the *snapshot* parameter.

```
@coordination.synchronized('{self.driver_prefix}-{snapshot.volume.id}')
def create_snapshot(self, snapshot):
```

Some drivers may require multiple locks for a critical section, which could potentially create deadlocks. Like in the following example, where *PowerMax* method *move_volume_between_storage_groups* creates 2 locks:

```
@coordination.synchronized(
    "emc-sg-{source_storagegroup_name}-{serial_number}")
@coordination.synchronized(
    "emc-sg-{target_storagegroup_name}-{serial_number}")
def move_volume_between_storage_groups(
        self, serial_number, device_id, source_storagegroup_name,
        target_storagegroup_name, extra_specs, force=False,
        parent_sg=None):
```

That code can result in a deadlock if 2 opposite requests come in concurrently and their first lock acquisition interleaves.

The solution is calling the *synchronized* decorator with both lock names and let it resolve the acquire ordering issue for us. The right code would be:

```
@coordination.synchronized(
    "emc-sg-{source_storagegroup_name}-{serial_number}",
    "emc-sg-{target_storagegroup_name}-{serial_number}")
def move_volume_between_storage_groups(
        self, serial_number, device_id, source_storagegroup_name,
        target_storagegroup_name, extra_specs, force=False,
        parent_sg=None):
```

Internally Cinder uses the Tooz library to provide the distributed locking. By default, this library is configured for Active-Passive deployments, where it uses file locks equivalent to those used for *Node locks*.

To support Active-Active deployments a specific driver will need to be configured using the *backend_url* configuration option in the *coordination* section.

For a detailed description of the requirement for global locks in cinder please refer to the replacing local locks with Tooz and manager local locks specs.

Drivers that use global locks based on volumes should implement method `clean_volume_file_locks` and if they use locks based on the snapshots they should also implement `clean_snapshot_file_locks` and use method `synchronized_remove` from `cinder.coordination`.

Example for the 3PAR driver:

```
@classmethod
def clean_volume_file_locks(cls, volume_id):
    coordination.synchronized_remove('3par-' + volume_id)
```

Cinder locking

Cinder uses the different locking mechanisms covered in this section to assure mutual exclusion on some actions. Heres an *incomplete* list:

Barbican keys

- Lock scope: Global.
- Critical section: Migrate Barbican encryption keys.
- Lock name: *{id}-_migrate_encryption_key*.

- Where: *_migrate_encryption_key* method.
- File: *cinder/keymgr/migration.py*.

Backup service

- Lock scope: Node.
- Critical section: Cleaning up resources at startup.
- Lock name: *backup-pgid-{process-group-id}*.
- Where: *_cleanup_incomplete_backup_operations* method.
- File: *cinder/backup/manager.py*.

Image cache

- Lock scope: Global.
- Critical section: Create a new image cache entry.
- Lock name: *{image_id}*.
- Where: *_prepare_image_cache_entry* method.
- File: *cinder/volume/flows/manager/create_volume.py*.

Throttling:

- Lock scope: Process.
- Critical section: Set parameters of a cgroup using *cgset* CLI.
- Lock name: *BlkioCgroup*.
- Where: *_inc_device* and *_dec_device* methods.
- File: *cinder/volume/throttling.py*.

Volume deletion:

- Lock scope: Global.
- Critical section: Volume deletion operation.
- Lock name: *{volume.id}-delete_volume*.
- Where: *delete_volume* method.
- File: *cinder/volume/manager.py*.

Volume deletion request:

- Lock scope: Status based.
- Critical section: Volume delete RPC call.
- Status requirements: attach_status != attached && not migrating
- Where: *delete* method.
- File: *cinder/volume/api.py*.

Snapshot deletion:

- Lock scope: Global.

- Critical section: Snapshot deletion operation.
- Lock name: *{snapshot.id}-delete_snapshot*.
- Where: *delete_snapshot* method.
- File: *cinder/volume/manager.py*.

Volume creation:

- Lock scope: Global.
- Critical section: Protect source of volume creation from deletion. Volume or Snapshot.
- Lock name: *{snapshot-id}-delete_snapshot* or *{volume-id}-delete_volume}*.
- Where: Inside *create_volume* method as context manager for calling *_fun_flow*.
- File: *cinder/volume/manager.py*.

Attach volume:

- Lock scope: Global.
- Critical section: Updating DB to show volume is attached.
- Lock name: *{volume_id}*.
- Where: *attach_volume* method.
- File: *cinder/volume/manager.py*.

Detach volume:

- Lock scope: Global.
- Critical section: Updating DB to show volume is detached.
- Lock name: *{volume_id}-detach_volume*.
- Where: *detach_volume* method.
- File: *cinder/volume/manager.py*.

Volume upload image:

- Lock scope: Status based.
- Critical section: *copy_volume_to_image* RPC call.
- Status requirements: status = available or (force && status = in-use)
- Where: *copy_volume_to_image* method.
- File: *cinder/volume/api.py*.

Volume extend:

- Lock scope: Status based.
- Critical section: *extend_volume* RPC call.
- Status requirements: status in (in-use, available)
- Where: *_extend* method.
- File: *cinder/volume/api.py*.

Volume migration:

- Lock scope: Status based.
- Critical section: *migrate_volume* RPC call.
- Status requirements: status in (in-use, available) && not migrating
- Where: *migrate_volume* method.
- File: *cinder/volume/api.py*.

Volume retype:

- Lock scope: Status based.
- Critical section: *retype* RPC call.
- Status requirements: status in (in-use, available) && not migrating
- Where: *retype* method.
- File: *cinder/volume/api.py*.

Driver locking

There is no general rule on where drivers should use locks. Each driver has its own requirements and limitations determined by the storage backend and the tools and mechanisms used to manage it.

Even if they are all different, commonalities may exist between drivers. Providing a list of where some drivers are using locks, even if the list is incomplete, may prove useful to other developers.

To contain the length of this document and keep it readable, the list with the drivers_locking_examples has its own document.

Cinder-API

The API service is the public face of Cinder. Its REST API makes it possible for anyone to manage and consume block storage resources. So requests from clients can, and usually do, come from multiple sources.

Each Cinder API service by default will run multiple workers. Each worker is run in a separate subprocess and will run a predefined maximum number of green threads.

The number of API workers is defined by the *osapi_volume_workers* configuration option. Defaults to the number of CPUs available.

Number of green threads per worker is defined by the *wsgi_default_pool_size* configuration option. Defaults to 100 green threads.

The service takes care of validating request parameters. Any detected error is reported immediately to the user.

Once the request has been validated, the database is changed to reflect the request. This can result in adding a new entry to the database and/or modifying an existing entry.

For create volume and create snapshot operations the API service will create a new database entry for the new resource. And the new information for the resource will be returned to the caller right after the service passes the request to the next Cinder service via RPC.

Operations like retype and delete will change the database entry referenced by the request, before making the RPC call to the next Cinder service.

Create backup and restore backup are two of the operations that will create a new entry in the database, and modify an existing one.

These database changes are very relevant to the high availability operation. Cinder core code uses resource states extensively to control exclusive access to resources.

Race prevention

The API service checks that resources referenced in requests are in a valid state. Unlike allowed resource states, valid states are those that allow an operation to proceed.

Validation usually requires checking multiple conditions. Careless coding leaves Cinder open to race conditions. Patterns in the form of DB data read, data check, and database entry modification, must be avoided in the Cinder API service.

Cinder has implemented a custom mechanism, called conditional updates, to prevent race conditions. Leverages the SQLAlchemy ORM library to abstract the equivalent `UPDATE ... FROM ... WHERE;` SQL query.

Complete reference information on the conditional updates mechanism is available on the *API Races - Conditional Updates* development document.

For a detailed description on the issue, ramifications, and solution, please refer to the API Race removal spec.

Cinder-Volume

The most common deployment option for Cinder-Volume is as Active-Passive. This requires a common storage backend, the same Cinder backend configuration in all nodes, having the *backend_host* set on the backend sections, and using a high-availability cluster resource manager like Pacemaker.

Attention

Having the same *host* value configured on more than one Cinder node is highly discouraged. Using *backend_host* in the backend section is the recommended way to set Active-Passive configurations. Setting the same *host* field will make Scheduler and Backup services report using the same database entry in the *services* table. This may create a good number of issues: We cannot tell when the service in a node is down, backups services will break other running services operation on start, etc.

For Active-Active configurations we need to include the Volume services that will be managing the same backends on the cluster. To include a node in a cluster, we need to define its name in the *[DEFAULT]* section using the *cluster* configuration option, and start or restart the service.

Note

We can create a cluster with a single volume node. Having a single node cluster allows us to later on add new nodes to the cluster without restarting the existing node.

Warning

The name of the cluster must be unique and cannot match any of the *host* or *backend_host* values. Non unique values will generate duplicated names for message queues.

When a Volume service is configured to be part of a cluster, and the service is restarted, the manager detects the change in configuration and moves existing resources to the cluster.

Resources are added to the cluster in the *_include_resources_in_cluster* method setting the *cluster_name* field in the database. Volumes, groups, consistency groups, and image cache elements are added to the cluster.

Clustered Volume services are different than normal services. To determine if a backend is up, it is no longer enough checking *service.is_up*, as that will only give us the status of a specific service. In a clustered deployment there could be other services that are able to service the same backend. Thats why well have to check if a service is clustered using *cinder.is_clustered* and if it is, check the clusters *is_up* property instead: *service.cluster.is_up*.

In the code, to detect if a cluster is up, the *is_up* property from the *Cluster* Versioned Object uses the *last_heartbeat* field from the same object. The *last_heartbeat* is a *column property* from the SQLAlchemy ORM model resulting from getting the latest *updated_at* field from all the services in the same cluster.

RPC calls

When we discussed the *Job distribution* we mentioned message queues having multiple listeners and how they were used to distribute jobs in a round robin fashion to multiple nodes.

For clustered Volume services we have the same queues used for broadcasting and to address a specific node, but we also have queues to broadcast to the cluster and to send jobs to the cluster.

Volume services will be listening in all these queues and they can receive request from any of them. Which theyll have to do to process RPC calls addressed to the cluster or to themselves.

Deciding the target message queue for request to the Volume service is done in the *volume/rpcapi.py* file.

We use method *_get_cctxt*, from the *VolumeAPI* class, to prepare the client context to make RPC calls. This method accepts a *host* parameter to indicate where we want to make the RPC. This *host* parameter refers to both hosts and clusters, and is used to determine the server and the topic.

When calling the *_get_cctx* method, we would need to pass the resources *host* field if its not clustered, and *cluster_name* if it is. To facilitate this, clustered resources implement the *service_topic_queue* property that automatically gives you the right value to pass to *_get_cctx*.

An example for the create volume:

```
def create_volume(self, ctxt, volume, request_spec, filter_properties,
                  allow_reschedule=True):
    cctxt = self._get_cctxt(volume.service_topic_queue)
    cctxt.cast(ctxt, 'create_volume',
               request_spec=request_spec,
               filter_properties=filter_properties,
               allow_reschedule=allow_reschedule,
               volume=volume)
```

As we know, snapshots dont have a *host* or *cluseter_name* fields, but we can still use the *service_topic_queue* property from the *Snapshot* Versioned Object to get the right value. The *Snapshot* internally checks these values from the *Volume* Versioned Object linked to that *Snapshot* to determine the right value. Heres an example for deleting a snapshot:

```
def delete_snapshot(self, ctxt, snapshot, unmanage_only=False):
    cctxt = self._get_cctxt(snapshot.service_topic_queue)
    cctxt.cast(ctxt, 'delete_snapshot', snapshot=snapshot,
               unmanage_only=unmanage_only)
```

Replication

Replication v2.1 failover is requested on a per node basis, so when a failover request is received by the API it is then redirected to a specific Volume service. Only one of the services that form the cluster for the storage backend will receive the request, and the others will be oblivious to this change and will continue using the same replication site they had been using before.

To support the replication feature on clustered Volume services, drivers need to implement the Active-Active replication spec. In this spec the *failover_host* method is split in two, *failover* and *failover_completed*.

On a backend supporting replication on Active-Active deployments, *failover_host* would end up being a call to *failover* followed by a call to *failover_completed*.

Code extract from the RBD driver:

```
def failover_host(self, context, volumes, secondary_id=None, groups=None):
    active_backend_id, volume_update_list, group_update_list = (
        self.failover(context, volumes, secondary_id, groups))
    self.failover_completed(context, secondary_id)
    return active_backend_id, volume_update_list, group_update_list
```

Enabling Active-Active on Drivers

Supporting Active-Active configurations is driver dependent, so they have to opt in. By default drivers are not expected to support Active-Active configurations and will fail on startup if we try to deploy them as such.

Drivers can indicate they support Active-Active setting the class attribute *SUPPORTS_ACTIVE_ACTIVE* to *True*. If a single driver supports multiple storage solutions, it can leave the class attribute as it is, and set it as an overriding instance attribute on *__init__*.

There is no well defined procedure required to allow driver maintainers to set *SUPPORTS_ACTIVE_ACTIVE* to *True*. Though there is an ongoing effort to write a spec on testing Active-Active.

So for now, we could say that its self-certification. Vendors must do their own testing until they are satisfied with their testing.

Real testing of Active-Active deployments requires multiple Cinder Volume nodes on different hosts, as well as a properly configured Tooz DLM.

Driver maintainers can use Devstack to catch the rough edges on their initial testing. Running 2 Cinder Volume services on an All-In-One DevStack installation makes it easy to deploy and debug.

Running 2 Cinder Volume services on the same node simulating different nodes can be easily done:

- Creating a new directory for local locks: Since we are running both services on the same node, a file lock could make us believe that the code would work on different nodes. Having a different lock directory, default is */opt/stack/data/cinder*, will prevent this.
- Creating a layover cinder configuration file: Cinder supports having different configurations files where each new files overrides the common parts of the old ones. We can use the same base cinder configuration provided by DevStack and write a different file with a *[DEFAULT]* section that configures *host* (to anything different than the one used in the first service), and *lock_path* (to the new directory we created). For example we could create */etc/cinder/cinder2.conf*.
- Create a new service unit: This service unit should be identical to the existing *devstack@c-vol* except replace the *ExecStart* that should have the postfix *config-file /etc/cinder/cinder2.conf*.

Once we have tested it in DevStack way we should deploy Cinder in a new Node, and continue with the testings.

It is not necessary to do the DevStack step first, we can jump to having Cinder in multiple nodes right from the start.

Whatever way we decide to test this, well have to change *cinder.conf* and add the *cluster* configuration option and restart the Cinder service. We also need to modify the driver under test to include the *SUPPORTS_ACTIVE_ACTIVE = True* class attribute.

Cinder-Scheduler

Unlike the Volume service, the Cinder Scheduler has supported Active-Active deployments for a long time.

Unfortunately, current support is not perfect, scheduling on Active-Active deployments has some issues.

The root cause of these issues is that the scheduler services dont have a reliable single source of truth for the information they rely on to make the scheduling.

Volume nodes periodically send a broadcast with the backend stats to all the schedulers. The stats include total storage space, free space, configured maximum over provisioning, etc. All the backends information is stored in memory at the Schedulers, and used to decide where to create new volumes, migrate them on a retype, and so on.

For additional information on the stats, please refer to the *volume stats* section of the Contributor/Developer docs.

Trying to keep updated stats, schedulers reduce available free space on backends in their internal dictionary. These updates are not shared between schedulers, so there is not a single source of truth, and other schedulers dont operate with the same information.

Until the next stat reports is sent, schedulers will not get in sync. This may create unexpected behavior on scheduling.

There are ongoing efforts to fix this problem. Multiple solutions are being discussed: using the database as a single source of truth, or using an external placement service.

When we added Active-Active support to the Cinder Volume service we had to update the scheduler to understand it. This mostly entailed 3 things:

- Setting the *cluster_name* field on Versioned Objects once a backend has been chosen.
- Grouping stats for all clustered hosts. We dont want to have individual entries for the stats of each host that manages a cluster, as there should be only one up to date value. We stopped using the *host*

field as the id for each host, and created a new property called *backend_id* that takes into account if the service is clustered and returns the host or the cluster as the identifier.

- Prevent race conditions on stats reports. Due to the concurrency on the multiple Volume services in a cluster, and the threading in the Schedulers, we could receive stat reports out of order (more up to date stats last). To prevent this we started time stamping the stats on the Volume services. Using the timestamps schedulers can discard older stats.

Heartbeats

Like any other non API service, schedulers also send heartbeats using the database.

The difference is that, unlike other services, the purpose of these heartbeats is merely informative. Admins can easily know whether schedulers are running or not with a Cinder command.

Using the same *host* configuration in all nodes defeats the whole purpose of reporting heartbeats in the schedulers, as they will all report on the same database entry.

Cinder-Backups

Originally, the Backup service was not only limited to Active-Passive deployments, but it was also tightly coupled to the Volume service. This coupling meant that the Backup service could only backup volumes created by the Volume service running on the same node.

In the Mitaka cycle, the Scalable Backup Service spec was implemented. This added support for Active-Active deployments to the backup service.

The Active-Active implementation for the backup service is different than the one we explained for the Volume Service. The reason lays not only on the fact that the Backup service supported it first, but also on it not supporting multiple backends, and not using the Scheduler for any operations.

Scheduling

For backups, its the API the one selecting the host that will do the backup, using methods *_get_available_backup_service_host*, *_is_backup_service_enabled*, and *_get_any_available_backup_service*.

These methods use the Backup services heartbeats to determine which hosts are up to handle requests.

Cleaning

Cleanup on Backup services is only performed on start up.

To know what resources each node is working on, they set the *host* field in the backup Versioned Object when they receive the RPC call. That way they can select them for cleanup on start.

The method in charge of doing the cleanup for the backups is called *_cleanup_incomplete_backup_operations*.

Unlike with the Volume service we cannot have a backup node clean up after another nodes.

Guru Meditation Reports

Cinder contains a mechanism whereby developers and system administrators can generate a report about the state of a running Cinder executable. This report is called a *Guru Meditation Report* (*GMR* for short).

Generating a GMR

A *GMR* can be generated by sending the *USR2* signal to any Cinder process with support (see below). The *GMR* will then output to standard error for that particular process.

For example, suppose that `cinder-api` has process id `8675`, and was run with `2>/var/log/cinder/cinder-api-err.log`. Then, `kill -USR2 8675` will trigger the Guru Meditation report to be printed to `/var/log/cinder/cinder-api-err.log`.

There is other way to trigger a generation of report, user should add a configuration in Cinders conf file:

```
[oslo_reports]
file_event_handler=['The path to a file to watch for changes to trigger '
                    'the reports, instead of signals. Setting this option '
                    'disables the signal trigger for the reports.']
file_event_handler_interval=['How many seconds to wait between polls when '
                             'file_event_handler is set, default value '
                             'is 1']
```

a *GMR* can be generated by touching the file which was specified in file_event_handler. The *GMR* will then output to standard error for that particular process.

For example, suppose that `cinder-api` was run with `2>/var/log/cinder/cinder-api-err.log`, and the file path is `/tmp/guru_report`. Then, `touch /tmp/guru_report` will trigger the Guru Meditation report to be printed to `/var/log/cinder/cinder-api-err.log`.

Structure of a GMR

The *GMR* is designed to be extensible; any particular executable may add its own sections. However, the base *GMR* consists of several sections:

Package
Shows information about the package to which this process belongs, including version information

Threads
Shows stack traces and thread ids for each of the threads within this process

Green Threads
Shows stack traces for each of the green threads within this process (green threads dont have thread ids)

Configuration
Lists all the configuration options currently accessible via the CONF object for the current process

Adding Support for GMRs to New Executables

Adding support for a *GMR* to a given executable is fairly easy.

First import the module (currently residing in oslo-incubator), as well as the Cinder version module:

```
from oslo_reports import guru_meditation_report as gmr
from cinder import version
```

Then, register any additional sections (optional):

```
TextGuruMeditation.register_section('Some Special Section',
                                    some_section_generator)
```

Finally (under main), before running the main loop of the executable (usually `service.server(server)` or something similar), register the *GMR* hook:

```
TextGuruMeditation.setup_autorun(version)
```

Extending the GMR

As mentioned above, additional sections can be added to the GMR for a particular executable. For more information, see the inline documentation about oslo.reports: oslo.reports

Replication

For backend devices that offer replication features, Cinder provides a common mechanism for exposing that functionality on a per volume basis while still trying to allow flexibility for the varying implementation and requirements of all the different backend devices.

There are 2 sides to Cinders replication feature, the core mechanism and the driver specific functionality, and in this document well only be covering the driver side of things aimed at helping vendors implement this functionality in their drivers in a way consistent with all other drivers.

Although well be focusing on the driver implementation there will also be some mentions on deployment configurations to provide a clear picture to developers and help them avoid implementing custom solutions to solve things that were meant to be done via the cloud configuration.

Overview

As a general rule replication is enabled and configured via the cinder.conf file under the drivers section, and volume replication is requested through the use of volume types.

NOTE: Current replication implementation is v2.1 and its meant to solve a very specific use case, the smoking hole scenario. Its critical that you read the Use Cases section of the spec here: https://specs.openstack.org/openstack/cinder-specs/specs/mitaka/cheesecake.html

From a users perspective volumes will be created using specific volume types, even if it is the default volume type, and they will either be replicated or not, which will be reflected on the `replication_status` field of the volume. So in order to know if a snapshot is replicated well have to check its volume.

After the loss of the primary storage site all operations on the resources will fail and VMs will no longer have access to the data. It is then when the Cloud Administrator will issue the `failover-host` command to make the cinder-volume service perform the failover.

After the failover is completed, the Cinder volume service will start using the failed-over secondary storage site for all operations and the user will once again be able to perform actions on all resources that were replicated, while all other resources will be in error status since they are no longer available.

Storage Device configuration

Most storage devices will require configuration changes to enable the replication functionality, and this configuration process is vendor and storage device specific so it is not contemplated by the Cinder core replication functionality.

It is up to the vendors whether they want to handle this device configuration in the Cinder driver or as a manual process, but the most common approach is to avoid including this configuration logic into Cinder and having the Cloud Administrators do a manual process following a specific guide to enable replication on the storage device before configuring the cinder volume service.

Service configuration

The way to enable and configure replication is common to all drivers and it is done via the `replication_device` configuration option that goes in the drivers specific section in the `cinder.conf` configuration file.

`replication_device` is a multi dictionary option, that should be specified for each replication target device the admin wants to configure.

While it is true that all drivers use the same `replication_device` configuration option this doesnt mean that they will all have the same data, as there is only one standardized and **REQUIRED** key in the configuration entry, all others are vendor specific:

- backend_id:<vendor-identifier-for-rep-target>

Values of `backend_id` keys are used to uniquely identify within the driver each of the secondary sites, although they can be reused on different driver sections.

These unique identifiers will be used by the failover mechanism as well as in the driver initialization process, and the only requirement is that is must never have the value default.

An example driver configuration for a device with multiple replication targets is show below:

```
.....
[driver-biz]
volume_driver=xxxx
volume_backend_name=biz

[driver-baz]
volume_driver=xxxx
volume_backend_name=baz

[driver-foo]
volume_driver=xxxx
volume_backend_name=foo
replication_device = backend_id:vendor-id-1,unique_key:val....
replication_device = backend_id:vendor-id-2,unique_key:val....
```

In this example the result of calling `self.configuration.safe_get('replication_device')` within the driver is the following list:

```
[{backend_id: vendor-id-1, unique_key: val1},
 {backend_id: vendor-id-2, unique_key: val2}]
```

It is expected that if a driver is configured with multiple replication targets, that replicated volumes are actually replicated on **all targets**.

Besides specific replication device keys defined in the `replication_device`, a driver may also have additional normal configuration options in the driver section related with the replication to allow Cloud Administrators to configure things like timeouts.

Capabilities reporting

There are 2 new replication stats/capability keys that drivers supporting replication v2.1 should be reporting: `replication_enabled` and `replication_targets`:

```
stats["replication_enabled"] = True|False
stats["replication_targets"] = [<backend-id_1, <backend-id_2>...]
```

If a driver is behaving correctly we can expect the `replication_targets` field to be populated whenever `replication_enabled` is set to `True`, and it is expected to either be set to `[]` or be missing altogether when `replication_enabled` is set to `False`.

The purpose of the `replication_enabled` field is to be used by the scheduler in volume types for creation and migrations.

As for the `replication_targets` field it is only provided for informational purposes so it can be retrieved through the `get_capabilities` using the admin REST API, but it will not be used for validation at the API layer. That way Cloud Administrators will be able to know available secondary sites where they can failover.

Volume Types / Extra Specs

The way to control the creation of volumes on a cloud with backends that have replication enabled is, like with many other features, through the use of volume types.

We wont go into the details of volume type creation, but suffice to say that you will most likely want to use volume types to discriminate between replicated and non replicated volumes and be explicit about it so that non replicated volumes wont end up in a replicated backend.

Since the driver is reporting the `replication_enabled` key, we just need to require it for replication volume types adding `replication_enabled='<is> True'` and also specifying it for all non replicated volume types `replication_enabled='<is> False'`.

Its up to the driver to parse the volume type info on create and set things up as requested. While the scoping key can be anything, its strongly recommended that all backends utilize the same key (replication) for consistency and to make things easier for the Cloud Administrator.

Additional replication parameters can be supplied to the driver using vendor specific properties through the volume types extra-specs so they can be used by the driver at volume creation time, or retype.

It is up to the driver to parse the volume type info on create and retype to set things up as requested. A good pattern to get a custom parameter from a given volume instance is this:

```
extra_specs = getattr(volume.volume_type, 'extra_specs', {})
custom_param = extra_specs.get('custom_param', 'default_value')
```

It may seem convoluted, but we must be careful when retrieving the `extra_specs` from the `volume_type` field as it could be `None`.

Vendors should try to avoid obfuscating their custom properties and expose them using the `_init_vendor_properties` method so they can be checked by the Cloud Administrator using the `get_capabilities` REST API.

NOTE: For storage devices doing per backend/pool replication the use of volume types is also recommended.

Volume creation

Drivers are expected to honor the replication parameters set in the volume type during creation, retyping, or migration.

When implementing the replication feature there are some driver methods that will most likely need modifications -if they are implemented in the driver (since some are optional)- to make sure that the backend is replicating volumes that need to be replicated and not replicating those that dont need to be:

- create_volume
- create_volume_from_snapshot
- create_cloned_volume
- retype
- clone_image
- migrate_volume

In these methods the driver will have to check the volume type to see if the volumes need to be replicated, we could use the same pattern described in the *Volume Types / Extra Specs* section:

```
def _is_replicated(self, volume):
    specs = getattr(volume.volume_type, 'extra_specs', {})
    return specs.get('replication_enabled') == '<is> True'
```

But it is **not** the recommended mechanism, and the `is_replicated` method available in volumes and volume types versioned objects instances should be used instead.

Drivers are expected to keep the `replication_status` field up to date and in sync with reality, usually as specified in the volume type. To do so in above mentioned methods implementation they should use the update model mechanism provided for each one of those methods. One must be careful since the update mechanism may be different from one method to another.

What this means is that most of these methods should be returning a `replication_status` key with the value set to `enabled` in the model update dictionary if the volume type is enabling replication. There is no need to return the key with the value of `disabled` if it is not enabled since that is the default value.

In the case of the `create_volume`, and `retype` method there is no need to return the `replication_status` in the model update since it has already been set by the scheduler on creation using the extra spec from the volume type. And on `migrate_volume` there is no need either since there is no change to the `replication_status`.

NOTE: For storage devices doing per backend/pool replication it is not necessary to check the volume type for the `replication_enabled` key since all created volumes will be replicated, but they are expected to return the `replication_status` in all those methods, including the `create_volume` method since the driver may receive a volume creation request without the replication enabled extra spec and therefore the driver will not have set the right `replication_status` and the driver needs to correct this.

Besides the `replication_status` field that drivers need to update there are other fields in the database related to the replication mechanism that the drivers can use:

- replication_extended_status
- replication_driver_data

These fields are string type fields with a maximum size of 255 characters and they are available for drivers to use internally as they see fit for their normal replication operation. So they can be assigned in the model update and later on used by the driver, for example during the failover.

To avoid using magic strings drivers must use values defined by the `ReplicationStatus` class in `cinder/objects/fields.py` file and these are:

- `ERROR`: When setting the replication failed on creation, retype, or migrate. This should be accompanied by the volume status `error`.
- `ENABLED`: When the volume is being replicated.
- `DISABLED`: When the volume is not being replicated.
- `FAILED_OVER`: After a volume has been successfully failed over.
- `FAILOVER_ERROR`: When there was an error during the failover of this volume.
- `NOT_CAPABLE`: When we failed-over but the volume was not replicated.

The first 3 statuses revolve around the volume creation and the last 3 around the failover mechanism.

The only status that should not be used for the volumes `replication_status` is the `FAILING_OVER` status.

Whenever we are referring to values of the `replication_status` in this document we will be referring to the `ReplicationStatus` attributes and not a literal string, so `ERROR` means `cinder.objects.field.ReplicationStatus.ERROR` and not the string ERROR.

Failover

This is the mechanism used to instruct the cinder volume service to fail over to a secondary/target device.

Keep in mind the use case is that the primary backend has died a horrible death and is no longer valid, so any volumes that were on the primary and were not being replicated will no longer be available.

The method definition required from the driver to implement the failback mechanism is as follows:

```
def failover_host(self, context, volumes, secondary_id=None):
```

There are several things that are expected of this method:

- Promotion of a secondary storage device to primary
- Generating the model updates
- Changing internally to access the secondary storage device for all future requests.

If no secondary storage device is provided to the driver via the `backend_id` argument (it is equal to `None`), then it is up to the driver to choose which storage device to failover to. In this regard it is important that the driver takes into consideration that it could be failing over from a secondary (there was a prior failover request), so it should discard current target from the selection.

If the `secondary_id` is not a valid one the driver is expected to raise `InvalidReplicationTarget`, for any other non recoverable errors during a failover the driver should raise `UnableToFailOver` or any child of `VolumeDriverException` class and revert to a state where the previous backend is in use.

The failover method in the driver will receive a list of replicated volumes that need to be failed over. Replicated volumes passed to the driver may have diverse `replication_status` values, but they will always be one of: `ENABLED`, `FAILED_OVER`, or `FAILOVER_ERROR`.

The driver must return a 2-tuple with the new storage device target id as the first element and a list of dictionaries with the model updates required for the volumes so that the driver can perform future actions on those volumes now that they need to be accessed on a different location.

Its not a requirement for the driver to return model updates for all the volumes, or for any for that matter as it can return `None` or an empty list if theres no update necessary. But if elements are returned in the model update list then it is a requirement that each of the dictionaries contains 2 key-value pairs, `volume_id` and `updates` like this:

```
[{
    'volume_id': volumes[0].id,
    'updates': {
        'provider_id': new_provider_id1,
        ...
    },
    'volume_id': volumes[1].id,
    'updates': {
        'provider_id': new_provider_id2,
        'replication_status': fields.ReplicationStatus.FAILOVER_ERROR,
        ...
    },
}]
```

In these updates there is no need to set the `replication_status` to `FAILED_OVER` if the failover was successful, as this will be performed by the manager by default, but it wont create additional DB queries if it is returned. It is however necessary to set it to `FAILOVER_ERROR` for those volumes that had errors during the failover.

Drivers dont have to worry about snapshots or non replicated volumes, since the manager will take care of those in the following manner:

- All non replicated volumes will have their current `status` field saved in the `previous_status` field, the `status` field changed to `error`, and their `replication_status` set to `NOT_CAPABLE`.
- All snapshots from non replicated volumes will have their statuses changed to `error`.
- All replicated volumes that failed on the failover will get their `status` changed to `error`, their current `status` preserved in `previous_status`, and their `replication_status` set to `FAILOVER_ERROR` .
- All snapshots from volumes that had errors during the failover will have their statuses set to `error`.

Any model update request from the driver that changes the `status` field will trigger a change in the `previous_status` field to preserve the current status.

Once the failover is completed the driver should be pointing to the secondary and should be able to create and destroy volumes and snapshots as usual, and it is left to the Cloud Administrators discretion whether resource modifying operations are allowed or not.

Failback

Drivers are not required to support failback, but they are required to raise a `InvalidReplicationTarget` exception if the failback is requested but not supported.

The way to request the failback is quite simple, the driver will receive the argument `secondary_id` with the value of `default`. That is why it was forbidden to use the `default` on the target configuration in the cinder configuration file.

Expected driver behavior is the same as the one explained in the *Failover* section:

- Promotion of the original primary to primary
- Generating the model updates
- Changing internally to access the original primary storage device for all future requests.

If the failback of any of the volumes fail the driver must return `replication_status` set to `ERROR` in the volume updates for those volumes. If they succeed it is not necessary to change the `replication_status` since the default behavior will be to set them to `ENABLED`, but it wont create additional DB queries if it is set.

The manager will update resources in a slightly different way than in the failover case:

- All non replicated volumes will not have any model modifications.
- All snapshots from non replicated volumes will not have any model modifications.
- All replicated volumes that failed on the failback will get their `status` changed to `error`, have their current `status` preserved in the `previous_status` field, and their `replication_status` set to `FAILOVER_ERROR`.
- All snapshots from volumes that had errors during the failover will have their statuses set to `error`.

We can avoid using the default magic string by using the `FAILBACK_SENTINEL` class attribute from the `VolumeManager` class.

Initialization

It stands to reason that a failed over Cinder volume service may be restarted, so there needs to be a way for a driver to know on start which storage device should be used to access the resources.

So, to let drivers know which storage device they should use the manager passes drivers the `active_backend_id` argument to their `__init__` method during the initialization phase of the driver. Default value is `None` when the default (primary) storage device should be used.

Drivers should store this value if they will need it, as the base driver is not storing it, for example to determine the current storage device when a failover is requested and we are already in a failover state, as mentioned above.

Freeze / Thaw

In many cases, after a failover has been completed well want to allow changes to the data in the volumes as well as some operations like attach and detach while other operations that modify the number of existing resources, like delete or create, are not allowed.

And that is where the freezing mechanism comes in; freezing a backend puts the control plane of the specific Cinder volume service into a read only state, or at least most of it, while allowing the data plane to proceed as usual.

While this will mostly be handled by the Cinder core code, drivers are informed when the freezing mechanism is enabled or disabled via these 2 calls:

```
freeze_backend(self, context)
thaw_backend(self, context)
```

In most cases the driver may not need to do anything, and then it doesnt need to define any of these methods as long as its a child class of the `BaseVD` class that already implements them as noops.

Raising a *VolumeDriverException* exception in any of these methods will result in a 500 status code response being returned to the caller and the manager will not log the exception, so its up to the driver to log the error if it is appropriate.

If the driver wants to give a more meaningful error response, then it can raise other exceptions that have different status codes.

When creating the *freeze_backend* and *thaw_backend* driver methods we must remember that this is a Cloud Administrator operation, so we can return errors that reveal internals of the cloud, for example the type of storage device, and we must use the appropriate internationalization translation methods when raising exceptions; for *VolumeDriverException* no translation is necessary since the manager doesnt log it or return to the user in any way, but any other exception should use the `_()` translation method since it will be returned to the REST API caller.

For example, if a storage device doesnt support the thaw operation when failed over, then it should raise an *Invalid* exception:

```
def thaw_backend(self, context):
    if self.failed_over:
        msg = _('Thaw is not supported by driver XYZ.')
        raise exception.Invalid(msg)
```

User Messages

General information

User messages are a way to inform users about the state of asynchronous operations. One example would be notifying the user of why a volume provisioning request failed. End users can request these messages via the Volume v3 REST API under the `/messages` resource. The REST API allows only GET and DELETE verbs for this resource.

Internally, you use the `cinder.message.api` to work with messages. In order to prevent leakage of sensitive information or breaking the volume service abstraction layer, free-form messages are *not* allowed. Instead, all messages must be defined using a combination of pre-defined fields in the `cinder.message.message_field` module.

The message ultimately displayed to end users is combined from an `Action` field and a `Detail` field.

- The `Action` field describes what was taking place when the message was created, for example, `Action.COPY_IMAGE_TO_VOLUME`.
- The `Detail` field is used to provide more information, for example, `Detail.NOT_ENOUGH_SPACE_FOR_IMAGE` or `Detail.QUOTA_EXCEED`.

Example

Example message generation:

```
from cinder import context
from cinder.message import api as message_api
from cinder.message import message_field

```

(continues on next page)

(continued from previous page)

```
self.message_api = message_api.API()

context = context.RequestContext()
volume_id = 'f292cc0c-54a7-4b3b-8174-d2ff82d87008'

self.message_api.create(
    context,
    message_field.Action.UNMANAGE_VOLUME,
    resource_uuid=volume_id,
    detail=message_field.Detail.UNMANAGE_ENC_NOT_SUPPORTED)
```

Will produce roughly the following:

```
GET /v3/6c430ede-9476-4128-8838-8d3929ced223/messages
{
  "messages": [
    {
     "id": "5429fffa-5c76-4d68-a671-37a8e24f37cf",
     "event_id": "VOLUME_VOLUME_006_008",
     "user_message": "unmanage volume: Unmanaging encrypted volumes is not␣
↪supported.",
     "message_level": "ERROR",
     "resource_type": "VOLUME",
     "resource_uuid": "f292cc0c-54a7-4b3b-8174-d2ff82d87008",
     "created_at": 2018-08-27T09:49:58-05:00,
     "guaranteed_until": 2018-09-27T09:49:58-05:00,
     "request_id": "req-936666d2-4c8f-4e41-9ac9-237b43f8b848",
    }
  ]
}
```

Adding user messages

If you are creating a message in the code but find that the predefined fields are insufficient, just add what you need to `cinder.message.message_field`. The key thing to keep in mind is that all defined fields should be appropriate for any API user to see and not contain any sensitive information. A good rule-of-thumb is to be very general in error messages unless the issue is due to a bad user action, then be specific.

As a convenience to developers, the `Detail` class contains a `EXCEPTION_DETAIL_MAPPINGS` dict. This maps `Detail` fields to particular Cinder exceptions, and allows you to create messages in a context where youve caught an Exception that could be any of several possibilities. Instead of having to sort through them where youve caught the exception, you can call `message_api.create` and pass it both the exception and a general detail field like `Detail.SOMETHING_BAD_HAPPENED` (thats not a real field, but you get the idea). If the passed exception is in the mapping, the resulting message will have the mapped `Detail` field instead of the generic one.

Usage patterns

These are taken from the Cinder code. The exact code may have changed by the time you read this, but the general idea should hold.

No exception in context

From cinder/compute/nova.py:

```
def extend_volume(self, context, server_ids, volume_id):
    api_version = '2.51'
    events = [self._get_volume_extended_event(server_id, volume_id)
              for server_id in server_ids]
    result = self._send_events(context, events, api_version=api_version)
    if not result:
        self.message_api.create(
            context,
            message_field.Action.EXTEND_VOLUME,
            resource_uuid=volume_id,
            detail=message_field.Detail.NOTIFY_COMPUTE_SERVICE_FAILED)
    return result
```

- You must always pass the context object and an action.
- Were working with an existing volume, so pass its ID as the `resource_uuid`.
- You need to fill in some detail, or else the code will supply an `UNKNOWN_ERROR`, which isnt very helpful.

Cinder exception in context

From cinder/scheduler/manager.py:

```
except exception.NoValidBackend as ex:
    QUOTAS.rollback(context, reservations,
                    project_id=volume.project_id)
    _extend_volume_set_error(self, context, ex, request_spec)
    self.message_api.create(
        context,
        message_field.Action.EXTEND_VOLUME,
        resource_uuid=volume.id,
        exception=ex)
```

- You must always pass the context object and an action.
- Since we have it available, pass the volume ID as the resource_uuid.
- Its a Cinder exception. Check to see if its in the mapping.
  - If its there, we can pass it, and the detail will be supplied by the code.
  - It its not, consider adding it and mapping it to an existing `Detail` field. If theres no current `Detail` field for that exception, go ahead and add that, too.

- On the other hand, maybe its in the mapping, but you have more information in this code context than is available in the mapped `Detail` field. In that case, you may want to use a different `Detail` field (creating it if necessary).
- Remember, if you pass *both* a mapped exception *and* a detail, the passed detail will be ignored and the mapped `Detail` field will be used instead.

General Exception in context

Not passing the Exception to message_api.create()

From cinder/volume/manager.py:

```
try:
    self.driver.extend_volume(volume, new_size)
except exception.TargetUpdateFailed:
    # We just want to log this but continue on with quota commit
    LOG.warning('Volume extended but failed to update target.')
except Exception:
    LOG.exception("Extend volume failed.",
                  resource=volume)
    self.message_api.create(
        context,
        message_field.Action.EXTEND_VOLUME,
        resource_uuid=volume.id,
        detail=message_field.Detail.DRIVER_FAILED_EXTEND)
```

- Pass the context object and an action; pass a `resource_uuid` since we have it.
- Were not passing the exception, so the `detail` we pass is guaranteed to be used.

Passing the Exception to message_api.create()

From cinder/volume/manager.py:

```
try:
    if volume_metadata.get('readonly') == 'True' and mode != 'ro':
        raise exception.InvalidVolumeAttachMode(mode=mode,
                                                volume_id=volume.id)
    utils.require_driver_initialized(self.driver)

    LOG.info('Attaching volume %(volume_id)s to instance '
             '%(instance)s at mountpoint %(mount)s on host '
             '%(host)s.',
             {'volume_id': volume_id, 'instance': instance_uuid,
              'mount': mountpoint, 'host': host_name_sanitized},
             resource=volume)
    self.driver.attach_volume(context,
                              volume,
                              instance_uuid,
                              host_name_sanitized,
                              mountpoint)
```

(continues on next page)

(continued from previous page)

```
except Exception as excep:
    with excutils.save_and_reraise_exception():
        self.message_api.create(
            context,
            message_field.Action.ATTACH_VOLUME,
            resource_uuid=volume_id,
            exception=excep)
        attachment.attach_status = (
            fields.VolumeAttachStatus.ERROR_ATTACHING)
        attachment.save()
```

- Pass the context object and an action; pass a resource_uuid since we have it.
- Were passing an exception, which could be a Cinder `InvalidVolumeAttachMode`, which is in the mapping. In that case, the mapped `Detail` will be used; otherwise, the code will supply a `Detail.UNKNOWN_ERROR`.

  This is appropriate if we really have no idea what happened. If its possible to provide more information, we can pass a different, generic `Detail` field (creating it if necessary). The passed detail would be used for any exception thats *not* in the mapping. If its a mapped exception, then the mapped `Detail` field will be used.

Module documentation

The Message API Module

Handles all requests related to user facing messages.

class API

API for handling user messages.

Cinder Messages describe the outcome of a user action using predefined fields that are members of objects defined in the cinder.message.message_field package. They are intended to be exposed to end users. Their primary purpose is to provide end users with a means of discovering what went wrong when an asynchronous action in the Volume REST API (for which theyve already received a 2xx response) fails.

Messages contain an expires_at field based on the creation time plus the value of the message_ttl configuration option. They are periodically reaped by a task of the SchedulerManager class whose periodicity is given by the message_reap_interval configuration option.

cleanup_expired_messages(*context*)

create(*context*, *action*, *resource_type='VOLUME'*, *resource_uuid=None*, *exception=None*, *detail=None*, *level='ERROR'*)

Create a message record with the specified information.

Parameters

- **context** current context object
- **action** a message_field.Action field describing what was taking place when this message was created

- **resource_type** a message_field.Resource field describing the resource this message applies to. Default is message_field.Resource.VOLUME
- **resource_uuid** the resource ID if this message applies to an existing resource. Default is None
- **exception** if an exception has occurred, you can pass it in and it will be translated into an appropriate message detail ID (possibly message_field.Detail.UNKNOWN_ERROR). The message in the exception itself is ignored in order not to expose sensitive information to end users. Default is None
- **detail** a message_field.Detail field describing the event the message is about. Default is None, in which case message_field.Detail.UNKNOWN_ERROR will be used for the message unless an exception in the message_field.EXCEPTION_DETAIL_MAPPINGS is passed; in that case the message_field.Detail field thats mapped to the exception is used.
- **level** a string describing the severity of the message. Suggested values are INFO, ERROR, WARNING. Default is ERROR.

`create_from_request_context`(*context*, *exception=None*, *detail=None*, *level='ERROR'*)

Create a message record with the specified information.

Parameters

- **context** current context object which we must have populated with the message_action, message_resource_type and message_resource_id fields
- **exception** if an exception has occurred, you can pass it in and it will be translated into an appropriate message detail ID (possibly message_field.Detail.UNKNOWN_ERROR). The message in the exception itself is ignored in order not to expose sensitive information to end users. Default is None
- **detail** a message_field.Detail field describing the event the message is about. Default is None, in which case message_field.Detail.UNKNOWN_ERROR will be used for the message unless an exception in the message_field.EXCEPTION_DETAIL_MAPPINGS is passed; in that case the message_field.Detail field thats mapped to the exception is used.
- **level** a string describing the severity of the message. Suggested values are INFO, ERROR, WARNING. Default is ERROR.

`delete`(*context*, *id*)

Delete message with the specified id.

`get`(*context*, *id*)

Return message with the specified id.

`get_all`(*context*, *filters=None*, *marker=None*, *limit=None*, *offset=None*, *sort_keys=None*, *sort_dirs=None*)

Return all messages for the given context.

The Message Field Module

Message Resource, Action, Detail and user visible message.

Use Resource, Action and Details combination to indicate the Event in the format of:

EVENT: VOLUME_RESOURCE_ACTION_DETAIL

Also, use exception-to-detail mapping to decrease the workload of classifying event in cinders task code.

The Defined Messages Module

This module is DEPRECATED and is currently only used by `cinder.api.v3.messages` to handle pre-Pike message database objects. (Editorial comment:: With the default `message_ttl` of 2592000 seconds (30 days), its probably safe to remove this module during the Train development cycle.)

Event ID and user visible message mapping.

Event IDs are used to look up the message to be displayed for an API Message object. All defined messages should be appropriate for any API user to see and not contain any sensitive information. A good rule-of-thumb is to be very general in error messages unless the issue is due to a bad user action, then be specific.

class EventIds

Bases: `object`

ATTACH_READONLY_VOLUME = 'VOLUME_000003'

IMAGE_FROM_VOLUME_OVER_QUOTA = 'VOLUME_000004'

UNABLE_TO_ALLOCATE = 'VOLUME_000002'

UNKNOWN_ERROR = 'VOLUME_000001'

UNMANAGE_ENCRYPTED_VOLUME_UNSUPPORTED = 'VOLUME_000005'

get_message_text(*event_id*)

Migration

Introduction to volume migration

Cinder provides the volume migration support within the same deployment, which means the node of cinder volume service, c-vol node where the source volume is located, is able to access the c-vol node where the destination volume is located, and both of them share the same Cinder API service, scheduler service, message queue service, etc.

As a general rule migration is possible for volumes in available or in-use status, for the driver which has implemented volume migration. So far, we are confident that migration will succeed for available volumes, whose drivers implement the migration routines. However, the migration of in-use volumes is driver dependent. It depends on different drivers involved in the operation. It may fail depending on the source or destination driver of the volume.

For example, for older releases (before Ussuri), the migration of in-use volume from RBD to LVM will succeed, but from LVM to RBD, it will fail. Currently, migration of in-use volumes will not succeed for any backend when they are attached to an instance in any of these states: SHUTOFF, SUSPENDED, or SOFT-DELETED.

There are two major scenarios, which volume migration supports in Cinder:

Scenario 1: Migration between two back-ends with the same volume type, regardless if they are located on the same c-vol node or not.

Scenario 2: Migration between two back-ends with different volume types, regardless if the back-ends are located on the same c-vol node or not.

Note

Retyping an unencrypted volume to the same size encrypted volume will most likely fail. Even though the volume is the same size as the source volume, the encrypted volume needs to store additional encryption information overhead. This results in the new volume not being large enough to hold all data. Please do not try this in older releases.

How to do volume migration via CLI

Scenario 1 of volume migration is done via the following command from the CLI:

```
cinder migrate [--force-host-copy [<True|False>]]
               [--lock-volume [<True|False>]]
               <volume> <host>

Mandatory arguments:
  <volume>                ID of volume to migrate.
  <host>                  Destination host. The format of host is
                          host@backend#POOL, while 'host' is the host
                          name of the volume node, 'backend' is the back-end
                          name and 'POOL' is a logical concept to describe
                          a set of storage resource, residing in the
                          back-end. If the back-end does not have specified
                          pools, 'POOL' needs to be set with the same name
                          as 'backend'.

Optional arguments:
  --force-host-copy [<True|False>]
                          Enables or disables generic host-based force-
                          migration, which bypasses the driver optimization.
                          Default=False.
  --lock-volume [<True|False>]
                          Enables or disables the termination of volume
                          migration caused by other commands. This option
                          applies to the available volume. True means it locks
                          the volume state and does not allow the migration to
                          be aborted. The volume status will be in maintenance
                          during the migration. False means it allows the volume
                          migration to be aborted. The volume status is still in
                          the original status. Default=False.
```

Important note: Currently, error handling for failed migration operations is under development in Cinder. If we would like the volume migration to finish without any interruption, please set lock-volume to True. If it is set to False, we cannot predict what will happen, if other actions like attach, detach, extend, etc, are issued on the volume during the migration. It all depends on which stage the volume migration has reached and when the request of another action comes.

Scenario 2 of volume migration can be done via the following command from the CLI:

```
cinder retype --migration-policy on-demand
              <volume> <volume-type>
Mandatory arguments:
  <volume>                Name or ID of volume for which to modify type.
  <volume-type>           New volume type.
```

Source volume type and destination volume type must be different and they must refer to different back-ends.

Configurations

To set up an environment to try the volume migration, we need to configure at least two different back-ends on the same node of cinder volume service, c-vol node or two back-ends on two different volume nodes of cinder volume service, c-vol nodes. Which command to use, cinder migrate or cinder retype, depends on which type of volume we would like to test.

Scenario 1 for migration

To configure the environment for Scenario 1 migration, e.g. a volume is migrated from back-end <driver-backend> on Node 1 to back-end <driver-backend> on Node 2, cinder.conf needs to contains the following entries for the same back-end on both of source and the destination nodes:

For Node 1:
[<driver-backend>] volume_driver=xxxx volume_backend_name=<driver-backend>

For Node 2:
[<driver-backend>] volume_driver=xxxx volume_backend_name=<driver-backend>

If a volume with a predefined volume type is going to migrate, the back-end drivers from Node 1 and Node 2 should have the same value for volume_backend_name, which means <driver-backend> should be the same for Node 1 and Node 2. The volume type can be created with the extra specs {volume_backend_name: driver-biz}.

If we are going to migrate a volume with a volume type of none, it is not necessary to set the same value to volume_backend_name for both Node 1 and Node 2.

Scenario 2 for migration

To configure the environment for Scenario 2 migration: For example, a volume is migrated from driver-biz back-end on Node 1 to driver-net back-end on Node 2, cinder.conf needs to contains the following entries:

For Node 1:
[driver-biz] volume_driver=xxxx volume_backend_name=driver-biz

For Node 2:
[driver-net] volume_driver=xxxx volume_backend_name=driver-net

For example, a volume is migrated from driver-biz back-end on Node 1 to driver-biz back-net on the same node, cinder.conf needs to contains the following entries:

[driver-biz] volume_driver=xxxx volume_backend_name=driver-biz

[driver-net] volume_driver=xxxx volume_backend_name=driver-net

Two volume types need to be created. One is with the extra specs: {volume_backend_name: driver-biz}. The other is with the extra specs: {volume_backend_name: driver-net}.

What can be tracked during volume migration

The volume migration is an administrator only action and it may take a relatively long time to finish. The property migration status will indicate the stage of the migration process for the volume. The administrator can check the migration status via the cinder list or cinder show <volume-id> command. The cinder list command presents a list of all the volumes with some properties displayed, including the migration status, only to the administrator. However, the migration status is not included if cinder list is issued by an ordinary user. The cinder show <volume-id> will present all the detailed information of a specific volume, including the migration status, only to the administrator.

If the migration status of a volume shows starting, migrating or completing, it means the volume is in the process of a migration. If the migration status is success, it means the migration has finished and the previous migration of this volume succeeded. If the migration status is error, it means the migration has finished and the previous migration of this volume failed.

How to implement volume migration for a back-end driver

There are two kinds of implementations for the volume migration currently in Cinder.

The first is the generic host-assisted migration, which consists of two different transfer modes, block-based and file-based. This implementation is based on the volume attachment to the node of cinder volume service, c-vol node. Any back-end driver supporting iSCSI will be able to support the generic host-assisted migration for sure. The back-end driver without iSCSI supported needs to be tested to decide if it supports this kind of migration. The block-based transfer mode is done by dd command, applying to drivers like LVM, Storwize, etc, and the file-based transfer mode is done by file copy, typically applying to the RBD driver.

The second is the driver specific migration. Since some storage back-ends have their special commands to copy the volume, Cinder also provides a way for them to implement in terms of their own internal commands to migrate.

If the volume is migrated between two nodes configured with the same storage back-end, the migration will be optimized by calling the method migrate_volume in the driver, if the driver provides an implementation for it to migrate the volume within the same back-end, and will fallback to the generic host-assisted migration provided in the manager, if no such implementation is found or this implementation is not applicable for this migration.

If your storage driver in Cinder provides iSCSI support, it should naturally work under the generic host-assisted migration, when force-host-copy is set to True from the API request. Normally you do not need to change any code, unless you need to transfer the volume from your driver via a different way from the block-based transfer or the file-based transfer.

If your driver uses a network connection to communicate the block data itself, you can use file I/O to participate in migration. Please take the RBD driver as a reference for this implementation.

If you would like to implement a driver specific volume migration for your driver, the API method associated with the driver specific migration is the following admin only method:

```
migrate_volume(self, ctxt, volume, host)
```

If your driver is taken as the destination back-end for a generic host-assisted migration and your driver needs to update the volume model after a successful migration, you need to implement the following method for your driver:

```
update_migrated_volume(self, ctxt, volume, new_volume, original_volume_status)
```

Required methods

There is one mandatory method that needs to be implemented for the driver to implement the driver specific volume migration.

migrate_volume

Used to migrate the volume directly if source and destination are managed by same storage.

There is one optional method that could be implemented for the driver to implement the generic host-assisted migration.

update_migrated_volume

Used to return the key-value pairs to update the volume model after a successful migration. The key-value pairs returned are supposed to be the final values your driver would like to be in the volume model, if a migration is completed.

This method can be used in a generally wide range, but the most common use case covered in this method is to rename the back-end name to the original volume id in your driver to make sure that the back-end still keeps the same id or name as it is before the volume migration. For this use case, there are two important fields: _name_id and provider_location.

The field _name_id is used to map the cinder volume id and the back-end id or name. The default value is None, which means the cinder volume id is the same to the back-end id or name. If they are different, _name_id is used to saved the back-end id or name.

The field provider_location is used to save the export information, created by the volume attach. This field is optional, since some drivers support the export creation and some do not. It is the driver maintainers responsibility to decide what this field needs to be.

If the back-end id or name is renamed successfully, this method can return {_name_id: None, provider_location: None}. It is the choice for your driver to implement this method and decide what use cases should be covered.

Running Cinder API under Apache

Files

Copy the file etc/cinder/api-httpd.conf to the appropriate location for your Apache server, most likely:

`/etc/httpd/conf.d/cinder_wsgi.conf`

Update this file to match your system configuration (for example, some distributions put httpd logs in the apache2 directory and some in the httpd directory). Create the directory /var/www/cgi-bin/cinder/. You can either hard or soft link the file cinder/wsgi/wsgi.py to be osapi_volume under the /var/www/cgi-bin/cinder/ directory. For a distribution appropriate place, it should probably be copied to:

`/usr/share/openstack/cinder/httpd/cinder.py`

Cinders primary configuration file (etc/cinder.conf) and the PasteDeploy configuration file (etc/cinder-paste.ini) must be readable to httpd in one of the default locations described in Configuring Cinder.

Access Control

If you are running with Linux kernel security module enabled (for example SELinux or AppArmor), make sure that the configuration file has the appropriate context to access the linked file.

Upgrades

Starting from Mitaka release Cinder gained the ability to be upgraded without introducing downtime of control plane services. Operator can simply upgrade Cinder services instances one-by-one. To achieve that, developers need to make sure that any introduced change doesnt break older services running in the same Cinder deployment.

In general there is a requirement that release N will keep backward compatibility with release N-1 and in a deployment Ns and N-1s services can safely coexist. This means that when performing a live upgrade you cannot skip any release (e.g. you cannot upgrade N to N+2 without upgrading it to N+1 first). Further in the document N will denote the current release, N-1 a previous one, N+1 the next one, etc.

Having in mind that we only support compatibility with N-1, most of the compatibility code written in N needs to exist just for one release and can be removed in the beginning of N+1. A good practice here is to mark them with `TODO` or `FIXME` comments to make them easy to find in the future.

Please note that proper upgrades solution should support both release-to-release upgrades as well as upgrades of deployments following the Cinder master more closely. We cannot just merge patches implementing compatibility at the end of the release - we should keep things compatible through the whole release.

To achieve compatibility, discipline is required from the developers. There are several planes on which incompatibility may occur:

- **REST API changes** - these are prohibited by definition and this document will not describe the subject. For further information one may use API Working Group guidelines for reference.
- **Database schema migrations** - e.g. if N-1 was relying on some column in the DB being present, Ns migrations cannot remove it. N+1s however can (assuming N has no notion of the column).
- **Database data migrations** - if a migration requires big amount of data to be transferred between columns or tables or converted, it will most likely lock the tables. This may cause services to be unresponsive, causing the downtime.
- **RPC API changes** - adding or removing RPC method parameter, or the method itself, may lead to incompatibilities.
- **RPC payload changes** - adding, renaming or removing a field from the dict passed over RPC may lead to incompatibilities.

Next sections of this document will focus on explaining last four points and provide means to tackle required changes in these matters while maintaining backward compatibility.

Database schema and data migrations

In general incompatible database schema migrations can be tracked to ALTER and DROP SQL commands instruction issued either against a column or table. This is why a unit test that blocks such migrations was introduced. We should try to keep our DB modifications additive. Moreover we should aim not to introduce migrations that cause the database tables to lock for a long period. Long lock on whole table can block other queries and may make real requests to fail.

Adding a column

This is the simplest case - we dont have any requirements when adding a new column apart from the fact that it should be added as the last one in the table. If thats covered, the DB engine will make sure the migration wont be disruptive.

Dropping a column not referenced in SQLAlchemy code

When we want to remove a column that wasnt present in any SQLAlchemy model or it was in the model, but model was not referenced anywhere in our code (this basically means that N-1 wasnt depending on the presence of that column in the DB), then the situation is simple. We should be able to safely drop the column in N release.

Removal of unnecessary column

When we want to remove a used column without migrating any data out of it (for example because whats kept in the column is obsolete), then we just need to remove it from the SQLAlchemy model and API in N release. In N+1 or as a post-upgrade migration in N we can merge a migration issuing DROP for this column (we cannot do that earlier because N-1 will depend on the presence of that column).

ALTER on a column

A rule of thumb to judge which ALTER or DROP migrations should be allowed is to look in the MySQL documentation. If operation has yes in all 4 columns besides Copies Table?, then it *probably* can be allowed. If operation doesnt allow concurrent DML it means that table row modifications or additions will be blocked during the migration. This sometimes isnt a problem - for example its not the end of the world if a service wont be able to report its status one or two times (and `services` table is normally small). Please note that even if this does apply to rename a column operation, we cannot simply do such ALTER, as N-1 will depend on the older name.

If an operation on column or table cannot be allowed, then it is required to create a new column with desired properties and start moving the data (in a live manner). In worst case old column can be removed in N+2. Whole procedure is described in more details below.

In aforementioned case we need to make more complicated steps stretching through 3 releases - always keeping the backwards compatibility. In short when we want to start to move data inside the DB, then in N we should:

- Add a new column for the data.
- Write data in both places (N-1 needs to read it).
- Read data from the old place (N-1 writes there).
- Prepare online data migration cinder-manage command to be run before upgrading to N+1 (because N+1 will read from new place, so we need to make sure all the records have new place populated).

In N+1 we should:

- Write data to both places (N reads from old one).
- Read data from the new place (N saves there).

In N+2

- Remove old place from SQLAlchemy.
- Read and write only to the new place.

- Remove the column as the post-upgrade migration (or as first migration in N+3).

Please note that this is the most complicated case. If data in the column cannot actually change (for example `host` in `services` table), in N we can read from new place and fallback to the old place if data is missing. This way we can skip one release from the process.

Of course real-world examples may be different. E.g. sometimes it may be required to write some more compatibility code in the oslo.versionedobjects layer to compensate for different versions of objects passed over RPC. This is explained more in *RPC payload changes (oslo.versionedobjects)* section.

More details about that can be found in the online-schema-upgrades spec.

RPC API changes

It can obviously break service communication if RPC interface changes. In particular this applies to changes of the RPC method definitions. To avoid that we assume Ns RPC API compatibility with N-1 version (both ways - `rpcapi` module should be able to downgrade the message if needed and `manager` module should be able to tolerate receiving messages in older version.

Below is an example RPC compatibility shim from Mitakas `cinder.volume.manager`. This code allows us to tolerate older versions of the messages:

```
def create_volume(self, context, volume_id, request_spec=None,
                  filter_properties=None, allow_reschedule=True,
                  volume=None):

    """Creates the volume."""
    # FIXME(thangp): Remove this in v2.0 of RPC API.
    if volume is None:
        # For older clients, mimic the old behavior and look up the volume
        # by its volume_id.
        volume = objects.Volume.get_by_id(context, volume_id)
```

And heres a contrary shim in cinder.volume.rpcapi (RPC client) that downgrades the message to make sure it will be understood by older instances of the service:

```
def create_volume(self, ctxt, volume, host, request_spec,
                  filter_properties, allow_reschedule=True):
    request_spec_p = jsonutils.to_primitive(request_spec)
    msg_args = {'volume_id': volume.id, 'request_spec': request_spec_p,
                'filter_properties': filter_properties,
                'allow_reschedule': allow_reschedule}
    if self.client.can_send_version('1.32'):
        version = '1.32'
        msg_args['volume'] = volume
    else:
        version = '1.24'

    new_host = utils.extract_host(host)
    cctxt = self.client.prepare(server=new_host, version=version)
    request_spec_p = jsonutils.to_primitive(request_spec)
    cctxt.cast(ctxt, 'create_volume', **msg_args)
```

As can be seen theres this magic `self.client.can_send_version()` method which detects if were running in a version-heterogeneous environment and need to downgrade the message. Detection is based on dynamic RPC version pinning. In general all the services (managers) report supported RPC API version. RPC API client gets all the versions from the DB, chooses the lowest one and starts to downgrade messages to it.

To limit impact on the DB the pinned version of certain RPC API is cached. After all the services in the deployment are updated, operator should restart all the services or send them a SIGHUP signal to force reload of version pins.

As we need to support only N RPC API in N+1 release, we should be able to drop all the compatibility shims in N+1. To be technically correct when doing so we should also bump the major RPC API version. We do not need to do that in every release (it may happen that through the release nothing will change in RPC API or cost of technical debt of compatibility code is lower than the cost of complicated procedure of increasing major version of RPC APIs).

The process of increasing the major version is explained in details in Novas documentation. Please note that in case of Cinder were accessing the DB from all of the services, so we should follow the more complicated Mixed version environments process for every of our services.

In case of removing whole RPC method we need to leave it there in Ns manager and can remove it in N+1 (because N-1 will be talking with N). When adding a new one we need to make sure that when the RPC client is pinned to a too low version any attempt to send new message should fail (because client will not know if manager receiving the message will understand it) or ensure the manager will get updated before clients by stating the recommended order of upgrades for that release.

RPC payload changes (oslo.versionedobjects)

oslo.versionedobjects is a library that helps us to maintain compatibility of the payload sent over RPC. As during the process of upgrades it is possible that a newer version of the service will send an object to an older one, it may happen that newer object is incompatible with older service.

Version of an object should be bumped every time we make a change that will result in an incompatible change of the serialized object. Tests will inform you when you need to bump the version of a versioned object, but rule of thumb is that we should never bump the version when we modify/adding/removing a method to the versioned object (unlike Nova we dont use remotable methods), and should always bump it when we modify the fields dictionary.

There are exceptions to this rule, for example when we change a `fields.StringField` by a custom `fields.BaseEnumField`. The reason why a version bump is not required in this case its because the actual data doesnt change, we are just removing magic string by an enumerate, but the strings used are exactly the same.

As mentioned before, you dont have to know all the rules, as we have a test that calculates the hash of all objects taking all these rules into consideration and will tell you exactly when you need to bump the version of a versioned object.

You can run this test with `tox -epy35 -- --path cinder/tests/unit/objects/test_objects.py`. But you may need to run it multiple times until it passes since it may not detect all required bumps at once.

Then youll see which versioned object requires a bump and you need to bump that version and update the object_data dictionary in the test file to reflect the new version as well as the new hash.

There is a very common false positive on the version bump test, and that is when we have modified a versioned object that is being used by other objects using the `fields.ObjectField` class. Due to the backporting mechanism implemented in Cinder we dont require bumping the version for these cases and well just need to update the hash used in the test.

For example if we were to add a new field to the Volume object and then run the test we may think that we need to bump Volume, Snapshot, Backup, RequestSpec, and VolumeAttachment objects, but we really only need to bump the version of the Volume object and update the hash for all the other objects.

Imagine that we (finally!) decide that `request_spec` sent in `create_volume` RPC cast is duplicating data and we want to start to remove redundant occurrences. When running in version-mixed environment older services will still expect this redundant data. We need a way to somehow downgrade the `request_spec` before sending it over RPC. And this is were o.vo come in handy. o.vo provide us the infrastructure to keep the changes in object versioned and to be able to downgrade them to a particular version.

Lets take a step back - similarly to the RPC API situation we need a way to tell if we need to send a backward-compatible version of the message. In this case we need to know to what version to downgrade the object. Were using a similar solution to the one used for RPC API for that. A problem here is that we need a single identifier (that we will be reported to `services` DB table) to denote whole set of versions of all the objects. To do that weve introduced a concept of `CinderObjectVersionHistory` object, where we keep sets of individual object versions aggregated into a single version string. When making an incompatible change in a single object you need to bump its version (we have a unit test enforcing that) *and* add a new version to `cinder.objects.base.CinderObjectVersionsHistory` (theres a unit test as well). Example code doing that is below:

```
OBJ_VERSIONS.add('1.1', {'Service': '1.2', 'ServiceList': '1.1'})
```

This line adds a new 1.1 aggregated object version that is different from 1.0 by two objects - `Service` in 1.2 and `ServiceList` in 1.1. This means that the commit which added this line bumped versions of these two objects.

Now if we know that a service were talking to is running 1.1 aggregated version - we need to downgrade `Service` and `ServiceList` to 1.2 and 1.1 respectively before sending. Please note that of course other objects are included in the 1.1 aggregated version, but you just need to specify what changed (all the other versions of individual objects will be taken from the last version - 1.0 in this case).

Getting back to `request_spec` example. So lets assume we want to remove `volume_properties` from there (most of data in there is already somewhere else inside the `request_spec` object). Weve made a change in the object fields, weve bumped its version (from 1.0 to 1.1), weve updated hash in the `cinder.tests.unit.test_objects` to synchronize it with the current state of the object, making the unit test pass and weve added a new aggregated object history version in `cinder.objects.base`.

What else is required? We need to provide code that actually downgrades RequestSpec object from 1.1 to 1.0 - to be used when sending the object to older services. This is done by implementing `obj_make_compatible` method in the object:

```
from oslo_utils import versionutils

def obj_make_compatible(self, primitive, target_version):
    super(RequestSpec, self).obj_make_compatible(primitive, target_version)
    target_version = versionutils.convert_version_to_tuple(target_version)
    if target_version < (1, 1) and not 'volume_properties' in primitive:
        volume_properties = {}
        # TODO: Aggregate all the required information from primitive.
        primitive['volume_properties'] = volume_properties
```

Please note that primitive is a dictionary representation of the object and not an object itself. This is because o.vo are of course sent over RPC as dicts.

With these pieces in place Cinder will take care of sending `request_spec` with `volume_properties` when running in mixed environment and without when all services are upgraded and will understand `request_spec` without `volume_properties` element.

Note that o.vo layer is able to recursively downgrade all of its fields, so when *request_spec* will be used as a field in other object, it will be correctly downgraded.

A more common case where we need backporting code is when we add new fields. In such case the backporting consist on removing the newly added fields. For example if we add 3 new fields to the Group object in version 1.1, then we need to remove them if backporting to earlier versions:

```
from oslo_utils import versionutils

def obj_make_compatible(self, primitive, target_version):
    super(Group, self).obj_make_compatible(primitive, target_version)
    target_version = versionutils.convert_version_to_tuple(target_version)
    if target_version < (1, 1):
        for key in ('group_snapshot_id', 'source_group_id',
                    'group_snapshots'):
            primitive.pop(key, None)
```

As time goes on we will be adding more and more new fields to our objects, so we may end up with a long series of if and for statements like in the Volume object:

```
from oslo_utils import versionutils

def obj_make_compatible(self, primitive, target_version):
    super(Volume, self).obj_make_compatible(primitive, target_version)
    target_version = versionutils.convert_version_to_tuple(target_version)
    if target_version < (1, 4):
        for key in ('cluster', 'cluster_name'):
            primitive.pop(key, None)
    if target_version < (1, 5):
        for key in ('group', 'group_id'):
            primitive.pop(key, None)
```

So a different pattern would be preferable as it will make the backporting easier for future additions:

```
from oslo_utils import versionutils

def obj_make_compatible(self, primitive, target_version):
    added_fields = (((1, 4), ('cluster', 'cluster_name')),
                    ((1, 5), ('group', 'group_id')))
    super(Volume, self).obj_make_compatible(primitive, target_version)
    target_version = versionutils.convert_version_to_tuple(target_version)
    for version, remove_fields in added_fields:
        if target_version < version:
            for obj_field in remove_fields:
                primitive.pop(obj_field, None)
```

Upgrade Checks

Starting with the Stein release of OpenStack, Cinder has added support for Upgrade Checks. Upgrade checks provide a release-specific readiness check before restarting services with new code. Details on how to run an Upgrade Check can be seen in the *CLI interface for :doc:'cinder status commands </cli/cinder-status>* page.

Upgrade checks are intended to help identify changes between releases that may impact the deployment environment. As a result, developers should take time to consider if the operator would benefit from having an Upgrade Check added along with changes they are proposing. The following are a few examples of changes that would require an Upgrade Check:

- Changes to Configuration Options * Removal * Change in Behavior
- Driver Removal
- Changes to Configuration File Locations
- Deprecations

To add an Upgrade Check edit the *cinder/cmd/status.py* file. Add a new function that contains the check you wish to implement. Functions need to return either a *uc.Result* where the result can be one of:

- SUCCESS
- FAILURE, <Failure explanation>
- WARNING, <Warning explanation>

Your new function should then be added to the *_upgrade_checks* tuple. For your check give the name of the Upgrade Check to be displayed to end users upon success or failure as well as the name of the function used to implement your check. Upgrade Checks should be submitted with Unit Tests.

The *doc/source/cli/cinder-status.rst* documentation should be updated to indicate the release for which your Upgrade Check was released and to explain the reason or limitations of your check, if appropriate. A release note should also be created with an explanation of the Upgrade Check in the *upgrade* section.

It is preferable to have Upgrade Checks submitted as part of the patch that is making the change in question. The checks, however, can be submitted as a separate patch and are appropriate for backport if they are being created after a release has been cut.

For additional details on Upgrade Checks please see Novas Upgrade Checks Documentation .

What can be checked?

The cinder-status CLI tool is assumed to be run from a place where it can read cinder.conf for the services, and that it can access the Cinder database to query information.

It cannot be assumed to have network access to a storage backend a backend may only be accessible from the Cinder Volume service and not reachable directly from where this tool is run.

Generic Volume Groups

Introduction to generic volume groups

Generic volume group support was added in cinder in the Newton release. There is support for creating group types and group specs, creating groups of volumes, and creating snapshots of groups. Detailed information on how to create a group type, a group, and a group snapshot can be found in *block storage admin guide*.

How is generic volume groups different from consistency groups in cinder? The consistency group feature was introduced in cinder in Juno and are supported by a few drivers. Currently consistency groups in cinder only support consistent group snapshot. It cannot be extended easily to serve other purposes. A tenant may want to put volumes used in the same application together in a group so that it is easier to manage them together, and this group of volumes may or may not support consistent group snapshot. Generic volume group is introduced to solve this problem. By decoupling the tight relationship between the group construct and the consistency concept, generic volume groups can be extended to support other features in the future.

Action items for drivers supporting consistency groups

Drivers currently supporting consistency groups are in the following:

- Juno: EMC VNX
- Kilo: EMC VMAX, IBM (GPFS, Storwize, SVC, and XIV), ProphetStor, Pure
- Liberty: Dell Storage Center, EMC XtremIO, HPE 3Par and LeftHand
- Mitaka: EMC ScaleIO, NetApp Data ONTAP, SolidFire
- Newton: CoprHD, FalconStor, Huawei

Since the addition of generic volume groups, there is plan to migrate consistency groups to generic volume groups. A migration command and changes in CG APIs to support migrating CGs to groups are developed and merged in Ocata [1][2]. In order to support rolling upgrade, it will take a couple of releases before consistency groups can be deprecated.

For drivers planning to add consistency groups support, the new generic volume group driver interfaces should be implemented instead of the CG interfaces.

For drivers already supporting consistency groups, the new generic volume group driver interfaces should be implemented to include the CG support.

For drivers wanting generic volume groups but not consistent group snapshot support, no code changes are necessary. By default, every cinder volume driver already supports generic volume groups since Newton because the support was added to the common code. Testing should be done for every driver to make sure this feature works properly.

Drivers already supporting CG are expected to add CG support to generic volume groups by Pike-1. This is a deadline discussed and agreed upon at the Ocata summit in Barcelona.

Group Type and Group Specs / Volume Types and Extra Specs

The driver interfaces for consistency groups and generic volume groups are very similar. One new concept introduced for generic volume groups is the group type. Group type is used to categorize a group just like a volume type is used to describe a volume. Similar to extra specs for a volume type, group specs are also introduced to be associated with a group type. Group types allow a user to create different types of groups.

A group can support multiple volume types and volume types are required as input parameters when creating a group. In addition to volume types, a group type is also required when creating a group.

Group types and volume types are created by the Cloud Administrator. A tenant uses the group types and volume types to create groups and volumes.

A driver can support both consistent group snapshot and a group of snapshots that do not maintain the write order consistency by using different group types. In other words, a group supporting consistent group snapshot is a special type of generic volume group.

For a group to support consistent group snapshot, the group specs in the corresponding group type should have the following entry:

```
{'consistent_group_snapshot_enabled': <is> True}
```

Similarly, for a volume to be in a group that supports consistent group snapshots, the volume type extra specs would also have the following entry:

```
{'consistent_group_snapshot_enabled': <is> True}
```

By requiring the above entry to be in both group specs and volume type extra specs, we can make sure the scheduler will choose a backend that supports the group type and volume types for a group. It is up to the driver to parse the group type info when creating a group, parse the volume type info when creating a volume, and set things up as requested.

Capabilities reporting

The following entry is expected to be added to the stats/capabilities update for drivers supporting consistent group snapshot:

```
stats["consistent_group_snapshot_enabled"] = True
```

Driver methods

The following driver methods should to be implemented for the driver to support consistent group snapshot:

- create_group(context, group)
- delete_group(context, group, volumes)
- update_group(context, group, add_volumes=None, remove_volumes=None)
- **create_group_from_src(context, group, volumes,**
  group_snapshot=None, snapshots=None, source_group=None, source_vols=None)
- create_group_snapshot(context, group_snapshot, snapshots)
- delete_group_snapshot(context, group_snapshot, snapshots)

Here is an example that add CG capability to generic volume groups [3]. Details of driver interfaces are as follows.

create_group

This method creates a group. It has context and group object as input parameters. A group object has volume_types and group_type_id that can be used by the driver.

create_group returns model_update. model_update will be in this format: {status: xxx, }.

If the status in model_update is error, the manager will throw an exception and it will be caught in the try-except block in the manager. If the driver throws an exception, the manager will also catch it in the try-except block. The group status in the db will be changed to error.

For a successful operation, the driver can either build the model_update and return it or return None. The group status will be set to available.

delete_group

This method deletes a group. It has context, group object, and a list of volume objects as input parameters. It returns model_update and volumes_model_update.

volumes_model_update is a list of volume dictionaries. It has to be built by the driver. An entry will be in this format: {id: xxx, status: xxx, }. model_update will be in this format: {status: xxx, }. The driver should populate volumes_model_update and model_update and return them.

The manager will check volumes_model_update and update db accordingly for each volume. If the driver successfully deleted some volumes but failed to delete others, it should set statuses of the volumes accordingly so that the manager can update db correctly.

If the status in any entry of volumes_model_update is error_deleting or error, the status in model_update will be set to the same if it is not already error_deleting or error.

If the status in model_update is error_deleting or error, the manager will raise an exception and the status of the group will be set to error in the db. If volumes_model_update is not returned by the driver, the manager will set the status of every volume in the group to error in the except block.

If the driver raises an exception during the operation, it will be caught by the try-except block in the manager. The statuses of the group and all volumes in it will be set to error.

For a successful operation, the driver can either build the model_update and volumes_model_update and return them or return None, None. The statuses of the group and all volumes will be set to deleted after the manager deletes them from db.

update_group

This method adds existing volumes to a group or removes volumes from a group. It has context, group object, a list of volume objects to be added to the group, and a list of a volume objects to be removed from the group. It returns model_update, add_volumes_update, and remove_volumes_update.

model_update is a dictionary that the driver wants the manager to update upon a successful return. If None is returned, the manager will set the status to available.

add_volumes_update and remove_volumes_update are lists of dictionaries that the driver wants the manager to update upon a successful return. Note that each entry requires a {id: xxx} so that the correct volume entry can be updated. If None is returned, the volume will remain its original status.

If the driver throws an exception, the status of the group as well as those of the volumes to be added/removed will be set to error.

create_group_from_src

This method creates a group from source. The source can be a group_snapshot or a source group. create_group_from_src has context, group object, a list of volume objects, group_snapshot object, a list of snapshot objects, source group object, and a list of source volume objects as input parameters. It returns model_update and volumes_model_update.

volumes_model_update is a list of dictionaries. It has to be built by the driver. An entry will be in this format: {id: xxx, status: xxx, }. model_update will be in this format: {status: xxx, }.

To be consistent with other volume operations, the manager will assume the operation is successful if no exception is thrown by the driver. For a successful operation, the driver can either build the model_update and volumes_model_update and return them or return None, None.

create_group_snapshot

This method creates a group_snapshot. It has context, group_snapshot object, and a list of snapshot objects as input parameters. It returns model_update and snapshots_model_update.

snapshots_model_update is a list of dictionaries. It has to be built by the driver. An entry will be in this format: {id: xxx, status: xxx, }. model_update will be in this format: {status: xxx, }. The driver should populate snapshots_model_update and model_update and return them.

The manager will check snapshots_model_update and update db accordingly for each snapshot. If the driver successfully created some snapshots but failed to create others, it should set statuses of the snapshots accordingly so that the manager can update db correctly.

If the status in any entry of snapshots_model_update is error, the status in model_update will be set to the same if it is not already error.

If the status in model_update is error, the manager will raise an exception and the status of group_snapshot will be set to error in the db. If snapshots_model_update is not returned by the driver, the manager will set the status of every snapshot to error in the except block.

If the driver raises an exception during the operation, it will be caught by the try-except block in the manager and the statuses of group_snapshot and all snapshots will be set to error.

For a successful operation, the driver can either build the model_update and snapshots_model_update and return them or return None, None. The statuses of group_snapshot and all snapshots will be set to available at the end of the manager function.

delete_group_snapshot

This method deletes a group_snapshot. It has context, group_snapshot object, and a list of snapshot objects. It returns model_update and snapshots_model_update.

snapshots_model_update is a list of dictionaries. It has to be built by the driver. An entry will be in this format: {id: xxx, status: xxx, }. model_update will be in this format: {status: xxx, }. The driver should populate snapshots_model_update and model_update and return them.

The manager will check snapshots_model_update and update db accordingly for each snapshot. If the driver successfully deleted some snapshots but failed to delete others, it should set statuses of the snapshots accordingly so that the manager can update db correctly.

If the status in any entry of snapshots_model_update is error_deleting or error, the status in model_update will be set to the same if it is not already error_deleting or error.

If the status in model_update is error_deleting or error, the manager will raise an exception and the status of group_snapshot will be set to error in the db. If snapshots_model_update is not returned by the driver, the manager will set the status of every snapshot to error in the except block.

If the driver raises an exception during the operation, it will be caught by the try-except block in the manager and the statuses of group_snapshot and all snapshots will be set to error.

For a successful operation, the driver can either build the model_update and snapshots_model_update and return them or return None, None. The statuses of group_snapshot and all snapshots will be set to deleted after the manager deletes them from db.

Migrate CGs to Generic Volume Groups

This section only affects drivers already supporting CGs by the Newton release. Drivers planning to add CG support after Newton are not affected.

A group type named default_cgsnapshot_type will be created by the migration script. The following command needs to be run to migrate migrate data and copy data from consistency groups to groups and from cgsnapshots to group_snapshots. Migrated consistency groups and cgsnapshots will be removed from the database:

```
cinder-manage db online_data_migrations
--max_count <max>
```

max_count is optional. Default is 50.

After running the above migration command to migrate CGs to generic volume groups, CG and group APIs work as follows:

- Create CG only creates in the groups table.
- Modify CG modifies in the CG table if the CG is in the CG table, otherwise it modifies in the groups table.
- Delete CG deletes from the CG or the groups table depending on where the CG is.
- List CG checks both CG and groups tables.
- List CG Snapshots checks both the CG and the groups tables.
- Show CG checks both tables.
- Show CG Snapshot checks both tables.
- Create CG Snapshot creates either in the CG or the groups table depending on where the CG is.
- Create CG from Source creates in either the CG or the groups table depending on the source.
- Create Volume adds the volume either to the CG or the group.
- default_cgsnapshot_type is reserved for migrating CGs.
- Group APIs will only write/read in/from the groups table.
- Group APIs will not work on groups with default_cgsnapshot_type.
- Groups with default_cgsnapshot_type can only be operated by CG APIs.
- After CG tables are removed, we will allow default_cgsnapshot_type to be used by group APIs.

References

[1] Migration script
https://review.openstack.org/#/c/350350/

[2] CG APIs changes for migrating CGs
https://review.openstack.org/#/c/401839/

[3] Example adding CG capability to generic volume groups
https://review.openstack.org/#/c/413927/

Database migrations

Note

This document details how to generate database migrations as part of a new feature or bugfix. For info on how to apply existing database migrations, refer to the documentation for the `cinder-manage db`

sync command in *cinder-manage*. For info on the general upgrade process for a cinder deployment, refer to *Upgrades*.

Occasionally the databases used in cinder will require schema or data migrations.

Schema migrations

Changed in version 19.0.0: (Xena)

The database migration engine was changed from `sqlalchemy-migrate` to `alembic`.

Changed in version 22.0.0: (Antelope)

The legacy `sqlalchemy-migrate`-based database migrations were removed.

The alembic database migration tool is used to manage schema migrations in cinder. The migration files and related metadata can be found in `cinder/db/migrations`. As discussed in *Upgrades*, these can be run by end users using the **cinder-manage db sync** command.

Note

There were also legacy migrations provided in the `cinder/db/legacy_migrations` directory . These were provided to facilitate upgrades from pre-Xena (19.0.0) deployments. They were removed in the 22.0.0 (Antelope) release.

The best reference for alembic is the alembic documentation, but a small example is provided here. You can create the migration either manually or automatically. Manual generation might be necessary for some corner cases such as renamed tables but auto-generation will typically handle your issues. Examples of both are provided below. In both examples, were going to demonstrate how you could add a new model, `Foo`, to the main database.

```
diff --git cinder/db/sqlalchemy/models.py cinder/db/sqlalchemy/models.py
index 7eab643e14..8f70bcdaca 100644
--- cinder/db/sqlalchemy/models.py
+++ cinder/db/sqlalchemy/models.py
@@ -73,6 +73,16 @@ def MediumText():
         sqlalchemy.dialects.mysql.MEDIUMTEXT(), 'mysql')


+class Foo(BASE, models.SoftDeleteMixin):
+    """A test-only model."""
+
+    __tablename__ = 'foo'
+
+    id = sa.Column(sa.Integer, primary_key=True)
+    uuid = sa.Column(sa.String(36), nullable=True)
+    bar = sa.Column(sa.String(255))
+
+
 class Service(BASE, models.SoftDeleteMixin):
     """Represents a running service on a host."""
```

(you might not be able to apply the diff above cleanly - this is just a demo).

Auto-generating migration scripts

In order for alembic to compare the migrations with the underlying models, it require a database that it can inspect and compare the models against. As such, we first need to create a working database. Well bypass `cinder-manage` for this and go straight to the **`alembic`** CLI. The `alembic.ini` file provided in the `cinder/db` directory is helpfully configured to use an SQLite database by default (`cinder.db`). Create this database and apply the current schema, as dictated by the current migration scripts:

```
$ tox -e venv -- alembic -c cinder/db/alembic.ini \
    upgrade head
```

Once done, you should notice the new `cinder.db` file in the root of the repo. Now, lets generate the new revision:

```
$ tox -e venv -- alembic -c cinder/db/alembic.ini \
    revision -m "Add foo model" --autogenerate
```

This will create a new file in `cinder/db/migrations/versions` with `add_foo_model` in the name including (hopefully!) the necessary changes to add the new `Foo` model. You **must** inspect this file once created, since theres a chance youll be missing imports or something else which will need to be manually corrected. Once youve inspected this file and made any required changes, you can apply the migration and make sure it works:

```
$ tox -e venv -- alembic -c cinder/db/alembic.ini \
    upgrade head
```

Manually generating migration scripts

For trickier migrations or things that alembic doesnt understand, you may need to manually create a migration script. This is very similar to the auto-generation step, with the exception being that you dont need to have a database in place beforehand. As such, you can simply run:

```
$ tox -e venv -- alembic -c cinder/db/alembic.ini \
    revision -m "Add foo model"
```

As before, this will create a new file in `cinder/db/migrations/versions` with `add_foo_model` in the name. You can simply modify this to make whatever changes are necessary. Once done, you can apply the migration and make sure it works:

```
$ tox -e venv -- alembic -c cinder/db/alembic.ini \
    upgrade head
```

Data migrations

Managing the Development Cycle

Release Cycle Tasks

This document describes the relative ordering and rough timeline for all of the steps related to tasks that need to be completed during a release cycle for Cinder.

Before PTG (after closing previous release)

1. Collect topics and prepare notes for PTG discussions in an etherpad. The PTGbot will generate a list of etherpads at some point that will be named according to the convention:

   ```
   https://etherpad.openstack.org/p/<release-name>-ptg-cinder
   ```

   (You can use a different name, but following the convention makes it easy to locate the etherpad for any project for any release. Something weve done in the past is to do the planning on an etherpad named:

   ```
   https://etherpad.openstack.org/p/<release-name>-ptg-cinder-planning
   ```

   and then move the topics over to the real etherpad when the team has decided on what to include and the ordering. Do whatever works for you. Just make sure the team knows where the planning etherpad is and give everyone plenty of reminders to add topics.

2. Add any Cinder-specific schedule information to the release calendar as soon as its available. Example patch: https://review.opendev.org/c/openstack/releases/+/754484

   - We used to wait to do this until after proposed deadlines were discussed at the PTG, but recently people have been getting antsy about what the deadlines are as soon as the stable branch for the previous release is cut (which is roughly a month before the PTG). So you may want to go ahead and post the patch early and announce the dates at a Cinder meeting so that people can point out conflicts. Or do it the old-fashioned way and work it out at the PTG. Either way, the point is to make sure you dont forget to add Cinder-specific dates to the main release schedule.

3. Review the *Cinder Groups in Gerrit and Launchpad*.

Between PTG and Milestone-1

1. Review output from the PTG and set Review-Priority on any high priority items identified from those discussions. Send out recap to the mailing list.

2. Review information about standing Cinder meetings posted at https://meetings.opendev.org/ in case any changes were discussed at the PTG. You make changes by proposing a patch to https://opendev.org/opendev/irc-meetings

   Example patch: https://review.opendev.org/c/opendev/irc-meetings/+/695339

3. **Supported Python versions**

   - The supported Python runtimes for the cycle may have changed from the previous cycle. You can find them at https://governance.openstack.org/tc/reference/runtimes/
   - Review the tox testenvs defined in `tox.ini` and make sure there are functional testenvs for each. You dont have to worry about unit teststox is smart enough to know what to do for those-but if you specify `tox -e functional-py312` tox will bomb unless theres a functional-py312 testenv defined.
   - The OpenStack required check and gate tests are defined in a template in zuul.d/project-templates.yaml in the openstack/openstack-zuul-jobs repo. The template is maintained by the OpenStack QA team. It should have an easily recognizable name, for example, `openstack-python3-zed-jobs`.

Usually there will be autogenerated patches for each cinder project repo to change the template from the previous cycles to the current cycles, so watch for those. Or you can proactively make the changes yourself as soon as the template is available.

Example new template patch: https://review.opendev.org/c/openstack/openstack-zuul-jobs/+/831826

- Check the `setup.cfg` file in each cinder deliverable to make sure that the claimed supported Python versions line up with the cycles supported Python versions.

4. Focus on spec reviews to get them approved and updated early in the cycle to allow enough time for implementation.
5. Review new driver submissions and give early feedback so there isnt a rush at the new driver deadline. Check for status of third party CI and any missing steps they need to know about.
6. Review community-wide goals and decide a plan or response to them.

Milestone-1

1. Propose library releases for os-brick or python-cinderclient if there are merged commits ready to be released. Watch for any releases proposed by the release team.
2. Check progress on new drivers and specs and warn contributors if it looks like they are at risk for making it in this cycle.

Between Milestone-1 and Milestone-2

1. cinderlib is a trailing deliverable type on a cycle-with-intermediary release model. That means that its release for the *previous* cycle hasnt happened yet. The release must happen no later than 3 months after the main release, which will put it roughly one week before Milestone-2 (check the current release schedule for the exact deadline). Example patch: https://review.opendev.org/c/openstack/releases/+/742503
2. Review stable backports and release status.
3. The Cinder Spec Freeze usually occurs sometime within this window. After all the approved specs have merged, propose a patch that adds a directory for the next release. (You may have to wait until the release name has been determined by the TC.) Example patch: https://review.opendev.org/c/openstack/cinder-specs/+/778436
4. Watch for and respond to updates to new driver patches.

Milestone-2

1. Propose library releases for os-brick or python-cinderclient if there are merged commits ready to be released. Watch for any releases proposed by the release team.

Between Milestone-2 and Milestone-3

1. Review stable backports and release status.
2. Set Review-Priority for any os-brick changes that are needed for feature work to make sure they are ready by the library freeze prior to Milestone-3.

3. Make sure any new feature work that needs client changes are proposed and on track to land before the client library freeze at Milestone-3. Ensure microversion bumps are reflected in cinderclient/api_versions.py MAX_VERSION.
4. The week before Milestone-3, propose releases for unreleased changes in os-brick. (The release team may have already proposed an auto- generated patch 1-2 weeks earlier; make sure you -1 it if there are still changes that need to land in os-brick before release.) Include branch request for stable/$series creation. Example patch: https://review.opendev.org/c/openstack/releases/+/804670

Milestone-3

1. Propose releases for unreleased changes in python-cinderclient and python-brick-cinderclient-ext. These will be the official cycle releases for these deliverables. Watch for a release patch proposed by the release team; it may need to be updated to include all the appropriate changes. Include branch request for stable/$series creation. Example patches: | https://review.opendev.org/c/openstack/releases/+/806583 | https://review.opendev.org/c/openstack/releases/+/807167
2. Set Review-Priority -1 for any feature work not complete in time for inclusion in this cycle. Remind contributors that FFE will need to be requested to still allow it in this cycle.
3. Complete the responses to community-wide goals if not already done.
4. Add cycle-highlights in the releases deliverable file. The deadline for this has been moved up (since wallaby) to the Friday of M-3 week. (There should be an entry on the cycle release schedule, and a reminder email with subject [PTLs][release] xxx Cycle Highlights to the ML.)

   The Foundation people use the info to start preparing press releases for the cycle coordinated release, so its good to have key features mentioned. (If something has an FFE and youre not sure if it will land, you can always update the cycle-highlights later and shoot an email to whoever sent out the reminder so they know to look for it.)

   Example patch: https://review.opendev.org/c/openstack/releases/+/807398

Between Milestone-3 and RC1

1. Make sure the maximum microversion is up-to-date in the version history file `cinder/api/openstack/rest_api_version_history.rst`
   - Any patch that bumped the microversion should have already included an entry in this file; you need to add (Maximum in <release-name>) to the last (highest) entry.
   - This file is pulled into the api-ref by the documentation build process.
2. Prepare prelude release notes as summaries of the content of the release so that those are merged before their first release candidate.
3. Check the Driver Removal History section (bottom) of `doc/source/reference/support-matrix.rst` to make sure any drivers removed during the cycle are mentioned there.
4. Check the upgrade check tool `cmd/status.py` to make sure the removed drivers list is up to date.

RC1 week

1. Propose RC1 release for cinder or watch for proposal from the release team. Include `stable/$series` branching request with the release.
2. Update any cycle-highlights for the release cycle if there was something you werent sure about at M-3.
3. Remind contributors that `master` is now the next cycle but focus should be on wrapping up the current cycle.
4. Watch for translation and new stable branch patches and merge them quickly.

Between RC1 and Final

1. The release team has started adding a release-notes field to the deliverables yaml files. You can watch for the patch and vote on it if you see it. Example patch: https://review.opendev.org/c/openstack/releases/+/810236
2. Related to the previous point: at this time in the cycle, the release notes for all the cinder cycle deliverables (cinder, os-brick, python-cinderclient, and python-brick-cinderclient-ext) should have been published automatically at https://docs.openstack.org/releasenotes/. Sometimes the promotion job fails, though, so its good to check that the release notes for the current cycle are actually there.
3. Propose additional RC releases as needed.

   Note

   Try to avoid creating more than 3 release candidates so we are not creating candidates that consumers are then trained to ignore. Each release candidate should be kept for at least 1 day, so if there is a proposal to create RCx but clearly a reason to create another one, delay RCX to include the additional patches.

4. Watch for translation patches and merge them quickly.
5. Make sure final RC request is done one week before the final release date.
6. Watch for the final release proposal from the release team to review and +1 so team approval is included in the metadata that goes onto the signed tag. Example patch:
   https://review.opendev.org/c/openstack/releases/+/785754
   Heres what it looks like when people forget to check for this patch:
   https://review.opendev.org/c/openstack/releases/+/812251

Final Release

1. Start planning for next release cycle.
2. Check for bugfixes that would be good to backport to older stable branches.
3. Propose any bugfix releases for things that did not make the freeze for final library or service releases.

Post-Final Release

1. Make sure at least three SQLAlchemy-Migrate migrations are reserved for potential backports. Example patch: https://review.opendev.org/c/openstack/cinder/+/649436
2. Unblock any new driver submission patches that missed the previous release cycles deadline.
3. Review approved cinder-specs that were merged to the previous cycle folder that did not get implemented. Revert or move those specs to the next cycless folder.
4. The oldest active stable branch (that is, the oldest one you can still release from) will go to Extended Maintenance mode shortly after the coordinated release. Watch for an email notification from the release team about the projected date, which you can also find in the Next Phase column for that release series on https://releases.openstack.org
   - Prioritize any open reviews that should get into the final stable release from this branch for all relevant cinder deliverables and motivate the cinder-stable-maint cores to review them.
   - Propose a final release for any deliverable that needs one. Example patch: https://review.opendev.org/c/openstack/releases/+/761929
   - The release team will probably propose a placeholder patch to tag the stable branch for each deliverable as <release>-em (or if they havent gotten around to it yet, you can propose it yourself). Verify that the hash is at the current HEAD for each deliverable (it may have changed if some last-minute stuff was merged). Example patch: https://review.opendev.org/c/openstack/releases/+/762372
   - After the transition to EM patch has merged, update the zuul jobs for the cinder-tempest-plugin. We always have 3 jobs for the active stable branches plus jobs for master. Add a new job for the most recent release and remove the job for the stable branch that just went to EM. Example patch: https://review.opendev.org/c/openstack/cinder-tempest-plugin/+/756330

Cinder Groups in Gerrit and Launchpad

Cinder-related groups in Launchpad

group	what	who	where
Cinder team	not sure, exactly	an open team, anyone with a Launchpad account can join	https://launchpad.net/~cinder
Cinder Bug Team team	can triage (change status fields) on bugs	an open team, people self-nominate	https://launchpad.net/~cinder-bugs
Cinder Drivers team	Maintains the Launchpad space for Cinder, os-brick, cinderlib, python-cinderclient, and cinder-tempest-plugin	Anyone who is interested in doing some work, has a Launchpad account, and is approved by the current members	https://launchpad.net/~cinder-drivers
Cinder Core security contacts team	can see and work on private security bugs while they are under embargo	subset of cinder-core (the OpenStack Vulnerablity Management Team likes to keep this team small), so even though the PTL can add people, you should propose them on the mailing list first	https://launchpad.net/~cinder-coresec

Cinder-related groups in Gerrit

The Cinder project has total control over the membership of these groups.

group	what	who	where
cinder-core	+2 powers in Cinder project code repositories	cinder core reviewers	https://review.opendev.org/#/admin/groups/83,members
cinder-specs-core	+2 powers in cinder-specs repository	cinder-core plus other appropriate people	https://review.opendev.org/#/admin/groups/344,members
cinder-tempest-plugin-core	+2 powers on the cinder-tempest-plugin repository	cinder-core plus other appropriate people	https://review.opendev.org/#/admin/groups/2088,members
rbd-iscsi-client-core	+2 powers on the rbd-iscsi-client repository	cinder-core (plus others if appropriate; currently only cinder-core)	https://review.opendev.org/admin/groups/b25813f5baef62b9449371c91f7dbacbcf7bc6d6,members

The Cinder project shares control over the membership of these groups. If you want to add someone to one of these groups who doesnt already have membership by being in an included group, be sure to include the other groups or individual members in your proposal email.

group	what	who	where
cinder-stable-maint	+2 powers on backports to stable branches	subset of cinder-core (selected in consultation with stable-maint-core) plus the stable-maint-core team	https://review.opendev.org/#/admin/groups/534,members
devstack-plugin-ceph-core	+2 powers on the code repo for the Ceph devstack plugin	cinder-core, devstack-core, manila-core, qa-release, other appropriate people	https://review.opendev.org/#/admin/groups/1196,members
devstack-plugin-nfs-core	+2 powers on the code repo for the NFS devstack plugin	cinder-core, devstack-core, other appropriate people	https://review.opendev.org/#/admin/groups/1330,members
devstack-plugin-open-cas-core	+2 powers on the code repo for the Open CAS devstack plugin	cinder-core, devstack-core, other appropriate people	https://review.opendev.org/#/admin/groups/2082,members

NOTE: The following groups exist, but I dont think they are used for anything anymore.

group	where
cinder-ci	https://review.opendev.org/#/admin/groups/508,members
cinder-milestone	https://review.opendev.org/#/admin/groups/82,members
cinder-release	https://review.opendev.org/#/admin/groups/144,members
cinder-release-branch	https://review.opendev.org/#/admin/groups/1507,members

How Gerrit groups are connected to project repositories

The connection between the groups defined in gerrit and what they can do is defined in the project-config repository: https://opendev.org/openstack/project-config

- `gerrit/projects.yaml` sets the config file for a project
- `gerrit/acls/openstack` contains the config files

The Special Relationship with OpenStack Command Line Client and SDK

The OpenStack Command Line Client (aka OSC) and the OpenStack SDK provide unified interfaces across most of the OpenStack APIs. To facilitate this, they make use of two kinds of core teams:

- service cores: people very familiar with the particular API thats being given an interface in the OSC or SDK. For example, cinder cores can be service cores for implementations in OSC or SDK that provide an interface to the Block Storage API.
- additionally, the OSC and SDK projects have their own core teams whose members have a broader vision over the OSC and SDK, and therefore can enforce consistency across all the service code. This way, end users will be provided with a consistent and predictable interface to OpenStack as a whole.

The cinder-core gerrit group acts as service cores for the OSC and SDK. This means that the cinder-core members have +2 powers on OSC or SDK reviews but do not have permission to approve patches. The connection between cinder-core and the python-openstackclient and openstacksdk code repositories is made directly in their ACL files in the `project-config` repository:

- `gerrit/acls/openstack/openstacksdk.config`
- `gerrit/acls/openstack/python-openstackclient.config`

Documentation Contribution

Background Concepts for Cinder

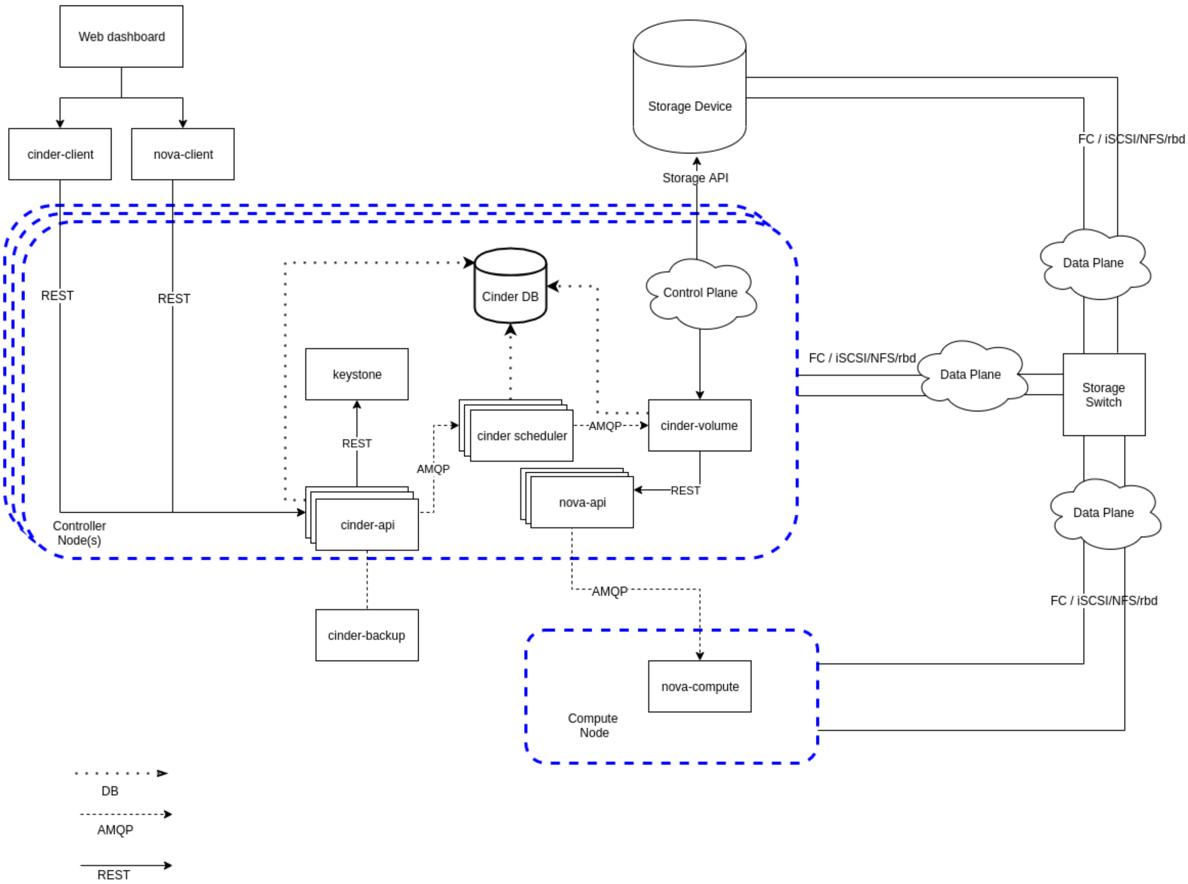
Cinder System Architecture

The Cinder Block Storage Service is intended to be ran on one or more nodes.

Cinder uses a sql-based central database that is shared by all Cinder services in the system. The amount and depth of the data fits into a sql database quite well. For small deployments this seems like an optimal solution. For larger deployments, and especially if security is a concern, cinder will be moving towards multiple data stores with some kind of aggregation system.

Components

Below you will find a brief explanation of the different components.



- DB: sql database for data storage. Used by all components (LINKS NOT SHOWN).
- Web Dashboard: potential external component that talks to the api.

- api: component that receives http requests, converts commands and communicates with other components via the queue or http.
- Auth Manager: component responsible for users/projects/and roles. Can backend to DB or LDAP. This is not a separate binary, but rather a python class that is used by most components in the system.
- scheduler: decides which host gets each volume.
- volume: manages dynamically attachable block devices.
- backup: manages backups of block storage devices.

Volume Attach/Detach workflow

There are six API calls associated with attach/detach of volumes in Cinder (3 calls for each operation). This can lead to some confusion for developers trying to work on Cinder. The convention is actually quite simple, although it may be difficult to decipher from the code.

Attach/Detach Operations are multi-part commands

There are three things that happen in the workflow for an attach or detach call.

1. Update the status of the volume in the DB (ie attaching/detaching)

- For Attach, this is the cinder.volume.api.reserve_volume method
- For Detach, the analogous call is cinder.volume.api.begin_detaching

2. Handle the connection operations that need to be done on the Volume

- For Attach, this is the cinder.volume.api.initialize_connection method
- For Detach, the analogous call is cinder.volume.api.terminate_connection

3. Finalize the status of the volume and release the resource

- For attach, this is the cinder.volume.api.attach method
- For detach, the analogous call is cinder.volume.api.detach

Attach workflow

reserve_volume(self, context, volume)

Probably the most simple call in to Cinder. This method simply checks that the specified volume is in an available state and can be attached. Any other state results in an Error response notifying Nova that the volume is NOT available. The only valid state for this call to succeed is available.

NOTE: multi-attach will add in-use to the above acceptable states.

If the volume is in fact available, we immediately issue an update to the Cinder database and mark the status of the volume to attaching thereby reserving the volume so that it wont be used by another API call anywhere else.

initialize_connection(self, context, volume, connector)

This is the only attach related API call that should be doing any significant work. This method is responsible for building and returning all of the info needed by the caller (Nova) to actually attach the specified volume to the remote node. This method returns vital information to the caller that includes things like CHAP credential, iqn and lun information. An example response is shown here:

```
{
    'driver_volume_type': 'iscsi',
    'data': {
        'auth_password': 'YZ2Hceyh7VySh5HY',
        'target_discovered': False,
        'encrypted': False,
        'qos_specs': None,
        'target_iqn': 'iqn.2010-10.org.openstack:volume-8b1ec3fe-8c57-45ca-
↪a1cf-a481bfc8fce2',
        'target_portal': '11.0.0.8:3260',
        'volume_id': '8b1ec3fe-8c57-45ca-a1cf-a481bfc8fce2',
        'target_lun': 1,
        'access_mode': 'rw',
        'auth_username': 'nE9PY8juynmmZ95F7Xb7',
        'auth_method': 'CHAP'
    }
}
```

In the process of building this data structure, the Cinder Volume Manager makes a number of calls to the backend driver, and builds a volume_attachment entry in the database to store the connection information passed in via the connector object.

driver.validate_connector

Simply verifies that the initiator data is included in the passed in connector (there are some drivers that utilize pieces of this connector data, but in the case of the reference, it just verifies its there).

driver.create_export

This is the target specific, persistent data associated with a volume. This method is responsible for building an actual iSCSI target, and providing the location and auth information which will be used to form the response data in the parent request. We call this infor the model_update and its used to update vital target information associated with the volume in the Cinder database.

driver.initialize_connection

Now that weve actually built a target and persisted the important bits of information associated with it, were ready to actually assign the target to a volume and form the needed info to pass back out to our caller. This is where we finally put everything together and form the example data structure response shown earlier.

This method is sort of deceptive, it does a whole lot of formatting of the data weve put together in the create_export call, but it doesnt really offer any new info. Its completely dependent on the information that was gathered in the create_export call and put into the database. At this point, all were doing is taking all the various entries from the database and putting it together into the desired format/structure.

The key method call for updating and obtaining all of this info was done by the create_export call. This formatted data is then passed back up to the API and returned as the response back out to Nova.

At this point, we return attach info to the caller that provides everything needed to make the remote iSCSI connection.

attach(self, context, volume, instance_uuid, host_name, mountpoint, mode)

This is the last call that *should* be pretty simple. The intent is that this is simply used to finalize the attach process. In other words, we simply update the status on the Volume in the database, and provide a mechanism to notify the driver that the attachment has completed successfully.

Theres some additional information that has been added to this finalize call over time like instance_uuid, host_name etc. Some of these are only provided during the actual attach call and may be desired for some drivers for one reason or another.

Detach workflow

begin_detaching(self, context, volume)

Analogous to the Attach workflows `reserve_volume` method. Performs a simple conditional update of Volume status to `detaching`.

terminate_connection(self, context, volume, connector, force=False)

Analogous to the Attach workflows `initialize_connection` method.

Used to send calls down to drivers/target-drivers to do any sort of cleanup they might require.

For most this is a noop, as connections and **iscsi session management is the responsibility of the initiator**. HOWEVER, there are a number of special cases here, particularly for target-drivers like LIO that use access-groups, in those cases they remove the initiator from the access list during this call which effectively closes sessions from the target side.

detach(self, context, volume, attachment_id)

The final update to the DB and yet another opportunity to pass something down to the volume-driver. Initially a simple call-back that now has quite a bit of cruft built up in the volume-manager.

For drivers like LVM this again is a noop and just updates the db entry to mark things as complete and set the volume to available again.

Volume Attach/Detach workflow - V2

Previously there were six API calls associated with attach/detach of volumes in Cinder (3 calls for each operation). As the projects grew and the functionality of *simple* things like attach/detach evolved things have become a bit vague and we have a number of race issues during the calls that continually cause us some problems.

Additionally, the existing code path makes things like multi-attach extremely difficult to implement due to no real good tracking mechanism of attachment info.

To try and improve this weve proposed a new Attachments Object and API. Now we keep an Attachment record for each attachment that we want to perform as opposed to trying to infer the information from the Volume Object.

Attachment Object

We actually already had a VolumeAttachment Table in the db, however we werent really using it, or at least using it efficiently. For V2 of attach implementation (V3 API) flow well use the Attachment Table (object) as the primary handle for managing attachment(s) for a volume.

In addition, we also introduce the AttachmentSpecs Table which will store the connector information for an Attachment so we no longer have the problem of lost connector info, or trying to reassemble it.

New API and Flow

attachment-create

` cinder --os-volume-api-version 3.27 attachment-create <volume-id> <instance-uuid> `

The attachment_create call simply creates an empty Attachment record for the specified Volume with an Instance UUID field set. This is particularly useful for cases like Nova Boot from Volume where Nova hasnt sent the job to the actual Compute host yet, but needs to make initial preparations to reserve the volume for use, so here we can reserve the volume and indicate that we will be attaching it to <Instance-UUID> in the future.

Alternatively, the caller may provide a connector in which case the Cinder API will create the attachment and perform the update on the attachment to set the connector info and return the connection data needed to make a connection.

The attachment_create call can be used in one of two ways:

1. Create an empty Attachment object (reserve). In this case the attachment_create call requires an instance_uuid and a volume_uuid, and just creates an empty Attachment object and returns the UUID of Attachment to the caller.

2. Create and complete the Attachment process in one call. The reserve process is only needed in certain cases, in many cases Nova actually has enough information to do everything in a single call. Also, non-nova consumers typically dont require the granularity of a separate reserve at all.

   To perform the complete operation, include the connector data in the attachment_create call and the Cinder API will perform the reserve and initialize the connection in the single request.

This full usage of attachment-create would be:

```
usage: cinder --os-volume-api-version 3.27 attachment-create
       <volume>  <instance_uuid> ...

Positional arguments:
<volume>                  Name or ID of volume or volumes to attach.
<instance_uuid>           ID of instance attaching to.

Optional arguments:
--connect <connect>       Make an active connection using provided connector␣
↪info (True or False).
--initiator <initiator>   iqn of the initiator attaching to. Default=None.
--ip <ip>                 ip of the system attaching to. Default=None.
--host <host>             Name of the host attaching to. Default=None.
--platform <platform>     Platform type. Default=x86_64.
```

(continues on next page)

(continued from previous page)

```
--ostype <ostype>         OS type. Default=linux2.
--multipath <multipath>   Use multipath. Default=False.
--mountpoint <mountpoint> Mountpoint volume will be attached at. Default=None.
```

Returns the connection information for the attachment:

```
+-------------------+-------------------------------------------------------------
↪--------------+
| Property          | Value                                                       ␣
↪              |
+-------------------+-------------------------------------------------------------
↪--------------+
| access_mode       | rw                                                          ␣
↪              |
| attachment_id     | 6ab061ad-5c45-48f3-ad9c-bbd3b6275bf2                        ␣
↪              |
| auth_method       | CHAP                                                        ␣
↪              |
| auth_password     | kystSioDKHSV2j9y                                            ␣
↪              |
| auth_username     | hxGUgiWvsS4GqAQcfA78                                        ␣
↪              |
| encrypted         | False                                                       ␣
↪              |
| qos_specs         | None                                                        ␣
↪              |
| target_discovered | False                                                       ␣
↪              |
| target_iqn        | iqn.2010-10.org.openstack:volume-23212c97-5ed7-42d7-
↪b433-dbf8fc38ec35 |
| target_lun        | 0                                                           ␣
↪              |
| target_portal     | 192.168.0.9:3260                                            ␣
↪              |
| volume_id         | 23212c97-5ed7-42d7-b433-dbf8fc38ec35                        ␣
↪              |
+-------------------+-------------------------------------------------------------
↪--------------+
```

attachment-update

` cinder --os-volume-api-version 3.27 attachment-update <attachment-id> `

Once we have a reserved volume, this CLI can be used to update an attachment for a cinder volume. This call is designed to be more of an attachment completion than anything else. It expects the value of a connector object to notify the driver that the volume is going to be connected and where its being connected to. The usage is the following:

```
usage: cinder --os-volume-api-version 3.27 attachment-update
              <attachment-id> ...
```

(continues on next page)

(continued from previous page)

```
Positional arguments:
  <attachment-id>           ID of attachment.

Optional arguments:
  --initiator <initiator>   iqn of the initiator attaching to. Default=None.
  --ip <ip>                 ip of the system attaching to. Default=None.
  --host <host>             Name of the host attaching to. Default=None.
  --platform <platform>     Platform type. Default=x86_64.
  --ostype <ostype>         OS type. Default=linux2.
  --multipath <multipath>   Use multipath. Default=False.
  --mountpoint <mountpoint> Mountpoint volume will be attached at.␣
↪Default=None.
```

attachment-delete

` cinder --os-volume-api-version 3.27 attachment-delete <attachment-id> `

Cinder Thin provisioning and Oversubscription

Background

After the support on Cinder for Thin provisioning, driver maintainers have been struggling to understand what is the expected behavior of their drivers and what exactly each value reported means. This document summarizes the concepts, definitions and terminology from all specs related to the subject and should be used as reference for new drivers implementing support for thin provisioning.

Core concepts and terminology

In order to maintain the same behavior among all drivers, we first need to define some concepts used throughout drivers. This terminology is discussed and defined in this spec[1] and should be used as reference in further implementations.

Stats to be reported

The following fields should be reported by drivers supporting thin provisioning on the get_volume_stats() function:

Mandatory Fields

```
thin_provisioning_support = True (or False)
```

Optional Fields

```
thick_provisioning_support = True (or False)
provisioned_capacity_gb = PROVISIONED_CAPACITY
max_over_subscription_ratio = MAX_RATIO
```

Note

If provisioned_capacity_gb is not reported, the value used in the scheduler calculations and filtering is allocated_capacity_gb.

Note

If max_over_subscription_ratio is not reported, the scheduler will use the value defined on the [DEFAULT] section. This falls back to the default value (20.0) if not set by the user.

[1] https://specs.openstack.org/openstack/cinder-specs/specs/queens/provisioning-improvements.html

Threading model

All OpenStack services use *green thread* model of threading, implemented through using the Python eventlet and greenlet libraries.

Green threads use a cooperative model of threading: thread context switches can only occur when specific eventlet or greenlet library calls are made (e.g., sleep, certain I/O calls). From the operating systems point of view, each OpenStack service runs in a single thread.

The use of green threads reduces the likelihood of race conditions, but does not completely eliminate them. In some cases, you may need to use the `@utils.synchronized(...)` decorator to avoid races.

In addition, since there is only one operating system thread, a call that blocks that main thread will block the entire process.

Yielding the thread in long-running tasks

If a code path takes a long time to execute and does not contain any methods that trigger an eventlet context switch, the long-running thread will block any pending threads.

This scenario can be avoided by adding calls to the eventlet sleep method in the long-running code path. The sleep call will trigger a context switch if there are pending threads, and using an argument of 0 will avoid introducing delays in the case that there is only a single green thread:

```
from eventlet import greenthread
...
greenthread.sleep(0)
```

In current code, time.sleep(0) does the same thing as greenthread.sleep(0) if time module is patched through eventlet.monkey_patch(). To be explicit, we recommend contributors use `greenthread.sleep()` instead of `time.sleep()`.

MySQL access and eventlet

There are some MySQL DB API drivers for oslo.db, like PyMySQL, MySQL-python etc. PyMySQL is the default MySQL DB API driver for oslo.db, and it works well with eventlet. MySQL-python uses an external C library for accessing the MySQL database. Since eventlet cannot use monkey-patching to intercept blocking calls in a C library, queries to the MySQL database using libraries like MySQL-python will block the main thread of a service.

The Diablo release contained a thread-pooling implementation that did not block, but this implementation resulted in a bug and was removed.

See this mailing list thread for a discussion of this issue, including a discussion of the impact on performance.

Internationalization

For internationalization guidelines, see the oslo.i18n documentation. The information below can be used to get started.

Cinder uses gettext so that user-facing strings such as log messages appear in the appropriate language in different locales.

To use gettext, make sure that the strings passed to the logger are wrapped in a `_Lx()` function call. For example:

```
LOG.info(_LI("block_device_mapping %s"), block_device_mapping)
```

There are a few different _() translation markers, depending on the logging level of the text:

- _LI() - Used for INFO level log messages
- _LW() - Used for WARNING level log messages
- _LE() - Used for ERROR level log messages (this includes LOG.exception)
- _() - Used for any exception messages, including strings used for both logging and exceptions.

Note

Starting with the Pike series, OpenStack no longer supports log translation markers like `_Lx()`, only `_()` should still be used for exceptions that could be user facing. It is not necessary to add `_Lx()` translation instructions to new code, and the instructions can be removed from old code. Refer to the email thread understanding log domain change on the openstack-dev mailing list for more details.

Do not use `locals()` for formatting messages because:

1. It is not as clear as using explicit dicts.
2. It could produce hidden errors during refactoring.
3. Changing the name of a variable causes a change in the message.
4. It creates a lot of otherwise unused variables.

If you do not follow the project conventions, your code may cause pep8 hacking check failures.

For translation to work properly, the top level scripts for Cinder need to first do the following before any Cinder modules are imported:

```
from cinder import i18n
i18n.enable_lazy()
```

Note: this should *only* be called from top level scripts - no library code or common modules should call this method.

Any files that use the _() for translation then must have the following lines:

```
from cinder.i18n import _
```

If the above code is missing, it may result in an error that looks like:

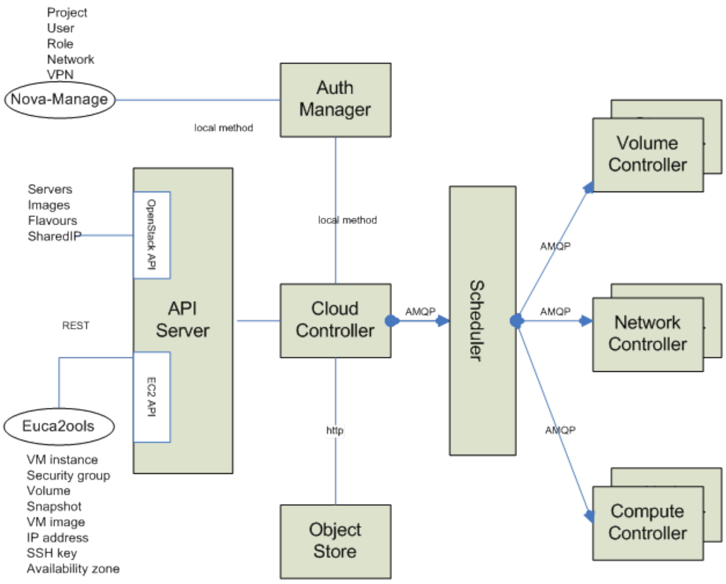
```
NameError: name '_' is not defined
```

AMQP and Cinder

AMQP is the messaging technology chosen by the OpenStack cloud. The AMQP broker, either RabbitMQ or Qpid, sits between any two Cinder components and allows them to communicate in a loosely coupled fashion. More precisely, Cinder components (the compute fabric of OpenStack) use Remote Procedure Calls (RPC hereinafter) to communicate to one another; however such a paradigm is built atop the publish/subscribe paradigm so that the following benefits can be achieved:

- Decoupling between client and servant (such as the client does not need to know where the servants reference is).
- Full a-synchronism between client and servant (such as the client does not need the servant to run at the same time of the remote call).
- Random balancing of remote calls (such as if more servants are up and running, one-way calls are transparently dispatched to the first available servant).

Cinder uses direct, fanout, and topic-based exchanges. The architecture looks like the one depicted in the figure below:



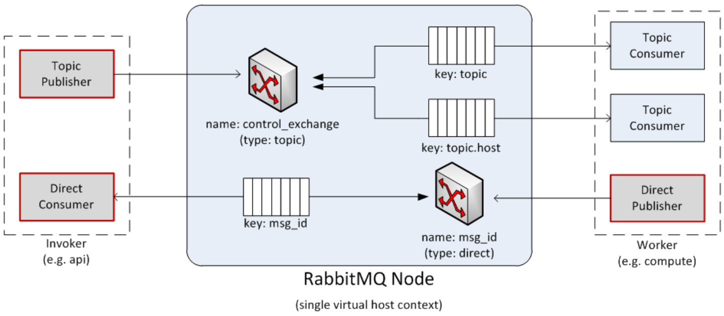
Cinder implements RPC (both request+response, and one-way, respectively nicknamed rpc.call and rpc.cast) over AMQP by providing an adapter class which take cares of marshaling and unmarshaling of messages into function calls. Each Cinder service (for example Scheduler, Volume, etc.) create two queues at the initialization time, one which accepts messages with routing keys NODE-TYPE.NODE-ID (for example cinder-volume.hostname) and another, which accepts messages with routing keys as generic NODE-TYPE (for example cinder-volume). The API acts as a consumer when RPC calls are request/response, otherwise is acts as publisher only.

Cinder RPC Mappings

The figure below shows the internals of a message broker node (referred to as a RabbitMQ node in the diagrams) when a single instance is deployed and shared in an OpenStack cloud. Every Cinder component connects to the message broker and, depending on its personality, may use the queue either as an Invoker (such as API or Scheduler) or a Worker (such as Volume). Invokers and Workers do not actually exist in the Cinder object model, but we are going to use them as an abstraction for sake of clarity. An Invoker is a component that sends messages in the queuing system via two operations: 1) rpc.call and ii) rpc.cast; a Worker is a component that receives messages from the queuing system and reply accordingly to rpc.call operations.

Figure 2 shows the following internal elements:

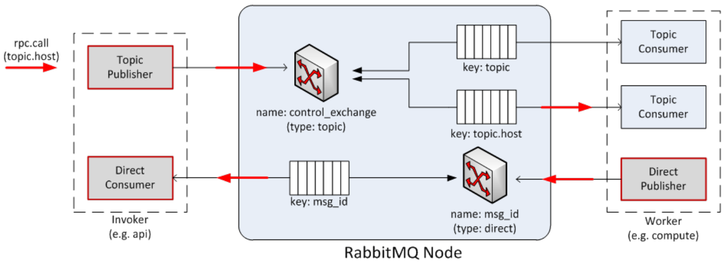
- Topic Publisher: a Topic Publisher comes to life when an rpc.call or an rpc.cast operation is executed; this object is instantiated and used to push a message to the queuing system. Every publisher connects always to the same topic-based exchange; its life-cycle is limited to the message delivery.
- Direct Consumer: a Direct Consumer comes to life if (an only if) a rpc.call operation is executed; this object is instantiated and used to receive a response message from the queuing system; Every consumer connects to a unique direct-based exchange via a unique exclusive queue; its life-cycle is limited to the message delivery; the exchange and queue identifiers are determined by a UUID generator, and are marshaled in the message sent by the Topic Publisher (only rpc.call operations).
- Topic Consumer: a Topic Consumer comes to life as soon as a Worker is instantiated and exists throughout its life-cycle; this object is used to receive messages from the queue and it invokes the appropriate action as defined by the Worker role. A Topic Consumer connects to the same topic-based exchange either via a shared queue or via a unique exclusive queue. Every Worker has two topic consumers, one that is addressed only during rpc.cast operations (and it connects to a shared queue whose exchange key is topic) and the other that is addressed only during rpc.call operations (and it connects to a unique queue whose exchange key is topic.host).
- Direct Publisher: a Direct Publisher comes to life only during rpc.call operations and it is instantiated to return the message required by the request/response operation. The object connects to a direct-based exchange whose identity is dictated by the incoming message.
- Topic Exchange: The Exchange is a routing table that exists in the context of a virtual host (the multi-tenancy mechanism provided by Qpid or RabbitMQ); its type (such as topic vs. direct) determines the routing policy; a message broker node will have only one topic-based exchange for every topic in Cinder.
- Direct Exchange: this is a routing table that is created during rpc.call operations; there are many instances of this kind of exchange throughout the life-cycle of a message broker node, one for each rpc.call invoked.
- Queue Element: A Queue is a message bucket. Messages are kept in the queue until a Consumer (either Topic or Direct Consumer) connects to the queue and fetch it. Queues can be shared or can be exclusive. Queues whose routing key is topic are shared amongst Workers of the same personality.



RPC Calls

The diagram below shows the message flow during an rpc.call operation:

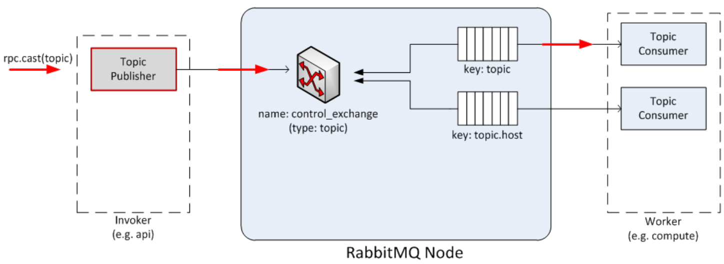
1. a Topic Publisher is instantiated to send the message request to the queuing system; immediately before the publishing operation, a Direct Consumer is instantiated to wait for the response message.
2. once the message is dispatched by the exchange, it is fetched by the Topic Consumer dictated by the routing key (such as topic.host) and passed to the Worker in charge of the task.
3. once the task is completed, a Direct Publisher is allocated to send the response message to the queuing system.
4. once the message is dispatched by the exchange, it is fetched by the Direct Consumer dictated by the routing key (such as msg_id) and passed to the Invoker.



RPC Casts

The diagram below the message flow during an rpc.cast operation:

1. A Topic Publisher is instantiated to send the message request to the queuing system.
2. Once the message is dispatched by the exchange, it is fetched by the Topic Consumer dictated by the routing key (such as topic) and passed to the Worker in charge of the task.



AMQP Broker Load

At any given time the load of a message broker node running either Qpid or RabbitMQ is function of the following parameters:

- Throughput of API calls: the number of API calls (more precisely rpc.call ops) being served by the OpenStack cloud dictates the number of direct-based exchanges, related queues and direct consumers connected to them.
- Number of Workers: there is one queue shared amongst workers with the same personality; however there are as many exclusive queues as the number of workers; the number of workers dictates also the number of routing keys within the topic-based exchange, which is shared amongst all workers.

The figure below shows the status of a RabbitMQ node after Cinder components bootstrap in a test environment (phantom is hostname). Exchanges and queues being created by Cinder components are:

- **Exchanges**
    1. cinder-scheduler_fanout (fanout exchange)
    2. cinder-volume.phantom@lvm_fanout (fanout exchange)
    3. cinder-volume_fanout (fanout exchange)
    4. openstack (topic exchange)
- **Queues**
    1. cinder-scheduler
    2. cinder-scheduler.phantom
    3. cinder-scheduler_fanout_572c35c0fbf94560b4c49572d5868ea5
    4. cinder-volume
    5. cinder-volume.phantom@lvm
    6. cinder-volume.phantom@lvm.phantom
    7. cinder-volume.phantom@lvm_fanout_cb3387f7a7684b1c9ee5f2f88325b7d5
    8. cinder-volume_fanout_9017a1a7f4b44867983dcddfb56531a2

```
[root@phantom ~]# rabbitmqctl list_exchanges name
Listing exchanges ...

amq.direct
amq.fanout
amq.headers
amq.match
amq.rabbitmq.log
amq.rabbitmq.trace
amq.topic
cinder-scheduler_fanout
cinder-volume.phantom@lvm_fanout
cinder-volume_fanout
openstack
...done.
[root@phantom ~]# rabbitmqctl list_queues name
Listing queues ...
cinder-scheduler
cinder-scheduler.phantom
cinder-scheduler_fanout_572c35c0fbf94560b4c49572d5868ea5
cinder-volume
cinder-volume.phantom@lvm
cinder-volume.phantom@lvm.phantom
cinder-volume.phantom@lvm_fanout_cb3387f7a7684b1c9ee5f2f88325b7d5
cinder-volume_fanout_9017a1a7f4b44867983dcddfb56531a2
...done.
```

RabbitMQ Gotchas

Cinder uses Kombu to connect to the RabbitMQ environment. Kombu is a Python library that in turn uses AMQPLib, a library that implements the standard AMQP 0.8 at the time of writing. When using Kombu, Invokers and Workers need the following parameters in order to instantiate a Connection object that connects to the RabbitMQ server (please note that most of the following material can be also found in the Kombu documentation; it has been summarized and revised here for sake of clarity):

- Hostname: The hostname to the AMQP server.
- Userid: A valid username used to authenticate to the server.
- Password: The password used to authenticate to the server.
- Virtual_host: The name of the virtual host to work with. This virtual host must exist on the server, and the user must have access to it. Default is /.
- Port: The port of the AMQP server. Default is 5672 (amqp).

The following parameters are default:

- Insist: insist on connecting to a server. In a configuration with multiple load-sharing servers, the Insist option tells the server that the client is insisting on a connection to the specified server. Default is False.
- Connect_timeout: the timeout in seconds before the client gives up connecting to the server. The default is no timeout.
- SSL: use SSL to connect to the server. The default is False.

More precisely Consumers need the following parameters:

- Connection: the above mentioned Connection object.
- Queue: name of the queue.
- Exchange: name of the exchange the queue binds to.

- Routing_key: the interpretation of the routing key depends on the value of the exchange_type attribute.
  - Direct exchange: if the routing key property of the message and the routing_key attribute of the queue are identical, then the message is forwarded to the queue.
  - Fanout exchange: messages are forwarded to the queues bound the exchange, even if the binding does not have a key.
  - Topic exchange: if the routing key property of the message matches the routing key of the key according to a primitive pattern matching scheme, then the message is forwarded to the queue. The message routing key then consists of words separated by dots (., like domain names), and two special characters are available; star (*) and hash (#). The star matches any word, and the hash matches zero or more words. For example .stock.# matches the routing keys usd.stock and eur.stock.db but not stock.nasdaq.
- Durable: this flag determines the durability of both exchanges and queues; durable exchanges and queues remain active when a RabbitMQ server restarts. Non-durable exchanges/queues (transient exchanges/queues) are purged when a server restarts. It is worth noting that AMQP specifies that durable queues cannot bind to transient exchanges. Default is True.
- Auto_delete: if set, the exchange is deleted when all queues have finished using it. Default is False.
- Exclusive: exclusive queues (such as non-shared) may only be consumed from by the current connection. When exclusive is on, this also implies auto_delete. Default is False.
- Exchange_type: AMQP defines several default exchange types (routing algorithms) that covers most of the common messaging use cases.
- Auto_ack: acknowledgement is handled automatically once messages are received. By default auto_ack is set to False, and the receiver is required to manually handle acknowledgment.
- No_ack: it disable acknowledgement on the server-side. This is different from auto_ack in that acknowledgement is turned off altogether. This functionality increases performance but at the cost of reliability. Messages can get lost if a client dies before it can deliver them to the application.
- Auto_declare: if this is True and the exchange name is set, the exchange will be automatically declared at instantiation. Auto declare is on by default. Publishers specify most the parameters of Consumers (such as they do not specify a queue name), but they can also specify the following:
- Delivery_mode: the default delivery mode used for messages. The value is an integer. The following delivery modes are supported by RabbitMQ:
  - 1 or transient: the message is transient. Which means it is stored in memory only, and is lost if the server dies or restarts.
  - 2 or persistent: the message is persistent. Which means the message is stored both in-memory, and on disk, and therefore preserved if the server dies or restarts.

The default value is 2 (persistent). During a send operation, Publishers can override the delivery mode of messages so that, for example, transient messages can be sent over a durable queue.

Other Resources

Project hosting with Launchpad

Launchpad hosts the Cinder project. The Cinder project homepage on Launchpad is https://launchpad.net/cinder.

Launchpad credentials

Creating a login on Launchpad is important even if you dont use the Launchpad site itself, since Launchpad credentials are used for logging in on several OpenStack-related sites. These sites include:

- Wiki
- Gerrit (see *Code Reviews*)
- Zuul (see *Continuous Integration with Zuul*)

Mailing list

The mailing list email is `openstack-discuss@lists.openstack.org`. This is a common mailing list across the OpenStack projects. To participate in the mailing list:

1. Subscribe to the list at https://lists.openstack.org/cgi-bin/mailman/listinfo/openstack-discuss

The mailing list archives are at https://lists.openstack.org/pipermail/openstack-discuss/.

Bug tracking

Report Cinder bugs at https://bugs.launchpad.net/cinder

Feature requests (Blueprints)

Cinder uses Launchpad Blueprints to track feature requests. Blueprints are at https://blueprints.launchpad.net/cinder.

Technical support (Answers)

Cinder no longer uses Launchpad Answers to track Cinder technical support questions.

Note that Ask OpenStack (which is not hosted on Launchpad) can be used for technical support requests.

Code Reviews

Cinder follows the same Review guidelines outlined by the OpenStack community. This page provides additional information that is helpful for reviewers of patches to Cinder.

Gerrit

Cinder uses the Gerrit tool to review proposed code changes. The review site is https://review.opendev.org

Gerrit is a complete replacement for Github pull requests. *All Github pull requests to the Cinder repository will be ignored.*

See Quick Reference for information on quick reference for developers. See Getting Started for information on how to get started using Gerrit. See Development Workflow for more detailed information on how to work with Gerrit.

The Great Change

With the demise of Python 2.7 in January 2020, beginning with the Ussuri development cycle, Cinder only needs to support Python 3 runtimes (in particular, 3.6 and 3.7). Thus we can begin to incorporate Python 3 language features and remove Python 2 compatibility code. At the same time, however, we are still supporting stable branches that must support Python 2. Our biggest interaction with the stable branches is backporting bugfixes, where in the ideal case, were just doing a simple cherry-pick of a commit from master to the stable branches. You can see that theres some tension here.

With that in mind, here are some guidelines for reviewers and developers that the Cinder community has agreed on during this phase where we want to write pure Python 3 but still must support Python 2 code.

Python 2 to Python 3 transition guidelines

- We need to be checking the code coverage of test cases very carefully so that new code has excellent coverage. The idea is that we want these tests to fail when a backport is proposed to a stable branch and the tests are run under Python 2 (if the code is using any Python-3-only language features).
- New features can use Python-3-only language constructs, but bugfixes likely to be backported should be more conservative and write for Python 2 compatibilty.
- The code for drivers may continue to use the six compatibility library at their discretion.
- We will not remove six from mainline Cinder code that impacts the drivers (for example, classes they inherit from).
- We can remove six from code that doesnt impact drivers, keeping in mind that backports may be more problematic, and hence making sure that we have really good test coverage.

Targeting Milestones

In an effort to guide team review priorities the Cinder team has adopted the process of adding comments to reviews to target a milestone for a particular patch. This process is not required for all patches but is beneficial for patches that may be time sensitive. For example patches that need to land earlier in the release cycle so as to get additional test time or because later development activities are dependent upon that functionality merging.

To target a patch to a milestone a reviewer should add a comment using the following format:

```
target-<release>-<milestone>
```

Release should be used to indicate the release to which the patch should be targeted, all lower case. The milestone is a single number, 1 to 3, indicating the milestone number. So, to target a patch to land in Milestone 2 of the Rocky release a comment like the following would be added:

```
target-rocky-2
```

Adding this tag allows reviewers to search for these tags and use them as a guide in review priorities.

Targeting patches should be done by Cinder Core Review Team members. If a patch developer feels that a patch should be targeted to a milestone the developer should bring the request up to the Cinder team in a weekly meeting or on the `#openstack-cinder` IRC channel.

Reviewing Vendor Patches

It is important to consider, when reviewing patches to a vendors Cinder driver, whether the patch passes the vendors CI process. CI reports are the only tool we have to ensure that a patch works with the Vendors driver. A patch to a vendors driver that does not pass that vendors CI should not be merged. If a patch is submitted by a person that does not work with the vendor that owns the driver, a +1 review from someone at that vendor is also required. Finally, a patch should not be merged before the Vendors CI has run against the patch.

Note

Patches which have passed vendor CI and have merged in master are exempt from this requirement upon backport to stable and/or driverfixes branches as vendors are not required to run CI on those branches. If the vendor, however, is running CI on stable and/or driverfix branches failures should not be ignored unless otherwise verified by a developer from the vendor.

Unit Tests

Cinder requires unit tests with all patches that introduce a new branch or function in the code. Changes that do not come with a unit test change should be considered closely and usually returned to the submitter with a request for the addition of unit test.

Note

Unit test changes are not validated in any way by vendors CI. Vendor CIs run the tempest volume tests against a change which does not include a unit test execution.

CI Job rechecks

CI job runs may result in false negatives for a considerable number of causes:

- Network failures.
- Not enough resources on the job runner.
- Storage timeouts caused by the array running nightly maintenance jobs.
- External service failure: pypi, package repositories, etc.
- Non cinder components spurious bugs.

And the list goes on and on.

When we detect one of these cases the normal procedure is to run a recheck writing a comment with `recheck` for core Zuul jobs, or the specific third party CI recheck command, for example `run-DellEMC PowerStore CI`.

These false negative have periods of time where they spike, for example when there are spurious failures, and a lot of rechecks are necessary until a valid result is posted by the CI job. And its in these periods of time where people acquire the tendency to blindly issue rechecks without looking at the errors reported by the jobs.

When these blind checks happen on real patch failures or with external services that are going to be out for a while, they lead to wasted resources as well as longer result times for patches in other projects.

The Cinder community has noticed this tendency and wants to fix it, so now it is strongly encouraged to avoid issuing naked rechecks and instead issue them with additional information to indicate that we have looked at the failure and confirmed it is unrelated to the patch.

Here are some real examples of proper rechecks:

- Spurious issue in other component: `recheck tempest-integrated-storage : intermittent failure nova bug #1836754`

- Deployment issue on the job: `recheck cinder-plugin-ceph-tempest timed out, errors all over the place`
- External service failure: `Third party recheck grenade : Failed to retrieve .deb packages`

Another common case for blindly rechecking a patch is when it is only changing a specific driver but there are failures on jobs that dont use that driver. In such cases we still have to look at the failures, because they can be failures that are going to take a while to fix, and issuing a recheck will be futile at that time and we should wait for a couple of hours, or maybe even a day, before issuing a recheck that can yield the desired result.

Efficient Review Guidelines

This section will guide you through the best practices you can follow to do quality code reviews:

- **Failing Gate**: You can check for jobs like pep8, py36, py38, functional etc that are generic to all the patches and look for possible failures in linting, unit test, functional test etc and provide feedback on fixing it. Usually its the authors responsibility to do a local run of tox and ensure they dont fail upstream but if something is failing on gate and the author is not be aware about how to fix it then we can provide valuable guidance on it. There are also jobs specific to particular area of code (for example, `cinder-plugin-ceph-tempest` for the RBD volume driver, `devstack-plugin-nfs-tempest-full` for the generic NFS driver etc) so look for issues in the jobs if they are related to the code changes proposed. There is a past example on why we should check these jobs, the `devstack-plugin-nfs-tempest-full` is a non-voting job and was failing on one of the FS drivers related patch which got merged and started failing the `NetApp CI` blocking the netapp features during that time.
- **Documentation**: Check whether the patch proposed requires documentation or not and ensure the proper documentation is added. If the proper documentation is added then the next step is to check the status of docs job if its failing or passing. If it passes, you can check how it looks in HTML as follows: Go to `openstack-tox-docs job` link -> `View Log` -> `docs` and go to the appropriate section for which the documentation is added. Rendering: We do have a job for checking failures related to document changes proposed (openstack-tox-docs) but we need to be aware that even if a document change passes all the syntactical rules, it still might not be logically correct i.e. after rendering it could be possible that the bullet points are not under the desired section or the spacing and indentation is not as desired. It is always good to check the final document after rendering in the docs job which might yield possible logical errors.
- **Readability**: In a large codebase (like Cinder), Readability is a big factor as remembering the logic of every code path is not feasible and contributors change from time to time. We should adapt to writing readable code which is easy to follow and can be understood by anyone having knowledge about Python constructs and working of Cinder. Sometimes it happens that a logic can only be written in a complex way, in that case, its always good practice to add a comment describing the functionality. So, if a logic proposed is not readable, do ask/suggest a more readable version of it and if thats not feasible then asking for a comment that would explain it is also a valid review point.
- **Type Annotations**: There has been an ongoing effort to implement type annotations all across Cinder with the help of mypy tooling. Certain areas of code already adapt to mypy coding style and its good practice that new code merging into Cinder should also adapt to it. We, as reviewers, should ensure that new code proposed should include mypy constructs.
- **Microversions**: Cinder uses the microversion framework for implementing new feature that causes a change in the API behavior (request/response) while maintaining backward compatibility at the

same time. There have been examples in the past where a patch adding a new microversion misses file(s) where the microversion changes are necessary so its a good practice for the author and reviewer to ensure that all files associated with a microversion change should be updated. You can find the list of files and changes required in our Microversion Doc.

- **Downvoting reason**: It often happens that the reviewer adds a bunch of comments some of which they would like to be addressed (blocking) and some of them are good to have but not a hard requirement (non-blocking). Its a good practice for the reviewer to mention for which comments is the -1 valid so to make sure they are always addressed.
- **Testing**: Always check if the patch adds the associated unit, functional and tempest tests depending on the change.
- **Commit Message**: There are few things that we should make sure the commit message includes:

  1) Make sure the author clearly explains in the commit message why the code changes are necessary and how exactly the code changes fix the issue.

  2) It should have the appropriate tags (Eg: Closes-Bug, Related-Bug, Blueprint, Depends-On etc). For detailed information refer to external references in commit message.

  3) It should follow the guidelines of commit message length i.e. 50 characters for the summary line and 72 characters for the description. More information can be found at Summary of Git commit message structure.

  4) Sometimes it happens that the author updates the code but forgets to update the commit message leaving the commit describing the old changes. Verify that the commit message is updated as per code changes.
- **Release Notes**: There are different cases where a releasenote is required like fixing a bug, adding a feature, changing areas affecting upgrade etc. You can refer to the Release notes section in our contributor docs for more information.
- **Ways of reviewing**: There are various ways you can go about reviewing a patch, following are some of the standard ways you can follow to provide valuable feedback on the patch:

  1) Testing it in local environment: The easiest way to check the correctness of a code change proposed is to reproduce the issue (steps should be in launchpad bug) and try the same steps after applying the patch to your environment and see if the provided code changes fix the issue. You can also go a little further to think of possible corner cases where an end user might possibly face issues again and provide the same feedback to cover those cases in the original change proposed.

  2) Optimization: If youre not aware about the code path the patch is fixing, you can still go ahead and provide valuable feedback about the python code if that can be optimized to improve maintainability or performance.

  3) Perform Dry Run: Sometimes the code changes are on code paths that we dont have or cant create environment for (like vendor driver changes or optional service changes like cinder-backup) so we can read through the code or use some example values to perform a dry run of the code and see if it fails in that scenario.

Continuous Integration with Zuul

Cinder uses Zuul as project gating system. The Zuul web front-end is at https://status.opendev.org.

Zuul ensures that only tested code gets merged. The configuration is mainly done in cinders .zuul.yaml file.

The following is a partial list of jobs that are configured to run on changes. Test jobs run initially on proposed changes and get run again after review and approval. Note that for each job run the code gets rebased to current HEAD to test exactly the state that gets merged.

openstack-tox-pep8
Run linters like PEP8 checks.

openstack-tox-pylint
Run Pylint checks.

openstack-tox-python27
Run unit tests using python2.7.

openstack-tox-python36
Run unit tests using python3.6.

openstack-tox-docs
Build this documentation for review.

The following jobs are some of the jobs that run after a change is merged:

publish-openstack-tox-docs
Build this documentation and publish to OpenStack Cinder.

publish-openstack-python-branch-tarball
Do `python setup.py sdist` to create a tarball of the cinder code and upload it to http://tarballs.openstack.org/cinder.

cinder

cinder package

Subpackages

cinder.api package

Subpackages

cinder.api.contrib package

Submodules

cinder.api.contrib.admin_actions module

class AdminController(**args*, ***kwargs*)

Bases: `Controller`

Abstract base class for AdminControllers.

authorize(*context*, *action_name*, *target_obj=None*)

collection = None

validate_update(*req*, *body*)

wsgi_actions = {'os-force_delete': '_force_delete', 'os-reset_status': '_reset_status'}

wsgi_extensions = []

class Admin_actions(*ext_mgr*)

Bases: *ExtensionDescriptor*

Enable admin actions.

alias = 'os-admin-actions'

get_controller_extensions()

List of extensions.ControllerExtension extension objects.

Controller extensions are used to extend existing controllers.

name = 'AdminActions'

updated = '2012-08-25T00:00:00+00:00'

class BackupAdminController(**args*, ***kwargs*)

Bases: *AdminController*

AdminController for Backups.

collection = 'backups'

wsgi_actions = {'os-force_delete': '_force_delete', 'os-reset_status': '_reset_status'}

wsgi_extensions = []

class SnapshotAdminController(**args*, ***kwargs*)

Bases: *AdminController*

AdminController for Snapshots.

collection = 'snapshots'

validate_update(*req*, *body*)

wsgi_actions = {'os-force_delete': '_force_delete', 'os-reset_status': '_reset_status'}

wsgi_extensions = []

class VolumeAdminController(**args*, ***kwargs*)

Bases: *AdminController*

AdminController for Volumes.

collection = 'volumes'

validate_update(*req*, *body*)

wsgi_actions = {'os-extend_volume_completion': '_extend_volume_completion', 'os-force_delete': '_force_delete', 'os-force_detach': '_force_detach', 'os-migrate_volume': '_migrate_volume', 'os-migrate_volume_completion': '_migrate_volume_completion', 'os-reset_status': '_reset_status'}

wsgi_extensions = []

cinder.api.contrib.availability_zones module

class Availability_zones(*ext_mgr*)

Bases: *ExtensionDescriptor*

Describe Availability Zones.

alias = 'os-availability-zone'

get_resources()

List of extensions.ResourceExtension extension objects.

Resources define new nouns, and are accessible through URLs.

name = 'AvailabilityZones'

updated = '2013-06-27T00:00:00+00:00'

class Controller(**args*, ***kwargs*)

Bases: *Controller*

index(*req*)

Describe all known availability zones.

wsgi_actions = {}

wsgi_extensions = []

cinder.api.contrib.backups module

The backups api.

class Backups(*ext_mgr*)

Bases: *ExtensionDescriptor*

Backups support.

alias = 'backups'

get_resources()

List of extensions.ResourceExtension extension objects.

Resources define new nouns, and are accessible through URLs.

name = 'Backups'

updated = '2012-12-12T00:00:00+00:00'

class BackupsController

Bases: *Controller*

The Backups API controller for the OpenStack API.

create(*req*, *body*)

Create a new backup.

delete(*req*, *id*)

Delete a backup.

`detail(req)`
Returns a detailed list of backups.

`export_record(req, id)`
Export a backup.

`import_record(req, body)`
Import a backup.

`index(req)`
Returns a summary list of backups.

`restore(req, id, body)`
Restore an existing backup to a volume.

`show(req, id)`
Return data about the given backup.

`wsgi_actions = {}`

`wsgi_extensions = []`

cinder.api.contrib.capabilities module

`class Capabilities(ext_mgr)`
Bases: `ExtensionDescriptor`

Capabilities support.

`alias = 'capabilities'`

`get_resources()`
List of extensions.ResourceExtension extension objects.

Resources define new nouns, and are accessible through URLs.

`name = 'Capabilities'`

`updated = '2015-08-31T00:00:00+00:00'`

`class CapabilitiesController`
Bases: `Controller`

The Capabilities controller for the OpenStack API.

`show(req, id)`
Return capabilities list of given backend.

`wsgi_actions = {}`

`wsgi_extensions = []`

cinder.api.contrib.cgsnapshots module

The cgsnapshots api.

class Cgsnapshots(*ext_mgr*)

Bases: `ExtensionDescriptor`

cgsnapshots support.

alias = 'cgsnapshots'

get_resources()

List of extensions.ResourceExtension extension objects.

Resources define new nouns, and are accessible through URLs.

name = 'Cgsnapshots'

updated = '2014-08-18T00:00:00+00:00'

class CgsnapshotsController

Bases: `Controller`

The cgsnapshots API controller for the OpenStack API.

create(*req*, *body*)

Create a new cgsnapshot.

delete(*req*, *id*)

Delete a cgsnapshot.

detail(*req*)

Returns a detailed list of cgsnapshots.

index(*req*)

Returns a summary list of cgsnapshots.

show(*req*, *id*)

Return data about the given cgsnapshot.

wsgi_actions = {}

wsgi_extensions = []

cinder.api.contrib.consistencygroups module

The consistencygroups api.

class ConsistencyGroupsController

Bases: `Controller`

The ConsistencyGroups API controller for the OpenStack API.

create(*req*, *body*)

Create a new consistency group.

create_from_src(*req*, *body*)

Create a new consistency group from a source.

The source can be a CG snapshot or a CG. Note that this does not require volume_types as the create API above.

delete(*req*, *id*, *body*)

Delete a consistency group.

detail(*req*)

Returns a detailed list of consistency groups.

index(*req*)

Returns a summary list of consistency groups.

show(*req*, *id*)

Return data about the given consistency group.

update(*req*, *id*, *body*)

Update the consistency group.

Expected format of the input parameter body:

```
{
    "consistencygroup":
    {
        "name": "my_cg",
        "description": "My consistency group",
        "add_volumes": "volume-uuid-1,volume-uuid-2,...",
        "remove_volumes": "volume-uuid-8,volume-uuid-9,..."
    }
}
```

wsgi_actions = {}

wsgi_extensions = []

class **Consistencygroups**(*ext_mgr*)

Bases: `ExtensionDescriptor`

consistency groups support.

alias = 'consistencygroups'

get_resources()

List of extensions.ResourceExtension extension objects.

Resources define new nouns, and are accessible through URLs.

name = 'Consistencygroups'

updated = '2014-08-18T00:00:00+00:00'

cinder.api.contrib.extended_services module

class **Extended_services**(*ext_mgr*)

Bases: `ExtensionDescriptor`

Extended services support.

alias = 'os-extended-services'

name = 'ExtendedServices'

updated = '2014-01-10T00:00:00-00:00'

cinder.api.contrib.extended_snapshot_attributes module

The Extended Snapshot Attributes API extension.

class ExtendedSnapshotAttributesController(*view_builder=None*)

Bases: Controller

detail(*req*, *resp_obj*)

show(*req*, *resp_obj*, *id*)

wsgi_actions = {}

wsgi_extensions = [('show', None), ('detail', None)]

class Extended_snapshot_attributes(*ext_mgr*)

Bases: *ExtensionDescriptor*

Extended SnapshotAttributes support.

alias = 'os-extended-snapshot-attributes'

get_controller_extensions()

List of extensions.ControllerExtension extension objects.

Controller extensions are used to extend existing controllers.

name = 'ExtendedSnapshotAttributes'

updated = '2012-06-19T00:00:00+00:00'

cinder.api.contrib.hosts module

The hosts admin extension.

class HostController

Bases: Controller

The Hosts API controller for the OpenStack API.

index(*req*)

show(*req*, *id*)

Shows the volume usage info given by hosts.

Parameters

- **req** security context
- **id** hostname

Returns

dict the host resources dictionary. ex.:

```
{'host': [{'resource': D},..]}
D: {'host': 'hostname','project': 'admin',
    'volume_count': 1, 'total_volume_gb': 2048}
```

update(*req*, *id*, *service=None*, **args*, ***kwargs*)

wsgi_actions = {}

wsgi_extensions = []

class Hosts(*ext_mgr*)

Bases: `ExtensionDescriptor`

Admin-only host administration.

alias = 'os-hosts'

get_resources()

List of extensions.ResourceExtension extension objects.

Resources define new nouns, and are accessible through URLs.

name = 'Hosts'

updated = '2011-06-29T00:00:00+00:00'

check_host(*fn*)

Makes sure that the host exists.

cinder.api.contrib.qos_specs_manage module

The QoS specs extension

class QoSSpecsController(*view_builder=None*)

Bases: `Controller`

The volume type extra specs API controller for the OpenStack API.

associate(*req*, *id*)

Associate a qos specs with a volume type.

associations(*req*, *id*)

List all associations of given qos specs.

create(*req*, *body=None*)

delete(*req*, *id*)

Deletes an existing qos specs.

delete_keys(*req*, *id*, *body*)

Deletes specified keys in qos specs.

disassociate(*req*, *id*)

Disassociate a qos specs from a volume type.

disassociate_all(*req*, *id*)

Disassociate a qos specs from all volume types.

index(*req*)

Returns the list of qos_specs.

show(*req*, *id*)

Return a single qos spec item.

update(*req*, *id*, *body=None*)

wsgi_actions = {}

wsgi_extensions = []

class **Qos_specs_manage**(*ext_mgr*)

Bases: *ExtensionDescriptor*

QoS specs support.

alias = 'qos-specs'

get_resources()

List of extensions.ResourceExtension extension objects.

Resources define new nouns, and are accessible through URLs.

name = 'Qos_specs_manage'

updated = '2013-08-02T00:00:00+00:00'

cinder.api.contrib.quota_classes module

class **QuotaClassSetsController**(*view_builder=None*)

Bases: *Controller*

show(*req*, *id*)

update(*req*, *id*, *body*)

wsgi_actions = {}

wsgi_extensions = []

class **Quota_classes**(*ext_mgr*)

Bases: *ExtensionDescriptor*

Quota classes management support.

alias = 'os-quota-class-sets'

get_resources()

List of extensions.ResourceExtension extension objects.

Resources define new nouns, and are accessible through URLs.

name = 'QuotaClasses'

updated = '2012-03-12T00:00:00+00:00'

cinder.api.contrib.quotas module

class QuotaSetsController(*view_builder=None*)

Bases: *Controller*

defaults(*req*, *id*)

delete(*req*, *id*)

Delete Quota for a particular tenant.

Parameters

- **req** request
- **id** target project id that needs to be deleted

show(*req*, *id*)

Show quota for a particular tenant

Parameters

- **req** request
- **id** target project id that needs to be shown

update(*req*, *id*, *body*)

Update Quota for a particular tenant

Parameters

- **req** request
- **id** target project id that needs to be updated
- **body** key, value pair that will be applied to the resources if the update succeeds

wsgi_actions = {}

wsgi_extensions = []

class Quotas(*ext_mgr*)

Bases: *ExtensionDescriptor*

Quota management support.

alias = 'os-quota-sets'

get_resources()

List of extensions.ResourceExtension extension objects.

Resources define new nouns, and are accessible through URLs.

name = 'Quotas'

updated = '2011-08-08T00:00:00+00:00'

cinder.api.contrib.resource_common_manage module

get_manageable_resources(*req*, *is_detail*, *function_get_manageable*, *view_builder*)

cinder.api.contrib.scheduler_hints module

class **Scheduler_hints**(*ext_mgr*)

Bases: `ExtensionDescriptor`

Pass arbitrary key/value pairs to the scheduler.

alias = 'OS-SCH-HNT'

name = 'SchedulerHints'

updated = '2013-04-18T00:00:00+00:00'

create(*req*, *body*)

cinder.api.contrib.scheduler_stats module

The Scheduler Stats extension

class **SchedulerStatsController**

Bases: `Controller`

The Scheduler Stats controller for the OpenStack API.

get_pools(*req*)

List all active pools in scheduler.

wsgi_actions = {}

wsgi_extensions = []

class **Scheduler_stats**(*ext_mgr*)

Bases: `ExtensionDescriptor`

Scheduler stats support.

alias = 'scheduler-stats'

get_resources()

List of extensions.ResourceExtension extension objects.

Resources define new nouns, and are accessible through URLs.

name = 'Scheduler_stats'

updated = '2014-09-07T00:00:00+00:00'

cinder.api.contrib.services module

class **ServiceController**(*ext_mgr=None*)

Bases: `Controller`

index(*req*)

Return a list of all running services.

Filter by host & service name.

update(*req*, *id*, *body*)

Enable/Disable scheduling for a service.

Includes Freeze/Thaw which sends call down to drivers and allows volume.manager for the specified host to disable the service rather than accessing the service directly in this API layer.

wsgi_actions = {}

wsgi_extensions = []

class Services(*ext_mgr*)

Bases: `ExtensionDescriptor`

Services support.

alias = 'os-services'

get_resources()

List of extensions.ResourceExtension extension objects.

Resources define new nouns, and are accessible through URLs.

name = 'Services'

updated = '2012-10-28T00:00:00-00:00'

cinder.api.contrib.snapshot_actions module

class SnapshotActionsController(**args*, ***kwargs*)

Bases: `Controller`

wsgi_actions = {'os-update_snapshot_status': '_update_snapshot_status'}

wsgi_extensions = []

class Snapshot_actions(*ext_mgr*)

Bases: `ExtensionDescriptor`

Enable snapshot manager actions.

alias = 'os-snapshot-actions'

get_controller_extensions()

List of extensions.ControllerExtension extension objects.

Controller extensions are used to extend existing controllers.

name = 'SnapshotActions'

updated = '2013-07-16T00:00:00+00:00'

cinder.api.contrib.snapshot_manage module

class SnapshotManageController(**args*, ***kwargs*)

Bases: `Controller`

The /os-snapshot-manage controller for the OpenStack API.

create(*req*, *body*)

Instruct Cinder to manage a storage snapshot object.

Manages an existing backend storage snapshot object (e.g. a Linux logical volume or a SAN disk) by creating the Cinder objects required to manage it, and possibly renaming the backend storage snapshot object (driver dependent).

From an API perspective, this operation behaves very much like a snapshot creation operation.

Required HTTP Body:

```
{
  "snapshot":
  {
    "volume_id": "<Cinder volume already exists in volume backend>",
    "ref":
       "<Driver-specific reference to the existing storage object>"
  }
}
```

See the appropriate Cinder drivers implementations of the manage_snapshot method to find out the accepted format of ref. For example,in LVM driver, it will be the logic volume name of snapshot which you want to manage.

This API call will return with an error if any of the above elements are missing from the request, or if the volume_id element refers to a cinder volume that could not be found.

The snapshot will later enter the error state if it is discovered that ref is bad.

Optional elements to snapshot are:

```
name            A name for the new snapshot.
description     A description for the new snapshot.
metadata        Key/value pairs to be associated with the new␣
↪snapshot.
```

detail(*req*)

Returns a detailed list of snapshots available to manage.

index(*req*)

Returns a summary list of snapshots available to manage.

wsgi_actions = {}

wsgi_extensions = [('index', None), ('detail', None)]

class Snapshot_manage(*ext_mgr*)

Bases: `ExtensionDescriptor`

Allows existing backend storage to be managed by Cinder.

`alias = 'os-snapshot-manage'`

`get_resources()`

List of extensions.ResourceExtension extension objects.

Resources define new nouns, and are accessible through URLs.

`name = 'SnapshotManage'`

`updated = '2014-12-31T00:00:00+00:00'`

cinder.api.contrib.snapshot_unmanage module

`class SnapshotUnmanageController(*args, **kwargs)`

Bases: `Controller`

`unmanage(req, id, body)`

Stop managing a snapshot.

This action is very much like a delete, except that a different method (unmanage) is called on the Cinder driver. This has the effect of removing the snapshot from Cinder management without actually removing the backend storage object associated with it.

There are no required parameters.

A Not Found error is returned if the specified snapshot does not exist.

`wsgi_actions = {'os-unmanage': 'unmanage'}`

`wsgi_extensions = []`

`class Snapshot_unmanage(ext_mgr)`

Bases: `ExtensionDescriptor`

Enable volume unmanage operation.

`alias = 'os-snapshot-unmanage'`

`get_controller_extensions()`

List of extensions.ControllerExtension extension objects.

Controller extensions are used to extend existing controllers.

`name = 'SnapshotUnmanage'`

`updated = '2014-12-31T00:00:00+00:00'`

cinder.api.contrib.types_extra_specs module

The volume types extra specs extension

`class Types_extra_specs(ext_mgr)`

Bases: `ExtensionDescriptor`

Type extra specs support.

alias = 'os-types-extra-specs'

get_resources()

List of extensions.ResourceExtension extension objects.

Resources define new nouns, and are accessible through URLs.

name = 'TypesExtraSpecs'

updated = '2011-08-24T00:00:00+00:00'

class VolumeTypeExtraSpecsController(*view_builder=None*)

Bases: `Controller`

The volume type extra specs API controller for the OpenStack API.

create(*req*, *type_id*, *body*)

delete(*req*, *type_id*, *id*)

Deletes an existing extra spec.

index(*req*, *type_id*)

Returns the list of extra specs for a given volume type.

show(*req*, *type_id*, *id*)

Return a single extra spec item.

update(*req*, *type_id*, *id*, *body*)

wsgi_actions = {}

wsgi_extensions = []

cinder.api.contrib.types_manage module

The volume types manage extension.

class Types_manage(*ext_mgr*)

Bases: `ExtensionDescriptor`

Types manage support.

alias = 'os-types-manage'

get_controller_extensions()

List of extensions.ControllerExtension extension objects.

Controller extensions are used to extend existing controllers.

name = 'TypesManage'

updated = '2011-08-24T00:00:00+00:00'

class VolumeTypesManageController(*view_builder=None*)

Bases: `Controller`

The volume types API controller for the OpenStack API.

wsgi_actions = {'create': '_create', 'delete': '_delete', 'update': '_update'}

wsgi_extensions = []

cinder.api.contrib.used_limits module

class UsedLimitsController(*view_builder=None*)

Bases: Controller

index(*req*, *resp_obj*)

wsgi_actions = {}

wsgi_extensions = [('index', None)]

class Used_limits(*ext_mgr*)

Bases: ExtensionDescriptor

Provide data on limited resources that are being used.

alias = 'os-used-limits'

get_controller_extensions()

List of extensions.ControllerExtension extension objects.

Controller extensions are used to extend existing controllers.

name = 'UsedLimits'

updated = '2013-10-03T00:00:00+00:00'

cinder.api.contrib.volume_actions module

class VolumeActionsController(**args*, ***kwargs*)

Bases: Controller

versioned_methods = {'_reimage': [<cinder.api.openstack.versioned_method.VersionedMethod object>]}

wsgi_actions = {'os-attach': '_attach', 'os-begin_detaching': '_begin_detaching', 'os-detach': '_detach', 'os-extend': '_extend', 'os-initialize_connection': '_initialize_connection', 'os-reimage': '_reimage', 'os-reserve': '_reserve', 'os-retype': '_retype', 'os-roll_detaching': '_roll_detaching', 'os-set_bootable': '_set_bootable', 'os-terminate_connection': '_terminate_connection', 'os-unreserve': '_unreserve', 'os-update_readonly_flag': '_volume_readonly_update', 'os-volume_upload_image': '_volume_upload_image'}

wsgi_extensions = []

class Volume_actions(*ext_mgr*)

Bases: ExtensionDescriptor

Enable volume actions.

`alias = 'os-volume-actions'`

`get_controller_extensions()`

List of extensions.ControllerExtension extension objects.

Controller extensions are used to extend existing controllers.

`name = 'VolumeActions'`

`updated = '2012-05-31T00:00:00+00:00'`

cinder.api.contrib.volume_encryption_metadata module

The volume encryption metadata extension.

`class VolumeEncryptionMetadataController(`*view_builder=None*`)`

Bases: `Controller`

The volume encryption metadata API extension.

`index(`*req, volume_id*`)`

Returns the encryption metadata for a given volume.

`show(`*req, volume_id, id*`)`

Return a single encryption item.

`wsgi_actions = {}`

`wsgi_extensions = []`

`class Volume_encryption_metadata(`*ext_mgr*`)`

Bases: `ExtensionDescriptor`

Volume encryption metadata retrieval support.

`alias = 'os-volume-encryption-metadata'`

`get_resources()`

List of extensions.ResourceExtension extension objects.

Resources define new nouns, and are accessible through URLs.

`name = 'VolumeEncryptionMetadata'`

`updated = '2013-07-10T00:00:00+00:00'`

cinder.api.contrib.volume_host_attribute module

`class VolumeHostAttributeController(`*view_builder=None*`)`

Bases: `Controller`

`detail(`*req, resp_obj*`)`

`show(`*req, resp_obj, id*`)`

`wsgi_actions = {}`

wsgi_extensions = [('show', None), ('detail', None)]

class Volume_host_attribute(*ext_mgr*)

Bases: ExtensionDescriptor

Expose host as an attribute of a volume.

alias = 'os-vol-host-attr'

get_controller_extensions()

List of extensions.ControllerExtension extension objects.

Controller extensions are used to extend existing controllers.

name = 'VolumeHostAttribute'

updated = '2011-11-03T00:00:00+00:00'

cinder.api.contrib.volume_image_metadata module

The Volume Image Metadata API extension.

class VolumeImageMetadataController(**args*, ***kwargs*)

Bases: Controller

create(*req*, *id*, *body*)

delete(*req*, *id*, *body*)

Deletes an existing image metadata.

detail(*req*, *resp_obj*)

index(*req*, *id*, *body*)

show(*req*, *resp_obj*, *id*)

wsgi_actions = {'os-set_image_metadata': 'create', 'os-show_image_metadata': 'index', 'os-unset_image_metadata': 'delete'}

wsgi_extensions = [('show', None), ('detail', None)]

class Volume_image_metadata(*ext_mgr*)

Bases: ExtensionDescriptor

Show image metadata associated with the volume.

alias = 'os-vol-image-meta'

get_controller_extensions()

List of extensions.ControllerExtension extension objects.

Controller extensions are used to extend existing controllers.

name = 'VolumeImageMetadata'

updated = '2012-12-07T00:00:00+00:00'

cinder.api.contrib.volume_manage module

class VolumeManageController(**args*, ***kwargs*)

Bases: Controller

The /os-volume-manage controller for the OpenStack API.

create(*req*, *body*)

Instruct Cinder to manage a storage object.

Manages an existing backend storage object (e.g. a Linux logical volume or a SAN disk) by creating the Cinder objects required to manage it, and possibly renaming the backend storage object (driver dependent)

From an API perspective, this operation behaves very much like a volume creation operation, except that properties such as image, snapshot and volume references dont make sense, because we are taking an existing storage object into Cinder management.

Required HTTP Body:

```
{
  "volume": {
    "host": "<Cinder host on which the existing storage resides>",
    "cluster": "<Cinder cluster on which the storage resides>",
    "ref": "<Driver-specific reference to existing storage object>"
  }
}
```

See the appropriate Cinder drivers implementations of the manage_volume method to find out the accepted format of ref.

This API call will return with an error if any of the above elements are missing from the request, or if the host element refers to a cinder host that is not registered.

The volume will later enter the error state if it is discovered that ref is bad.

Optional elements to volume are:

```
name                 A name for the new volume.
description          A description for the new volume.
volume_type          ID or name of a volume type to associate with
                     the new Cinder volume. Does not necessarily
                     guarantee that the managed volume will have the
                     properties described in the volume_type. The
                     driver may choose to fail if it identifies that
                     the specified volume_type is not compatible with
                     the backend storage object.
metadata             Key/value pairs to be associated with the new
                     volume.
availability_zone    The availability zone to associate with the new
                     volume.
bootable             If set to True, marks the volume as bootable.
```

detail(*req*)

Returns a detailed list of volumes available to manage.

index(*req*)

Returns a summary list of volumes available to manage.

wsgi_actions = {}

wsgi_extensions = [('index', None), ('detail', None)]

class Volume_manage(*ext_mgr*)

Bases: `ExtensionDescriptor`

Allows existing backend storage to be managed by Cinder.

alias = 'os-volume-manage'

get_resources()

List of extensions.ResourceExtension extension objects.

Resources define new nouns, and are accessible through URLs.

name = 'VolumeManage'

updated = '2014-02-10T00:00:00+00:00'

cinder.api.contrib.volume_mig_status_attribute module

class VolumeMigStatusAttributeController(*view_builder=None*)

Bases: `Controller`

detail(*req, resp_obj*)

show(*req, resp_obj, id*)

wsgi_actions = {}

wsgi_extensions = [('show', None), ('detail', None)]

class Volume_mig_status_attribute(*ext_mgr*)

Bases: `ExtensionDescriptor`

Expose migration_status as an attribute of a volume.

alias = 'os-vol-mig-status-attr'

get_controller_extensions()

List of extensions.ControllerExtension extension objects.

Controller extensions are used to extend existing controllers.

name = 'VolumeMigStatusAttribute'

updated = '2013-08-08T00:00:00+00:00'

cinder.api.contrib.volume_tenant_attribute module

class VolumeTenantAttributeController(*view_builder=None*)

Bases: `Controller`

detail(*req*, *resp_obj*)

show(*req*, *resp_obj*, *id*)

wsgi_actions = {}

wsgi_extensions = [('show', None), ('detail', None)]

class Volume_tenant_attribute(*ext_mgr*)

Bases: *ExtensionDescriptor*

Expose the internal project_id as an attribute of a volume.

alias = 'os-vol-tenant-attr'

get_controller_extensions()

List of extensions.ControllerExtension extension objects.

Controller extensions are used to extend existing controllers.

name = 'VolumeTenantAttribute'

updated = '2011-11-03T00:00:00+00:00'

cinder.api.contrib.volume_transfer module

class VolumeTransferController

Bases: *Controller*

The Volume Transfer API controller for the OpenStack API.

accept(*req*, *id*, *body*)

Accept a new volume transfer.

create(*req*, *body*)

Create a new volume transfer.

delete(*req*, *id*)

Delete a transfer.

detail(*req*)

Returns a detailed list of transfers.

index(*req*)

Returns a summary list of transfers.

show(*req*, *id*)

Return data about active transfers.

wsgi_actions = {}

wsgi_extensions = []

class Volume_transfer(*ext_mgr*)

Bases: *ExtensionDescriptor*

Volume transfer management support.

alias = 'os-volume-transfer'

get_resources()

List of extensions.ResourceExtension extension objects.

Resources define new nouns, and are accessible through URLs.

name = 'VolumeTransfer'

updated = '2013-05-29T00:00:00+00:00'

cinder.api.contrib.volume_type_access module

The volume type access extension.

class VolumeTypeAccessController

Bases: `object`

The volume type access API controller for the OpenStack API.

index(*req*, *type_id*)

class VolumeTypeActionController(*view_builder=None*)

Bases: `Controller`

The volume type access API controller for the OpenStack API.

create(*req*, *body*, *resp_obj*)

detail(*req*, *resp_obj*)

index(*req*, *resp_obj*)

show(*req*, *resp_obj*, *id*)

wsgi_actions = {'addProjectAccess': '_addProjectAccess', 'removeProjectAccess': '_removeProjectAccess'}

wsgi_extensions = [('show', None), ('index', None), ('detail', None), ('create', 'create')]

class Volume_type_access(*ext_mgr*)

Bases: `ExtensionDescriptor`

Volume type access support.

alias = 'os-volume-type-access'

get_controller_extensions()

List of extensions.ControllerExtension extension objects.

Controller extensions are used to extend existing controllers.

get_resources()

List of extensions.ResourceExtension extension objects.

Resources define new nouns, and are accessible through URLs.

name = 'VolumeTypeAccess'

updated = '2014-06-26T00:00:00Z'

cinder.api.contrib.volume_type_encryption module

The volume types encryption extension.

class VolumeTypeEncryptionController(*view_builder=None*)

Bases: Controller

The volume type encryption API controller for the OpenStack API.

create(*req*, *type_id*, *body*)

Create encryption specs for an existing volume type.

delete(*req*, *type_id*, *id*)

Delete encryption specs for a given volume type.

index(*req*, *type_id*)

Returns the encryption specs for a given volume type.

show(*req*, *type_id*, *id*)

Return a single encryption item.

update(*req*, *type_id*, *id*, *body*)

Update encryption specs for a given volume type.

wsgi_actions = {}

wsgi_extensions = []

class Volume_type_encryption(*ext_mgr*)

Bases: ExtensionDescriptor

Encryption support for volume types.

alias = 'encryption'

get_controller_extensions()

List of extensions.ControllerExtension extension objects.

Controller extensions are used to extend existing controllers.

get_resources()

List of extensions.ResourceExtension extension objects.

Resources define new nouns, and are accessible through URLs.

name = 'VolumeTypeEncryption'

updated = '2013-07-01T00:00:00+00:00'

cinder.api.contrib.volume_unmanage module

class VolumeUnmanageController(**args*, ***kwargs*)

Bases: Controller

unmanage(*req*, *id*, *body*)

Stop managing a volume.

This action is very much like a delete, except that a different method (unmanage) is called on the Cinder driver. This has the effect of removing the volume from Cinder management without actually removing the backend storage object associated with it.

There are no required parameters.

A Not Found error is returned if the specified volume does not exist.

A Bad Request error is returned if the specified volume is still attached to an instance.

wsgi_actions = {'os-unmanage': 'unmanage'}

wsgi_extensions = []

class **Volume_unmanage**(*ext_mgr*)

Bases: `ExtensionDescriptor`

Enable volume unmanage operation.

alias = 'os-volume-unmanage'

get_controller_extensions()

List of extensions.ControllerExtension extension objects.

Controller extensions are used to extend existing controllers.

name = 'VolumeUnmanage'

updated = '2012-05-31T00:00:00+00:00'

Module contents

Contrib contains extensions that are shipped with cinder.

It cant be called extensions because that causes namespacing problems.

select_extensions(*ext_mgr*)

standard_extensions(*ext_mgr*)

cinder.api.middleware package

Submodules

cinder.api.middleware.auth module

Common Auth Middleware.

class **CinderKeystoneContext**(*application*)

Bases: `Middleware`

Make a request context from keystone headers.

ENV_OVERWRITES = {'X_PROJECT_DOMAIN_ID': 'project_domain_id', 'X_PROJECT_DOMAIN_NAME': 'project_domain_name', 'X_USER_DOMAIN_ID': 'user_domain_id', 'X_USER_DOMAIN_NAME': 'user_domain_name'}

class **InjectContext**(*context*, **args*, ***kwargs*)

Bases: *Middleware*

Add a cinder.context to WSGI environ.

class **NoAuthMiddleware**(*application*)

Bases: *NoAuthMiddlewareBase*

Return a fake token if one isnt specified.

Sets project_id in URLs.

class **NoAuthMiddlewareBase**(*application*)

Bases: *Middleware*

Return a fake token if one isnt specified.

base_call(*req*, *project_id_in_path=False*)

class **NoAuthMiddlewareIncludeProjectID**(*application*)

Bases: *NoAuthMiddlewareBase*

Return a fake token if one isnt specified.

Does not set project_id in URLs.

pipeline_factory(*loader*, *global_conf*, ***local_conf*)

A paste pipeline replica that keys off of auth_strategy.

cinder.api.middleware.fault module

class **FaultWrapper**(*application*)

Bases: *Middleware*

Calls down the middleware stack, making exceptions into faults.

static **status_to_type**(*status*)

cinder.api.middleware.request_id module

class **RequestId**(**args*, ***kwargs*)

Bases: RequestId

Module contents

cinder.api.openstack package

Submodules

cinder.api.openstack.api_version_request module

class APIVersionRequest(*version_string=None*, *experimental=False*)

Bases: `ComparableMixin`

This class represents an API Version Request.

This class includes convenience methods for manipulation and comparison of version numbers as needed to implement API microversions.

get_string()

Returns a string representation of this object.

If this method is used to create an APIVersionRequest, the resulting object will be an equivalent request.

matches(*min_version*, *max_version=None*, *experimental=False*)

Compares this version to the specified min/max range.

Returns whether the version object represents a version greater than or equal to the minimum version and less than or equal to the maximum version.

If min_version is null then there is no minimum limit. If max_version is null then there is no maximum limit. If self is null then raise ValueError.

Parameters

- **min_version** Minimum acceptable version.
- **max_version** Maximum acceptable version.
- **experimental** Whether to match experimental APIs.

Returns

boolean

matches_versioned_method(*method*)

Compares this version to that of a versioned method.

max_api_version()

min_api_version()

cinder.api.openstack.versioned_method module

class VersionedMethod(*name*, *start_version*, *end_version*, *experimental*, *func*)

Bases: `ComparableMixin`

cinder.api.openstack.wsgi module

class ActionDispatcher

Bases: `object`

Maps method name to local methods through action name.

default(*data*)

dispatch(**args*, ***kwargs*)

Find and call local method.

class Controller(*view_builder=None*)

Bases: `object`

Default controller.

classmethod api_version(*min_ver*, *max_ver=None*, *experimental=False*)

Decorator for versioning API methods.

Add the decorator to any method which takes a request object as the first parameter and belongs to a class which inherits from wsgi.Controller.

Parameters

- **min_ver** string representing minimum version
- **max_ver** optional string representing maximum version

static assert_valid_body(*body*, *entity_name*)

static validate_name_and_description(*body*, *check_length=True*)

static validate_string_length(*value*, *entity_name*, *min_length=0*, *max_length=None*, *remove_whitespaces=False*)

Check the length of specified string.

Parameters

- **value** the value of the string
- **entity_name** the name of the string
- **min_length** the min_length of the string
- **max_length** the max_length of the string
- **remove_whitespaces** True if trimming whitespaces is needed else False

wsgi_actions = {}

wsgi_extensions = []

class ControllerMetaclass(*name*, *bases*, *cls_dict*)

Bases: `type`

Controller metaclass.

This metaclass automates the task of assembling a dictionary mapping action keys to method names.

static consolidate_vers(*versioned_methods*)

Consolidates a list of versioned methods dictionaries.

class DictSerializer

Bases: *ActionDispatcher*

Default request body serialization.

default(*data*)

serialize(*data*, *action='default'*)

exception Fault(*exception*)

Bases: HTTPException

Wrap webob.exc.HTTPException to provide API friendly response.

class JSONDeserializer

Bases: *TextDeserializer*

default(*datastring*)

class JSONDictSerializer

Bases: *DictSerializer*

Default JSON request body serialization.

default(*data*)

exception OverLimitFault(*message*, *details*, *retry_time*)

Bases: HTTPException

Rate-limited request response.

class Request(**args*, ***kwargs*)

Bases: Request

Add some OpenStack API-specific logic to the base webob.Request.

best_match_content_type()

Determine the requested response content-type.

best_match_language()

Determines best available locale from the Accept-Language header.

Returns

the best language match or None if the Accept-Language header was not available in the request.

cache_db_backup(*backup*)

cache_db_backups(*backups*)

cache_db_items(*key*, *items*, *item_key='id'*)

Get cached database items.

Allow API methods to store objects from a DB query to be used by API extensions within the same API request.

An instance of this class only lives for the lifetime of a single API request, so theres no need to implement full cache management.

cache_db_snapshot(*snapshot*)

cache_db_snapshots(*snapshots*)

cache_db_volume(*volume*)

cache_db_volume_type(*volume_type*)

cache_db_volume_types(*volume_types*)

cache_db_volumes(*volumes*)

cache_resource(*resource_to_cache*, *id_attribute='id'*, *name=None*)

Cache the given resource.

Allow API methods to cache objects, such as results from a DB query, to be used by API extensions within the same API request.

The resource_to_cache can be a list or an individual resource, but ultimately resources are cached individually using the given id_attribute.

Different resources types might need to be cached during the same request, they can be cached using the name parameter. For example:

Controller 1:
request.cache_resource(db_volumes, volumes) request.cache_resource(db_volume_types, types)

Controller 2:
db_volumes = request.cached_resource(volumes) db_type_1 = request.cached_resource_by_id(1, types)

If no name is given, a default name will be used for the resource.

An instance of this class only lives for the lifetime of a single API request, so theres no need to implement full cache management.

cached_resource(*name=None*)

Get the cached resources cached under the given resource name.

Allow an API extension to get previously stored objects within the same API request.

Note that the object data will be slightly stale.

Returns
a dict of id_attribute to the resource from the cached resources, an empty map if an empty collection was cached, or None if nothing has been cached yet under this name

cached_resource_by_id(*resource_id*, *name=None*)

Get a resource by ID cached under the given resource name.

Allow an API extension to get a previously stored object within the same API request. This is basically a convenience method to lookup by ID on the dictionary of all cached resources.

Note that the object data will be slightly stale.

Returns
the cached resource or None if the item is not in the cache

get_content_type()

Determine content type of the request body.

Does not do any body introspection, only checks header

get_db_backup(*backup_id*)

get_db_backups()

get_db_item(*key*, *item_key*)

Get database item.

Allow an API extension to get a previously stored object within the same API request.

Note that the object data will be slightly stale.

get_db_items(*key*)

Get database items.

Allow an API extension to get previously stored objects within the same API request.

Note that the object data will be slightly stale.

get_db_snapshot(*snapshot_id*)

get_db_snapshots()

get_db_volume(*volume_id*)

get_db_volume_type(*volume_type_id*)

get_db_volume_types()

get_db_volumes()

set_api_version_request(*url*)

Set API version request based on the request header information.

class Resource(*controller*, *action_peek=None*, ***deserializers*)

Bases: `Application`

WSGI app that handles (de)serialization and controller dispatch.

WSGI app that reads routing information supplied by RoutesMiddleware and calls the requested action method upon its controller. All controller action methods must accept a req argument, which is the incoming wsgi.Request. If the operation is a PUT or POST, the controller method must also accept a body argument (the deserialized request body). They may raise a webob.exc exception or return a dict, which will be serialized by requested content type.

Exceptions derived from webob.exc.HTTPException will be automatically wrapped in Fault() to provide API friendly error responses.

deserialize(*meth*, *content_type*, *body*)

dispatch(*method*, *request*, *action_args*)

Dispatch a call to the action-specific method.

get_action_args(*request_environment*)

Parse dictionary created by routes library.

get_body(*request*)

get_method(*request*, *action*, *content_type*, *body*)

Look up the action-specific method and its extensions.

post_process_extensions(*extensions*, *resp_obj*, *request*, *action_args*)

pre_process_extensions(*extensions*, *request*, *action_args*)

register_actions(*controller*)

Registers controller actions with this resource.

register_extensions(*controller*)

Registers controller extensions with this resource.

support_api_request_version = True

class ResourceExceptionHandler

Bases: `object`

Context manager to handle Resource exceptions.

Used when processing exceptions generated by API implementation methods (or their extensions). Converts most exceptions to Fault exceptions, with the appropriate logging.

class ResponseObject(*obj*, *code=None*, *headers=None*, ***serializers*)

Bases: `object`

Bundles a response object with appropriate serializers.

Object that app methods may return in order to bind alternate serializers with a response object to be serialized. Its use is optional.

attach(***kwargs*)

Attach slave templates to serializers.

property code

Retrieve the response status.

get_serializer(*content_type*, *default_serializers=None*)

Returns the serializer for the wrapped object.

Returns the serializer for the wrapped object subject to the indicated content type. If no serializer matching the content type is attached, an appropriate serializer drawn from the default serializers will be used. If no appropriate serializer is available, raises InvalidContentType.

property headers

Retrieve the headers.

preserialize(*content_type*, *default_serializers=None*)

Prepares the serializer that will be used to serialize.

Determines the serializer that will be used and prepares an instance of it for later call. This allows the serializer to be accessed by extensions for, e.g., template extension.

serialize(*request*, *content_type*, *default_serializers=None*)

Serializes the wrapped object.

Utility method for serializing the wrapped object. Returns a webob.Response object.

class TextDeserializer

Bases: *ActionDispatcher*

Default request body deserialization.

default(*datastring*)

deserialize(*datastring*, *action='default'*)

action(*name*)

Mark a function as an action.

The given name will be taken as the action key in the body.

This is also overloaded to allow extensions to provide non-extending definitions of create and delete operations.

action_peek_json(*body*)

Determine action to invoke.

deserializers(***deserializers*)

Attaches deserializers to a method.

This decorator associates a dictionary of deserializers with a method. Note that the function attributes are directly manipulated; the method is not wrapped.

extends(**args*, ***kwargs*)

Indicate a function extends an operation.

Can be used as either:

```
@extends
def index(...):
    pass
```

or as:

```
@extends(action='resize')
def _action_resize(...):
    pass
```

response(*code*)

Attaches response code to a method.

This decorator associates a response code with a method. Note that the function attributes are directly manipulated; the method is not wrapped.

serializers(***serializers*)

Attaches serializers to a method.

This decorator associates a dictionary of serializers with a method. Note that the function attributes are directly manipulated; the method is not wrapped.

Module contents

WSGI middleware for OpenStack API controllers.

class APIMapper(*controller_scan=<function controller_scan>*, *directory=None*, *always_scan=False*, *register=True*, *explicit=True*)

Bases: `Mapper`

connect(**args*, ***kwargs*)

Create and connect a new Route to the Mapper.

Usage:

```
m = Mapper()
m.connect(':controller/:action/:id')
m.connect('date/:year/:month/:day', controller="blog",
          action="view")
m.connect('archives/:page', controller="blog", action="by_page",
requirements = { 'page':'\d{1,2}' })
m.connect('category_list', 'archives/category/:section',
          controller='blog', action='category',
          section='home', type='list')
m.connect('home', '', controller='blog', action='view',
          section='home')
```

routematch(*url=None*, *environ=None*)

Match a URL against against one of the routes contained.

Will return None if no valid match is found, otherwise a result dict and a route object is returned.

```
resultdict, route_obj = m.match('/joe/sixpack')
```

class APIRouter(*ext_mgr=None*)

Bases: `Router`

Routes requests on the API to the appropriate controller and method.

ExtensionManager = None

classmethod factory(*global_config*, ***local_config*)

Simple paste factory, `cinder.wsgi.Router` doesnt have.

class ProjectMapper(*controller_scan=<function controller_scan>*, *directory=None*, *always_scan=False*, *register=True*, *explicit=True*)

Bases: `APIMapper`

resource(*member_name*, *collection_name*, ***kwargs*)

Base resource path handler

This method is compatible with resource paths that include a project_id and those that dont. Including project_id in the URLs was a legacy API requirement; and making API requests against such endpoints wont work for users that dont belong to a particular project.

cinder.api.schemas package

Submodules

cinder.api.schemas.admin_actions module

Schema for V3 admin_actions API.

cinder.api.schemas.attachments module

Schema for V3 Attachments API.

cinder.api.schemas.backups module

Schema for V3 Backups API.

cinder.api.schemas.clusters module

Schema for V3 Clusters API.

cinder.api.schemas.default_types module

Schema for V3 Default types API.

cinder.api.schemas.group_snapshots module

Schema for V3 Group Snapshots API.

cinder.api.schemas.group_specs module

cinder.api.schemas.group_types module

Schema for V3 Group types API.

cinder.api.schemas.groups module

Schema for V3 Generic Volume Groups API.

cinder.api.schemas.qos_specs module

cinder.api.schemas.quota_classes module

Schema for V3 Quota classes API.

cinder.api.schemas.quotas module

Schema for V3 Quotas API.

cinder.api.schemas.scheduler_hints module

Schema for V3 scheduler_hints API.

cinder.api.schemas.services module

cinder.api.schemas.snapshot_actions module

Schema for V3 snapshot actions API.

cinder.api.schemas.snapshot_manage module

Schema for V3 snapshot_manage API.

cinder.api.schemas.snapshots module

Schema for V3 Snapshots API.

cinder.api.schemas.types_extra_specs module

Schema for V3 types_extra_specs API.

cinder.api.schemas.volume_actions module

Schema for V3 volume_actions API.

cinder.api.schemas.volume_image_metadata module

Schema for V3 volume image metadata API.

cinder.api.schemas.volume_manage module

Schema for V3 volume manage API.

cinder.api.schemas.volume_metadata module

Schema for V3 Volume metadata API.

cinder.api.schemas.volume_transfer module

Schema for V3 volume transfer API.

cinder.api.schemas.volume_type_access module

Schema for V3 volume type access API.

cinder.api.schemas.volume_type_encryption module

Schema for V3 volume type encryption API.

cinder.api.schemas.volume_types module

cinder.api.schemas.volumes module

Schema for V3 Volumes API.

cinder.api.schemas.workers module

Schema for V3 Workers API.

Module contents

cinder.api.v2 package

Subpackages

cinder.api.v2.views package

Submodules

cinder.api.v2.views.volumes module

class ViewBuilder

Bases: *ViewBuilder*

Model a server API response as a python dictionary.

detail(*request*, *volume*)

Detailed view of a single volume.

detail_list(*request*, *volumes*, *volume_count=None*)

Detailed view of a list of volumes.

summary(*request*, *volume*)

Generic, non-detailed view of a volume.

summary_list(*request*, *volumes*, *volume_count=None*)

Show a list of volumes without many details.

Module contents

Submodules

cinder.api.v2.limits module

Module dedicated functions/classes dealing with rate limiting requests.

class Limit(*verb*, *uri*, *regex*, *value*, *unit*)

Bases: `object`

Stores information about a limit for HTTP requests.

UNITS = {1: 'SECOND', 60: 'MINUTE', 3600: 'HOUR', 86400: 'DAY'}

UNIT_MAP = {'DAY': 86400, 'HOUR': 3600, 'MINUTE': 60, 'SECOND': 1}

display()

Return a useful representation of this class.

display_unit()

Display the string name of the unit.

class Limiter(*limits*, ***kwargs*)

Bases: `object`

Rate-limit checking class which handles limits in memory.

check_for_delay(*verb*, *url*, *username=None*)

Check the given verb/user/user triplet for limit.

@return: Tuple of delay (in seconds) and error message (or None, None)

get_limits(*username=None*)

Return the limits for a given user.

static parse_limits(*limits*)

Convert a string into a list of Limit instances.

This implementation expects a semicolon-separated sequence of parenthesized groups, where each group contains a comma-separated sequence consisting of HTTP method, user-readable URI, a URI reg-exp, an integer number of requests which can be made, and a unit of measure. Valid values for the latter are SECOND, MINUTE, HOUR, and DAY.

@return: List of Limit instances.

class LimitsController(*view_builder=None*)

Bases: `Controller`

Controller for accessing limits in the OpenStack API.

index(*req*)

Return all global and rate limit information.

wsgi_actions = {}

wsgi_extensions = []

class RateLimitingMiddleware(*application*, *limits=None*, *limiter=None*, ***kwargs*)

Bases: `Middleware`

Rate-limits requests passing through this middleware.

All limit information is stored in memory for this implementation.

class WsgiLimiter(*limits=None*)

Bases: `object`

Rate-limit checking from a WSGI application.

Uses an in-memory *Limiter*.

To use, POST `/<username>` with JSON data such as:

```
{
    "verb" : GET,
    "path" : "/servers"
}
```

and receive a 204 No Content, or a 403 Forbidden with an X-Wait-Seconds header containing the number of seconds to wait before the action would succeed.

class WsgiLimiterProxy(*limiter_address*)

Bases: `object`

Rate-limit requests based on answers from a remote source.

`check_for_delay`(*verb*, *path*, *username=None*)

`static parse_limits`(*limits*)

Ignore a limits stringsimply doesnt apply for the limit proxy.

@return: Empty list.

`create_resource`()

cinder.api.v2.snapshots module

The volumes snapshots api.

`class SnapshotsController`(*ext_mgr=None*)

Bases: `Controller`

The Snapshots API controller for the OpenStack API.

`create`(*req*, *body*)

Creates a new snapshot.

`delete`(*req*, *id*)

Delete a snapshot.

`detail`(*req*)

Returns a detailed list of snapshots.

`index`(*req*)

Returns a summary list of snapshots.

`show`(*req*, *id*)

Return data about the given snapshot.

`update`(*req*, *id*, *body*)

Update a snapshot.

`wsgi_actions = {}`

`wsgi_extensions = []`

`create_resource`(*ext_mgr*)

cinder.api.v2.volume_metadata module

`class Controller`

Bases: `Controller`

The volume metadata API controller for the OpenStack API.

`create`(*req*, *volume_id*, *body*)

`delete`(*req*, *volume_id*, *id*)

Deletes an existing metadata.

`index`(*req*, *volume_id*)

Returns the list of metadata for a given volume.

show(*req*, *volume_id*, *id*)
Return a single metadata item.

update(*req*, *volume_id*, *id*, *body*)

update_all(*req*, *volume_id*, *body*)

wsgi_actions = {}

wsgi_extensions = []

create_resource()

cinder.api.v2.volumes module

The volumes api.

class VolumeController(*ext_mgr*)
Bases: `Controller`

The Volumes API controller for the OpenStack API.

create(*req*, *body*)
Creates a new volume.

delete(*req*, *id*)
Delete a volume.

detail(*req*)
Returns a detailed list of volumes.

index(*req*)
Returns a summary list of volumes.

show(*req*, *id*)
Return data about the given volume.

update(*req*, *id*, *body*)
Update a volume.

wsgi_actions = {}

wsgi_extensions = []

create_resource(*ext_mgr*)

Module contents

cinder.api.v3 package

Subpackages

cinder.api.v3.views package

Submodules

cinder.api.v3.views.attachments module

class ViewBuilder

Bases: `object`

Model an attachment API response as a python dictionary.

classmethod detail(*attachment*, *flat=False*)

Detailed view of an attachment.

classmethod list(*attachments*, *detail=False*)

Build a view of a list of attachments.

static summary(*attachment*, *flat=False*)

Non detailed view of an attachment.

cinder.api.v3.views.backups module

class ViewBuilder

Bases: *ViewBuilder*

Model a backups API V3 response as a python dictionary.

detail(*request*, *backup*)

Detailed view of a single backup.

cinder.api.v3.views.clusters module

class ViewBuilder

Bases: `object`

Map Cluster into dicts for API responses.

classmethod detail(*cluster*, *replication_data=False*, *flat=False*)

Detailed view of a cluster.

classmethod list(*clusters*, *detail=False*, *replication_data=False*)

static summary(*cluster*, *replication_data=False*, *flat=False*)

Generic, non-detailed view of a cluster.

cinder.api.v3.views.default_types module

class ViewBuilder

Bases: `object`

Model default type API response as a python dictionary.

create(*default_type*)

Detailed view of a default type when set.

detail(*default_type*)

Build a view of a default type.

```
{"default_type":
    {
      "project_id": "248592b4-a6da-4c4c-abe0-9d8dbe0b74b4",
```

(continues on next page)

(continued from previous page)

```
        "volume_type_id": "6bd1de9a-b8b5-4c43-a597-00170ab06b50"
    }
}
```

index(*default_types*)

Build a view of a list of default types.

```
{"default_types":
   [
    {
        "project_id": "248592b4-a6da-4c4c-abe0-9d8dbe0b74b4",
        "volume_type_id": "7152eb1e-aef0-4bcd-a3ab-46b7ef17e2e6"
    },
    {
        "project_id": "1234567-4c4c-abcd-abe0-1a2b3c4d5e6ff",
        "volume_type_id": "5e3b298a-f1fc-4d32-9828-0d720da81ddd"
    }
 ]
}
```

cinder.api.v3.views.group_snapshots module

class ViewBuilder

Bases: *ViewBuilder*

Model group_snapshot API responses as a python dictionary.

detail(*request*, *group_snapshot*)

Detailed view of a single group_snapshot.

detail_list(*request*, *group_snapshots*)

Detailed view of a list of group_snapshots .

summary(*request*, *group_snapshot*)

Generic, non-detailed view of a group_snapshot.

summary_list(*request*, *group_snapshots*)

Show a list of group_snapshots without many details.

cinder.api.v3.views.group_types module

class ViewBuilder

Bases: *ViewBuilder*

index(*request*, *group_types*)

Index over trimmed group types.

show(*request*, *group_type*, *brief=False*)

Trim away extraneous group type attributes.

cinder.api.v3.views.groups module

class ViewBuilder

Bases: *ViewBuilder*

Model group API responses as a python dictionary.

detail(*request*, *group*)

Detailed view of a single group.

detail_list(*request*, *groups*)

Detailed view of a list of groups .

summary(*request*, *group*)

Generic, non-detailed view of a group.

summary_list(*request*, *groups*)

Show a list of groups without many details.

cinder.api.v3.views.messages module

class ViewBuilder

Bases: *ViewBuilder*

Model a server API response as a python dictionary.

detail(*request*, *message*)

Detailed view of a single message.

index(*request*, *messages*, *message_count=None*)

Show a list of messages.

cinder.api.v3.views.resource_filters module

class ViewBuilder

Bases: `object`

Model an resource filters API response as a python dictionary.

classmethod list(*filters*)

Build a view of a list of resource filters.

```
{
   "resource_filters": [{
       "resource": "resource_1",
       "filters": ["filter1", "filter2", "filter3"]
    }]
}
```

cinder.api.v3.views.snapshots module

class ViewBuilder

Bases: *ViewBuilder*

Model a snapshots API V3 response as a python dictionary.

`detail`(*request*, *snapshot*)

Detailed view of a single snapshot.

cinder.api.v3.views.types module

class ViewBuilder

Bases: *ViewBuilder*

`index`(*request*, *volume_types*)

Index over trimmed volume types.

`show`(*request*, *volume_type*, *brief=False*)

Trim away extraneous volume type attributes.

cinder.api.v3.views.volumes module

class ViewBuilder

Bases: *ViewBuilder*

Model a volumes API V3 response as a python dictionary.

`detail`(*request*, *volume*)

Detailed view of a single volume.

`quick_summary`(*volume_count*, *volume_size*, *all_distinct_metadata=None*)

View of volumes summary.

It includes number of volumes, size of volumes and all distinct metadata of volumes.

cinder.api.v3.views.workers module

class ViewBuilder

Bases: `object`

Map Cluster into dicts for API responses.

classmethod service_list(*services*)

Module contents

Submodules

cinder.api.v3.attachments module

The volumes attachments API.

class AttachmentsController(*ext_mgr=None*)

Bases: *Controller*

The Attachments API controller for the OpenStack API.

`allowed_filters = {'attach_status', 'instance_id', 'status', 'volume_id'}`

`complete`(*req*, *id*, *body*)

Mark a volume attachment process as completed (in-use).

create(*req*, *body*)

Create an attachment.

This method can be used to create an empty attachment (reserve) or to create and initialize a volume attachment based on the provided input parameters.

If the caller does not yet have the connector information but needs to reserve an attachment for the volume (ie Nova BootFromVolume) the create can be called with just the volume-uuid and the server identifier. This will reserve an attachment, mark the volume as reserved and prevent any new attachment_create calls from being made until the attachment is updated (completed).

The alternative is that the connection can be reserved and initialized all at once with a single call if the caller has all of the required information (connector data) at the time of the call.

NOTE: In Nova terms server == instance, the server_id parameter referenced below is the UUID of the Instance, for non-nova consumers this can be a server UUID or some other arbitrary unique identifier.

Starting from microversion 3.54, we can pass the attach mode as argument in the request body.

Expected format of the input parameter body:

```
{
    "attachment":
    {
        "volume_uuid": "volume-uuid",
        "instance_uuid": "null|nova-server-uuid",
        "connector": "null|<connector-object>",
        "mode": "null|rw|ro"
    }
}
```

Example connector:

```
{
    "connector":
    {
        "initiator": "iqn.1993-08.org.debian:01:cad181614cec",
        "ip": "192.168.1.20",
        "platform": "x86_64",
        "host": "tempest-1",
        "os_type": "linux2",
        "multipath": false,
        "mountpoint": "/dev/vdb",
        "mode": "null|rw|ro"
    }
}
```

NOTE all thats required for a reserve is volume_uuid and an instance_uuid.

returns: A summary view of the attachment object

delete(*req*, *id*)

Delete an attachment.

Disconnects/Deletes the specified attachment, returns a list of any known shared attachment-ids for the effected backend device.

returns: A summary list of any attachments sharing this connection

detail(*req*)

Return a detailed list of attachments.

index(*req*)

Return a summary list of attachments.

show(*req*, *id*)

Return data about the given attachment.

update(*req*, *id*, *body*)

Update an attachment record.

Update a reserved attachment record with connector information and set up the appropriate connection_info from the driver.

Expected format of the input parameter body:

```
{
    "attachment":
    {
        "connector":
        {
            "initiator": "iqn.1993-08.org.debian:01:cad181614cec",
            "ip": "192.168.1.20",
            "platform": "x86_64",
            "host": "tempest-1",
            "os_type": "linux2",
            "multipath": false,
            "mountpoint": "/dev/vdb",
            "mode": "None|rw|ro"
        }
    }
}
```

versioned_methods = {'complete': [<cinder.api.openstack.versioned_method.VersionedMethod object>], 'create': [<cinder.api.openstack.versioned_method.VersionedMethod object>], 'delete': [<cinder.api.openstack.versioned_method.VersionedMethod object>], 'detail': [<cinder.api.openstack.versioned_method.VersionedMethod object>], 'index': [<cinder.api.openstack.versioned_method.VersionedMethod object>], 'show': [<cinder.api.openstack.versioned_method.VersionedMethod object>], 'update': [<cinder.api.openstack.versioned_method.VersionedMethod object>]}

wsgi_actions = {'os-complete': 'complete'}

wsgi_extensions = []

create_resource(*ext_mgr*)

Create the wsgi resource for this controller.

cinder.api.v3.backups module

The backups V3 API.

class BackupsController

Bases: *BackupsController*

The backups API controller for the OpenStack API V3.

detail(*req*)

Returns a detailed list of backups.

show(*req*, *id*)

Return data about the given backup.

update(*req*, *id*, *body*)

Update a backup.

versioned_methods = {'update': [<cinder.api.openstack.versioned_method.VersionedMethod object>]}

wsgi_actions = {}

wsgi_extensions = []

create_resource()

cinder.api.v3.clusters module

class ClusterController(*view_builder=None*)

Bases: *Controller*

allowed_list_keys = {'active_backend_id', 'binary', 'disabled', 'frozen', 'is_up', 'name', 'num_down_hosts', 'num_hosts', 'replication_status'}

detail(*req*)

Return a detailed list of all existing clusters.

Filter by is_up, disabled, num_hosts, and num_down_hosts.

index(*req*)

Return a non detailed list of all existing clusters.

Filter by is_up, disabled, num_hosts, and num_down_hosts.

replication_fields = {'active_backend_id', 'frozen', 'replication_status'}

show(*req*, *id*, *binary='cinder-volume'*)

Return data for a given cluster name with optional binary.

update(*req*, *id*, *body*)

Enable/Disable scheduling for a cluster.

versioned_methods = {'detail': [<cinder.api.openstack.versioned_method.VersionedMethod object>], 'index': [<cinder.api.openstack.versioned_method.VersionedMethod object>], 'show': [<cinder.api.openstack.versioned_method.VersionedMethod object>], 'update': [<cinder.api.openstack.versioned_method.VersionedMethod object>]}

wsgi_actions = {}

wsgi_extensions = []

create_resource()

cinder.api.v3.consistencygroups module

The consistencygroups V3 API.

class ConsistencyGroupsController

Bases: *ConsistencyGroupsController*

The ConsistencyGroups API controller for the OpenStack API V3.

update(*req*, *id*, *body*)

Update the consistency group.

Expected format of the input parameter body:

```
{
    "consistencygroup":
    {
        "name": "my_cg",
        "description": "My consistency group",
        "add_volumes": "volume-uuid-1,volume-uuid-2,...",
        "remove_volumes": "volume-uuid-8,volume-uuid-9,..."
    }
}
```

wsgi_actions = {}

wsgi_extensions = []

create_resource()

cinder.api.v3.default_types module

The resource filters api.

class DefaultTypesController(*view_builder=None*)

Bases: *Controller*

The Default types API controller for the OpenStack API.

create_update(*req*, *id*, *body*)

Set a default volume type for the specified project.

delete(*req*, *id*)

Unset a default volume type for a project.

detail(*req*, *id*)

Return detail of a default type.

index(*req*)

Return a list of default types.

versioned_methods = {'create_update': [<cinder.api.openstack.versioned_method.VersionedMethod object>], 'delete': [<cinder.api.openstack.versioned_method.VersionedMethod object>], 'detail': [<cinder.api.openstack.versioned_method.VersionedMethod object>], 'index': [<cinder.api.openstack.versioned_method.VersionedMethod object>]}

wsgi_actions = {}

wsgi_extensions = []

create_resource()

Create the wsgi resource for this controller.

cinder.api.v3.group_snapshots module

The group_snapshots API.

class GroupSnapshotsController

Bases: *Controller*

The group_snapshots API controller for the OpenStack API.

create(*req*, *body*)

Create a new group_snapshot.

delete(*req*, *id*)

Delete a group_snapshot.

detail(*req*)

Returns a detailed list of group_snapshots.

index(*req*)

Returns a summary list of group_snapshots.

reset_status(*req*, *id*, *body*)

show(*req*, *id*)

Return data about the given group_snapshot.

versioned_methods = {'create': [<cinder.api.openstack.versioned_method.VersionedMethod object>], 'delete': [<cinder.api.openstack.versioned_method.VersionedMethod object>], 'detail': [<cinder.api.openstack.versioned_method.VersionedMethod object>], 'index': [<cinder.api.openstack.versioned_method.VersionedMethod object>], 'reset_status': [<cinder.api.openstack.versioned_method.VersionedMethod object>], 'show': [<cinder.api.openstack.versioned_method.VersionedMethod object>]}

wsgi_actions = {'reset_status': 'reset_status'}

wsgi_extensions = []

create_resource()

cinder.api.v3.group_specs module

The group types specs controller

class GroupTypeSpecsController(*view_builder=None*)

Bases: *Controller*

The group type specs API controller for the OpenStack API.

create(*req*, *group_type_id*, *body*)

delete(*req*, *group_type_id*, *id*)

Deletes an existing group spec.

index(*req*, *group_type_id*)

Returns the list of group specs for a given group type.

show(*req*, *group_type_id*, *id*)

Return a single extra spec item.

update(*req*, *group_type_id*, *id*, *body*)

versioned_methods = {'create': [<cinder.api.openstack.versioned_method.VersionedMethod object>], 'delete': [<cinder.api.openstack.versioned_method.VersionedMethod object>], 'index': [<cinder.api.openstack.versioned_method.VersionedMethod object>], 'show': [<cinder.api.openstack.versioned_method.VersionedMethod object>], 'update': [<cinder.api.openstack.versioned_method.VersionedMethod object>]}

wsgi_actions = {}

wsgi_extensions = []

create_resource()

cinder.api.v3.group_types module

The group type & group type specs controller.

`class GroupTypesController(view_builder=None)`

Bases: `Controller`

The group types API controller for the OpenStack API.

`create(req, body)`

Creates a new group type.

`delete(req, id)`

Deletes an existing group type.

`index(req)`

Returns the list of group types.

`show(req, id)`

Return a single group type item.

`update(req, id, body)`

`versioned_methods = {'create': [<cinder.api.openstack.versioned_method.VersionedMethod object>], 'delete': [<cinder.api.openstack.versioned_method.VersionedMethod object>], 'index': [<cinder.api.openstack.versioned_method.VersionedMethod object>], 'show': [<cinder.api.openstack.versioned_method.VersionedMethod object>], 'update': [<cinder.api.openstack.versioned_method.VersionedMethod object>]}`

`wsgi_actions = {}`

`wsgi_extensions = []`

`create_resource()`

cinder.api.v3.groups module

The groups controller.

`class GroupsController`

Bases: `Controller`

The groups API controller for the OpenStack API.

`create(req, body)`

Create a new group.

`create_from_src(req, body)`

Create a new group from a source.

The source can be a group snapshot or a group. Note that this does not require group_type and volume_types as the create API above.

delete_group(*req*, *id*, *body*)

detail(*req*)

Returns a detailed list of groups.

disable_replication(*req*, *id*, *body*)

Disables replications for a group.

enable_replication(*req*, *id*, *body*)

Enables replications for a group.

failover_replication(*req*, *id*, *body*)

Fails over replications for a group.

index(*req*)

Returns a summary list of groups.

list_replication_targets(*req*, *id*, *body*)

List replication targets for a group.

reset_status(*req*, *id*, *body*)

show(*req*, *id*)

Return data about the given group.

update(*req*, *id*, *body*)

Update the group.

Expected format of the input parameter body:

```
{
    "group":
    {
        "name": "my_group",
        "description": "My group",
        "add_volumes": "volume-uuid-1,volume-uuid-2,...",
        "remove_volumes": "volume-uuid-8,volume-uuid-9,..."
    }
}
```

versioned_methods = {'create': [<cinder.api.openstack.versioned_method.VersionedMethod object>], 'create_from_src': [<cinder.api.openstack.versioned_method.VersionedMethod object>], 'delete_group': [<cinder.api.openstack.versioned_method.VersionedMethod object>], 'detail': [<cinder.api.openstack.versioned_method.VersionedMethod object>], 'disable_replication': [<cinder.api.openstack.versioned_method.VersionedMethod object>], 'enable_replication': [<cinder.api.openstack.versioned_method.VersionedMethod object>], 'failover_replication': [<cinder.api.openstack.versioned_method.VersionedMethod object>], 'index': [<cinder.api.openstack.versioned_method.VersionedMethod object>], 'list_replication_targets': [<cinder.api.openstack.versioned_method.VersionedMethod object>], 'reset_status': [<cinder.api.openstack.versioned_method.VersionedMethod object>], 'show': [<cinder.api.openstack.versioned_method.VersionedMethod object>], 'update': [<cinder.api.openstack.versioned_method.VersionedMethod object>]}

wsgi_actions = {'create-from-src': 'create_from_src', 'delete': 'delete_group', 'disable_replication': 'disable_replication', 'enable_replication': 'enable_replication', 'failover_replication': 'failover_replication', 'list_replication_targets': 'list_replication_targets', 'reset_status': 'reset_status'}

wsgi_extensions = []

create_resource()

cinder.api.v3.limits module

The limits V3 api.

class LimitsController(*view_builder=None*)

Bases: *LimitsController*

Controller for accessing limits in the OpenStack API.

index(*req*)

Return all global and rate limit information.

wsgi_actions = {}

wsgi_extensions = []

create_resource()

cinder.api.v3.messages module

The messages API.

class MessagesController(*ext_mgr*)

Bases: `Controller`

The User Messages API controller for the OpenStack API.

delete(*req*, *id*)

Delete a message.

index(*req*)

Returns a list of messages, transformed through view builder.

show(*req*, *id*)

Return the given message.

versioned_methods = {'delete': [<cinder.api.openstack.versioned_method.VersionedMethod object>], 'index': [<cinder.api.openstack.versioned_method.VersionedMethod object>], 'show': [<cinder.api.openstack.versioned_method.VersionedMethod object>]}

wsgi_actions = {}

wsgi_extensions = []

create_resource(*ext_mgr*)

cinder.api.v3.resource_common_manage module

class ManageResource

Bases: `object`

Mixin class for v3 of ManageVolume and ManageSnapshot.

It requires that any class inheriting from this one has *volume_api* and *_list_manageable_view* attributes.

VALID_SORT_DIRS = {'asc', 'desc'}

VALID_SORT_KEYS = {'reference', 'size'}

detail(*req*)

Returns a detailed list of volumes available to manage.

index(*req*)

Returns a summary list of volumes available to manage.

cinder.api.v3.resource_filters module

The resource filters api.

class ResourceFiltersController(*ext_mgr=None*)

Bases: `Controller`

The resource filter API controller for the OpenStack API.

index(*req*)

Return a list of resource filters.

versioned_methods = {'index': [<cinder.api.openstack.versioned_method.VersionedMethod object>]}

wsgi_actions = {}

wsgi_extensions = []

create_resource(*ext_mgr*)

Create the wsgi resource for this controller.

cinder.api.v3.router module

WSGI middleware for OpenStack Volume API.

class APIRouter(*ext_mgr=None*)

Bases: *APIRouter*

Routes requests on the API to the appropriate controller and method.

class ExtensionManager

Bases: `object`

Load extensions from the configured extension path.

See cinder/tests/api/extensions/foxinsocks/extension.py for an example extension implementation.

get_controller_extensions()

Returns a list of ControllerExtension objects.

get_resources()

Returns a list of ResourceExtension objects.

is_loaded(*alias*)

load_extension(*ext_factory*)

Execute an extension factory.

Loads an extension. The ext_factory is the name of a callable that will be imported and called with one argumentthe extension manager. The factory callable is expected to call the register() method at least once.

register(*ext*)

cinder.api.v3.snapshot_manage module

class SnapshotManageController(**args*, ***kwargs*)

Bases: *ManageResource*, *SnapshotManageController*

create(*req*, *body*)

Instruct Cinder to manage a storage snapshot object.

Manages an existing backend storage snapshot object (e.g. a Linux logical volume or a SAN disk) by creating the Cinder objects required to manage it, and possibly renaming the backend storage snapshot object (driver dependent).

From an API perspective, this operation behaves very much like a snapshot creation operation.

Required HTTP Body:

```
{
  "snapshot":
  {
    "volume_id": "<Cinder volume already exists in volume backend>",
    "ref":
       "<Driver-specific reference to the existing storage object>"
  }
}
```

See the appropriate Cinder drivers implementations of the manage_snapshot method to find out the accepted format of ref. For example,in LVM driver, it will be the logic volume name of snapshot which you want to manage.

This API call will return with an error if any of the above elements are missing from the request, or if the volume_id element refers to a cinder volume that could not be found.

The snapshot will later enter the error state if it is discovered that ref is bad.

Optional elements to snapshot are:

```
name            A name for the new snapshot.
description     A description for the new snapshot.
metadata        Key/value pairs to be associated with the new␣
↪snapshot.
```

wsgi_actions = {}

wsgi_extensions = []

create_resource()

cinder.api.v3.snapshot_metadata module

class Controller

Bases: *Controller*

The snapshot metadata API controller for the OpenStack API.

create(*req*, *snapshot_id*, *body*)

delete(*req*, *snapshot_id*, *id*)

Deletes an existing metadata.

index(*req*, *snapshot_id*)

Returns the list of metadata for a given snapshot.

show(*req*, *snapshot_id*, *id*)

Return a single metadata item.

update(*req*, *snapshot_id*, *id*, *body*)

update_all(*req*, *snapshot_id*, *body*)

wsgi_actions = {}

wsgi_extensions = []

create_resource()

cinder.api.v3.snapshots module

The volumes snapshots V3 API.

class **SnapshotsController**(*ext_mgr=None*)

Bases: `SnapshotsController`

The Snapshots API controller for the OpenStack API.

MV_ADDED_FILTERS = (('3.21', 'metadata'), ('3.64', 'use_quota'))

create(*req*, *body*)

Creates a new snapshot.

wsgi_actions = {}

wsgi_extensions = []

create_resource(*ext_mgr*)

cinder.api.v3.types module

The volume type & volume types extra specs extension.

class **VolumeTypesController**(*view_builder=None*)

Bases: `Controller`

The volume types API controller for the OpenStack API.

index(*req*)

Returns the list of volume types.

show(*req*, *id*)

Return a single volume type item.

wsgi_actions = {}

wsgi_extensions = []

create_resource()

cinder.api.v3.volume_manage module

class **VolumeManageController**(**args*, ***kwargs*)

Bases: `ManageResource`, `VolumeManageController`

create(*req*, *body*)

Instruct Cinder to manage a storage object.

Manages an existing backend storage object (e.g. a Linux logical volume or a SAN disk) by creating the Cinder objects required to manage it, and possibly renaming the backend storage object (driver dependent)

From an API perspective, this operation behaves very much like a volume creation operation, except that properties such as image, snapshot and volume references dont make sense, because we are taking an existing storage object into Cinder management.

Required HTTP Body:

```
{
  "volume": {
    "host": "<Cinder host on which the existing storage resides>",
    "cluster": "<Cinder cluster on which the storage resides>",
    "ref": "<Driver-specific reference to existing storage object>"
  }
}
```

See the appropriate Cinder drivers implementations of the manage_volume method to find out the accepted format of ref.

This API call will return with an error if any of the above elements are missing from the request, or if the host element refers to a cinder host that is not registered.

The volume will later enter the error state if it is discovered that ref is bad.

Optional elements to volume are:

```
name               A name for the new volume.
description        A description for the new volume.
volume_type        ID or name of a volume type to associate with
                   the new Cinder volume. Does not necessarily
                   guarantee that the managed volume will have the
                   properties described in the volume_type. The
                   driver may choose to fail if it identifies that
                   the specified volume_type is not compatible with
                   the backend storage object.
metadata           Key/value pairs to be associated with the new
                   volume.
availability_zone  The availability zone to associate with the new
                   volume.
bootable           If set to True, marks the volume as bootable.
```

wsgi_actions = {}

wsgi_extensions = []

create_resource()

cinder.api.v3.volume_metadata module

The volume metadata V3 api.

class Controller

Bases: *Controller*

The volume metadata API controller for the OpenStack API.

index(*req*, *volume_id*)

Returns the list of metadata for a given volume.

update(*req*, *volume_id*, *id*, *body*)

update_all(*req*, *volume_id*, *body*)

wsgi_actions = {}

wsgi_extensions = [('index', None), ('update', None), ('update_all', None)]

create_resource()

cinder.api.v3.volume_transfer module

class VolumeTransferController

Bases: *VolumeTransferController*

The transfer API controller for the OpenStack API V3.

create(*req*, *body*)

Create a new volume transfer.

detail(*req*)

Returns a detailed list of transfers.

index(*req*)

Returns a summary list of transfers.

wsgi_actions = {}

wsgi_extensions = []

create_resource()

cinder.api.v3.volumes module

The volumes V3 api.

class VolumeController(*ext_mgr=None*)

Bases: *VolumeController*

The Volumes API controller for the OpenStack API V3.

MV_ADDED_FILTERS = (('3.3', 'glance_metadata'), ('3.9', 'group_id'), ('3.59', 'created_at'), ('3.59', 'updated_at'), ('3.64', 'use_quota'))

create(*req*, *body*)

Creates a new volume.

Parameters

- **req** the request
- **body** the request body

Returns

dict the new volume dictionary

Raises

HTTPNotFound, HTTPBadRequest

delete(*req*, *id*)

Delete a volume.

revert(*req*, *id*, *body*)

revert a volume to a snapshot

summary(*req*)

Return summary of volumes.

versioned_methods = {'revert': [<cinder.api.openstack.versioned_method.VersionedMethod object>], 'summary': [<cinder.api.openstack.versioned_method.VersionedMethod object>]}

wsgi_actions = {'revert': 'revert'}

wsgi_extensions = []

create_resource(*ext_mgr*)

cinder.api.v3.workers module

class WorkerController(**args*, ***kwargs*)

Bases: Controller

cleanup(*req*, *body=None*)

Do the cleanup on resources from a specific service/host/node.

versioned_methods = {'cleanup': [<cinder.api.openstack.versioned_method.VersionedMethod object>]}

wsgi_actions = {}

wsgi_extensions = []

create_resource()

Module contents

cinder.api.validation package

Submodules

cinder.api.validation.parameter_types module

Common parameter types for validating request Body.

cinder.api.validation.validators module

Internal implementation of request Body validating middleware.

class FormatChecker(*formats: Iterable[str] | None = None*)

Bases: `FormatChecker`

A FormatChecker can output the message from cause exception

We need understandable validation errors messages for users. When a custom checker has an exception, the FormatChecker will output a readable message provided by the checker.

check(*param_value*, *format*)

Check whether the param_value conforms to the given format.

Parameters

- **param_value** the param_value to check
- **format** (*str*) the format that param_value should conform to

Type

any primitive type (str, number, bool)

Raises

`FormatError` if param_value does not conform to format

Module contents

Request Body validating middleware.

schema(*request_body_schema*, *min_version=None*, *max_version=None*)

Register a schema to validate request body.

Registered schema will be used for validating request body just before API method executing.

Parameters

- **request_body_schema** (*dict*) a schema to validate request body
- **min_version** A string of two numerals. X.Y indicating the minimum version of the JSON-Schema to validate against.
- **max_version** A string of two numerals. X.Y indicating the maximum version of the JSON-Schema to validate against.

cinder.api.views package

Submodules

cinder.api.views.availability_zones module

class ViewBuilder

Bases: *ViewBuilder*

Map cinder.volumes.api list_availability_zones response into dicts.

list(*request*, *availability_zones*)

cinder.api.views.backups module

class ViewBuilder

Bases: *ViewBuilder*

Model backup API responses as a python dictionary.

detail(*request*, *backup*)

Detailed view of a single backup.

detail_list(*request*, *backups*, *backup_count=None*)

Detailed view of a list of backups .

export_summary(*request*, *export*)

Generic view of an export.

restore_summary(*request*, *restore*)

Generic, non-detailed view of a restore.

summary(*request*, *backup*)

Generic, non-detailed view of a backup.

summary_list(*request*, *backups*, *backup_count=None*)

Show a list of backups without many details.

cinder.api.views.capabilities module

class ViewBuilder

Bases: *ViewBuilder*

Model capabilities API responses as a python dictionary.

summary(*request*, *capabilities*, *id*)

Summary view of a backend capabilities.

cinder.api.views.cgsnapshots module

class ViewBuilder

Bases: *ViewBuilder*

Model cgsnapshot API responses as a python dictionary.

detail(*request*, *cgsnapshot*)

Detailed view of a single cgsnapshot.

detail_list(*request*, *cgsnapshots*)

Detailed view of a list of cgsnapshots .

summary(*request*, *cgsnapshot*)

Generic, non-detailed view of a cgsnapshot.

summary_list(*request*, *cgsnapshots*)

Show a list of cgsnapshots without many details.

cinder.api.views.consistencygroups module

class ViewBuilder

Bases: *ViewBuilder*

Model consistencygroup API responses as a python dictionary.

detail(*request*, *consistencygroup*)

Detailed view of a single consistency group.

detail_list(*request*, *consistencygroups*)

Detailed view of a list of consistency groups .

summary(*request*, *consistencygroup*)

Generic, non-detailed view of a consistency group.

summary_list(*request*, *consistencygroups*)

Show a list of consistency groups without many details.

cinder.api.views.limits module

class ViewBuilder

Bases: `object`

OpenStack API base limits view builder.

build(*rate_limits*, *absolute_limits*)

cinder.api.views.manageable_snapshots module

class ViewBuilder

Bases: *ViewBuilder*

Model manageable snapshot responses as a python dictionary.

detail(*request*, *snapshot*)

Detailed view of a manageable snapshot description.

detail_list(*request*, *snapshots*, *count*)

Detailed view of a list of manageable snapshots.

summary(*request*, *snapshot*)

Generic, non-detailed view of a manageable snapshot description.

summary_list(*request*, *snapshots*, *count*)

Show a list of manageable snapshots without many details.

cinder.api.views.manageable_volumes module

class ViewBuilder

Bases: *ViewBuilder*

Model manageable volume responses as a python dictionary.

detail(*request*, *volume*)

Detailed view of a manageable volume description.

detail_list(*request*, *volumes*, *count*)

Detailed view of a list of manageable volumes.

summary(*request*, *volume*)

Generic, non-detailed view of a manageable volume description.

summary_list(*request*, *volumes*, *count*)

Show a list of manageable volumes without many details.

cinder.api.views.qos_specs module

class ViewBuilder

Bases: *ViewBuilder*

Model QoS specs API responses as a python dictionary.

associations(*request*, *associates*)

View of qos specs associations.

detail(*request*, *qos_spec*)

Detailed view of a single qos_spec.

summary(*request*, *qos_spec*)

Generic, non-detailed view of a qos_specs.

summary_list(*request*, *qos_specs*, *qos_count=None*)

Show a list of qos_specs without many details.

cinder.api.views.scheduler_stats module

class ViewBuilder

Bases: *ViewBuilder*

Model scheduler-stats API responses as a python dictionary.

detail(*request*, *pool*)

Detailed view of a single pool.

pools(*request*, *pools*, *detail*)

Detailed/Summary view of a list of pools seen by scheduler.

summary(*request*, *pool*)

Summary view of a single pool.

cinder.api.views.snapshots module

class ViewBuilder

Bases: *ViewBuilder*

Model snapshot API responses as a python dictionary.

detail(*request*, *snapshot*)

Detailed view of a single snapshot.

detail_list(*request*, *snapshots*, *snapshot_count=None*)

Detailed view of a list of snapshots.

summary(*request*, *snapshot*)

Generic, non-detailed view of a snapshot.

summary_list(*request*, *snapshots*, *snapshot_count=None*)

Show a list of snapshots without many details.

cinder.api.views.transfers module

class ViewBuilder

Bases: *ViewBuilder*

Model transfer API responses as a python dictionary.

create(*request*, *transfer*)

Detailed view of a single transfer when created.

detail(*request*, *transfer*)

Detailed view of a single transfer.

detail_list(*request*, *transfers*, *origin_transfer_count*)

Detailed view of a list of transfers .

summary(*request*, *transfer*)

Generic, non-detailed view of a transfer.

summary_list(*request*, *transfers*, *origin_transfer_count*)

Show a list of transfers without many details.

cinder.api.views.types module

class ViewBuilder

Bases: *ViewBuilder*

index(*request*, *volume_types*)

Index over trimmed volume types.

show(*request*, *volume_type*, *brief=False*)

Trim away extraneous volume type attributes.

cinder.api.views.versions module

class ViewBuilder(*base_url*)

Bases: object

build_versions(*versions*)

get_view_builder(*req*)

Module contents

Submodules

cinder.api.api_utils module

class GenericProjectInfo(*project_id: str*, *project_keystone_api_version: str*, *domain_id: str | None = None*, *name: str | None = None*, *description: str | None = None*)

Bases: object

Abstraction layer for Keystone V2 and V3 project objects

add_visible_admin_metadata(*volume*) → None

Add user-visible admin metadata to regular metadata.

Extracts the admin metadata keys that are to be made visible to non-administrators, and adds them to the regular metadata structure for the passed-in volume.

get_project(*context: context.RequestContext*, *project_id: str*) → *GenericProjectInfo*

Method to verify project exists in keystone

is_none_string(*val: Any*) → bool

Check if a string represents a None value.

remove_invalid_filter_options(*context: context.RequestContext*, *filters: dict*, *allowed_search_options: Iterable[str]*) → None

Remove search options that are not valid for non-admin API/context.

validate_integer(*value: int*, *name: str*, *min_value: int | None = None*, *max_value: int | None = None*) → int

Make sure that value is a valid integer, potentially within range.

Parameters

- **value** the value of the integer
- **name** the name of the integer
- **min_value** the min value of the integer
- **max_value** the max value of the integer

Returns

integer

validate_project_and_authorize(*context: context.RequestContext*, *project_id: str*, *policy_check: str*, *validate_only: bool = False*) → None

walk_class_hierarchy(*clazz: type*, *encountered: list[type] | None = None*) → Generator[type, None, None]

Walk class hierarchy, yielding most derived classes first.

cinder.api.common module

class **METADATA_TYPES**(*value*)

Bases: `Enum`

An enumeration.

image = 2

user = 1

class **ViewBuilder**

Bases: `object`

Model API responses as dictionaries.

convert_filter_attributes(*filters*, *resource*)

get_cluster_host(*req: Request*, *params: dict*, *cluster_version=None*) → tuple[str | None, str | None]

Get cluster and host from the parameters.

This method checks the presence of cluster and host parameters and returns them depending on the cluster_version.

If cluster_version is False we will never return the cluster_name and we will require the presence of the host parameter.

If cluster_version is None we will always check for the presence of the cluster parameter, and if cluster_version is a string with a version we will only check for the presence of the parameter if the version of the request is not less than it. In both cases we will require one and only one parameter, host or cluster.

get_enabled_resource_filters(*resource: str | None = None*) → dict[str, Any]

Get list of configured/allowed filters for the specified resource.

This method checks resource_query_filters_file and returns dictionary which contains the specified resource and its allowed filters:

```
{
    "resource": ["filter1", "filter2", "filter3"]
}
```

if resource is not specified, all of the configuration will be returned, and if the resource is not found, empty dict will be returned.

get_pagination_params(*params: dict*, *max_limit: int | None = None*) → tuple

Return marker, limit, offset tuple from request.

Parameters

params *wsgi.Requests* GET dictionary, possibly containing marker, limit, and offset variables. marker is the id of the last element the client has seen, limit is the maximum number of items to return and offset is the number of items to skip from the marker or from the first element. If limit is not specified, or > max_limit, we default to max_limit. Negative values for either offset or limit will cause exc.HTTPBadRequest() exceptions to be raised. If no offset is present well default to 0 and if no marker is present well default to None.

Max_limit
Max value limit return value can take

Returns
Tuple (marker, limit, offset)

`get_request_url`(*request: Request*) → str

`get_sort_params`(*params: dict*, *default_key: str = 'created_at'*, *default_dir: str = 'desc'*) → tuple[list[str], list[str]]

Retrieves sort keys/directions parameters.

Processes the parameters to create a list of sort keys and sort directions that correspond to either the sort parameter or the sort_key and sort_dir parameter values. The value of the sort parameter is a comma- separated list of sort keys, each key is optionally appended with :<sort_direction>.

Note that the sort_key and sort_dir parameters are deprecated in kilo and an exception is raised if they are supplied with the sort parameter.

The sort parameters are removed from the request parameters by this function.

Parameters

- **params** webob.multidict of request parameters (from cinder.api.openstack.wsgi.Request.params)
- **default_key** default sort key value, added to the list if no sort keys are supplied
- **default_dir** default sort dir value, added to the list if the corresponding key does not have a direction specified

Returns
list of sort keys, list of sort dirs

Raises
webob.exc.HTTPBadRequest If both sort and either sort_key or sort_dir are supplied parameters

`get_time_comparison_operators`() → tuple[str, ...]

Get time comparison operators.

This method returns tuple which contains the allowed comparison operators.

`limited`(*items: list*, *request: Request*, *max_limit: int | None = None*) → list

Return a slice of items according to requested offset and limit.

Parameters

- **items** A sliceable entity
- **request** `wsgi.Request` possibly containing offset and limit GET variables. offset is where to start in the list, and limit is the maximum number of items to return. If limit is not specified, 0, or > max_limit, we default to max_limit.

Negative values for either offset or limit will cause exc.HTTPBadRequest() exceptions to be raised.

- **max_limit** The maximum number of items to return from items

process_general_filtering(*resource*)

reject_invalid_filters(*context: context.RequestContext*, *filters*, *resource: str*, *enable_like_filter: bool = False*)

remove_version_from_href(*href: str*) → str

Removes the first API version from the href.

Given: http://cinder.example.com/v1.1/123 Returns: http://cinder.example.com/123

Given: http://cinder.example.com/v1.1 Returns: http://cinder.example.com

Given: http://cinder.example.com/volume/drivers/v1.1/flashsystem Returns: http://cinder.example.com/volume/drivers/flashsystem

cinder.api.extensions module

class ControllerExtension(*extension*, *collection*, *controller*)

Bases: `object`

Extend core controllers of cinder OpenStack API.

Provide a way to extend existing cinder OpenStack API core controllers.

class ExtensionDescriptor(*ext_mgr*)

Bases: `object`

Base class that defines the contract for extensions.

Note that you dont have to derive from this class to have a valid extension; it is purely a convenience.

alias = None

get_controller_extensions()

List of extensions.ControllerExtension extension objects.

Controller extensions are used to extend existing controllers.

get_resources()

List of extensions.ResourceExtension extension objects.

Resources define new nouns, and are accessible through URLs.

name = None

updated = None

class ExtensionManager

Bases: `object`

Load extensions from the configured extension path.

See cinder/tests/api/extensions/foxinsocks/extension.py for an example extension implementation.

get_controller_extensions()

Returns a list of ControllerExtension objects.

get_resources()

Returns a list of ResourceExtension objects.

is_loaded(*alias*)

load_extension(*ext_factory*)

Execute an extension factory.

Loads an extension. The ext_factory is the name of a callable that will be imported and called with one argumentthe extension manager. The factory callable is expected to call the register() method at least once.

register(*ext*)

class **ExtensionsResource**(*extension_manager*)

Bases: `Resource`

create(*req*)

delete(*req*, *id*)

index(*req*)

show(*req*, *id*)

class **ResourceExtension**(*collection*, *controller*, *parent=None*, *collection_actions=None*, *member_actions=None*, *custom_routes_fn=None*)

Bases: `object`

Add top level resources to the OpenStack API in cinder.

extension_authorizer(*api_name*, *extension_name*)

load_standard_extensions(*ext_mgr*, *logger*, *path*, *package*, *ext_list=None*)

Registers all standard API extensions.

cinder.api.microversions module

API Microversion definitions.

All new microversions should have a constant added here to be used throughout the code instead of the specific version number. Until patches land, its common to end up with merge conflicts with other microversion changes. Merge conflicts will be easier to handle via the microversion constants defined here as the version number will only need to be changed in a single location.

Actual version numbers should be used:

- In this file
- In cinder/api/openstack/rest_api_version_history.rst
- In cinder/api/openstack/api_version_request.py
- In release notes describing the new functionality
- In updates to api-ref

Nearly all microversion changes should include changes to all of those locations. Make sure to add relevant documentation, and make sure that documentation includes the final version number used.

get_api_version(*version*)

Gets a `APIVersionRequest` instance.

Parameters
version The microversion needed.

Returns
The `APIVersionRequest` instance.

get_mv_header(*version*)

Gets a formatted HTTP microversion header.

Parameters
version The microversion needed.

Returns
A tuple containing the microversion header with the requested version value.

get_prior_version(*version*)

Gets the microversion before the given version.

Mostly useful for testing boundaries. This gets the microversion defined just prior to the given version.

Parameters
version The version of interest.

Returns
The version just prior to the given version.

cinder.api.urlmap module

class Accept(*value*)

Bases: `object`

best_match(*supported_content_types*)

content_type_params(*best_content_type*)

Find parameters in Accept header for given content type.

class URLMap(*not_found_app=None*)

Bases: `URLMap`

parse_list_header(*value*)

Parse lists as described by RFC 2068 Section 2.

In particular, parse comma-separated lists where the elements of the list may include quoted-strings. A quoted-string could contain a comma. A non-quoted string could have quotes in the middle. Quotes are removed automatically after parsing.

The return value is a standard `list`:

```
>>> parse_list_header('token, "quoted value"')
['token', 'quoted value']
```

Parameters
`value` a string with a list header.

Returns
`list`

`parse_options_header`(*value*)

Parse Content-Type-like header into a tuple.

Parse a `Content-Type` like header into a tuple with the content type and the options:

```
>>> parse_options_header('Content-Type: text/html; mimetype=text/html')
('Content-Type:', {'mimetype': 'text/html'})
```

Parameters
`value` the header to parse.

Returns
(str, options)

`unquote_header_value`(*value*)

Unquotes a header value.

This does not use the real unquoting but what browsers are actually using for quoting.

Parameters
`value` the header value to unquote.

`urlmap_factory`(*loader*, *global_conf*, ***local_conf*)

cinder.api.versions module

`class Versions`(*ext_mgr=None*)

Bases: *APIRouter*

Route versions requests.

`class ExtensionManager`

Bases: `object`

Load extensions from the configured extension path.

See cinder/tests/api/extensions/foxinsocks/extension.py for an example extension implementation.

`get_controller_extensions`()

Returns a list of ControllerExtension objects.

`get_resources`()

Returns a list of ResourceExtension objects.

`is_loaded`(*alias*)

load_extension(*ext_factory*)

Execute an extension factory.

Loads an extension. The ext_factory is the name of a callable that will be imported and called with one argumentthe extension manager. The factory callable is expected to call the register() method at least once.

register(*ext*)

class VersionsController

Bases: *Controller*

all(*req*)

Return all known and enabled versions.

index(*req*)

Return versions supported after the start of microversions.

versioned_methods = {'index': [<cinder.api.openstack.versioned_method.VersionedMethod object>]}

wsgi_actions = {}

wsgi_extensions = []

create_resource()

Module contents

root_app_factory(*loader*, *global_conf*, ***local_conf*)

cinder.backup package

Submodules

cinder.backup.api module

Handles all requests relating to the volume backups service.

class API

Bases: *Base*

API for interacting with the volume backup manager.

create(*context: RequestContext*, *name: str | None*, *description: str | None*, *volume_id: str*, *container: str*, *incremental: bool = False*, *availability_zone: str | None = None*, *force: bool = False*, *snapshot_id: str | None = None*, *metadata: dict | None = None*) → *Backup*

Make the RPC call to create a volume backup.

delete(*context: RequestContext*, *backup: Backup*, *force: bool = False*) → None

Make the RPC call to delete a volume backup.

Call backup manager to execute backup delete or force delete operation. :param context: running context :param backup: the dict of backup that is got from DB. :param force: indicate force delete or not :raises InvalidBackup: :raises BackupDriverException: :raises ServiceNotFound:

export_record(*context: RequestContext*, *backup_id: str*) → dict

Make the RPC call to export a volume backup.

Call backup manager to execute backup export.

Parameters

- **context** running context
- **backup_id** backup id to export

Returns

dictionary a description of how to import the backup

Returns

contains backup_url and backup_service

Raises

InvalidBackup

get(*context: RequestContext*, *backup_id: str*) → *Backup*

get_all(*context: RequestContext*, *search_opts: dict | None = None*, *marker: str | None = None*, *limit: int | None = None*, *offset: int | None = None*, *sort_keys: list[str] | None = None*, *sort_dirs: list[str] | None = None*) → *BackupList*

get_available_backup_service_host(*host: str*, *az: str*) → str

import_record(*context: RequestContext*, *backup_service: str*, *backup_url: str*) → *Backup*

Make the RPC call to import a volume backup.

Parameters

- **context** running context
- **backup_service** backup service name
- **backup_url** backup description to be used by the backup driver

Raises

- *InvalidBackup*
- *ServiceNotFound*
- *InvalidInput*

reset_status(*context: RequestContext*, *backup_id: str*, *status: str*) → None

Make the RPC call to reset a volume backups status.

Call backup manager to execute backup status reset operation. :param context: running context :param backup_id: which backups status to be reset :param status: backups status to be reset :raises InvalidBackup:

restore(*context: RequestContext*, *backup_id: str*, *volume_id: str | None = None*, *name: str | None = None*) → dict

Make the RPC call to restore a volume backup.

update(*context: RequestContext*, *backup_id: str*, *fields: list*) → *Service*

cinder.backup.chunkeddriver module

Generic base class to implement metadata, compression and chunked data operations

class ChunkedBackupDriver(*context*, *chunk_size_bytes*, *sha_block_size_bytes*, *backup_default_container*, *enable_progress_timer*)

Bases: *BackupDriver*

Abstract chunked backup driver.

Implements common functionality for backup drivers that store volume data in multiple chunks in a backup repository when the size of the backed up cinder volume exceeds the size of a backup repository chunk.

Provides abstract methods to be implemented in concrete chunking drivers.

DRIVER_VERSION = '1.0.0'

DRIVER_VERSION_MAPPING = {'1.0.0': '_restore_v1'}

backup(*backup*, *volume_file*, *backup_metadata=True*)

Backup the given volume.

If backup[parent_id] is given, then an incremental backup is performed.

delete_backup(*backup*)

Delete the given backup.

abstract delete_object(*container*, *object_name*)

Delete object from container.

abstract get_container_entries(*container*, *prefix*)

Get container entry names.

abstract get_extra_metadata(*backup*, *volume*)

Return extra metadata to use in prepare_backup.

This method allows for collection of extra metadata in prepare_backup() which will be passed to get_object_reader() and get_object_writer(). Subclass extensions can use this extra information to optimize data transfers. Return a json serializable object.

abstract get_object_reader(*container*, *object_name*, *extra_metadata=None*)

Returns a reader object for the backed up chunk.

The object reader methods must not have any logging calls, as eventlet has a bug (https://github.com/eventlet/eventlet/issues/432) that would result in failures.

abstract get_object_writer(*container*, *object_name*, *extra_metadata=None*)

Returns a writer object which stores the chunk data.

The object returned should be a context handler that can be used in a with context.

The object writer methods must not have any logging calls, as eventlet has a bug (https://github.com/eventlet/eventlet/issues/432) that would result in failures.

abstract put_container(*container*)

Create the container if needed. No failure if it pre-exists.

restore(*backup*, *volume_id*, *volume_file*, *volume_is_new*)

Restore the given volume backup from backup repository.

Raises BackupRestoreCancel on any backup status change.

abstract update_container_name(*backup*, *container*)

Allow sub-classes to override container name.

This method exists so that sub-classes can override the container name as it comes in to the driver in the backup object. Implementations should return None if no change to the container name is desired.

cinder.backup.driver module

Base class for all backup drivers.

class BackupDriver(*context*)

Bases: `Base`

abstract backup(*backup*, *volume_file*, *backup_metadata=False*)

Start a backup of a specified volume.

Some I/O operations may block greenthreads, so in order to prevent starvation parameter volume_file will be a proxy that will execute all methods in native threads, so the method implementation doesnt need to worry about that..

check_for_setup_error()

Method for checking if backup backend is successfully installed.

Refer to `cinder.interface.backup_driver.BackupDriver.check_for_setup_error` for additional information.

abstract delete_backup(*backup*)

Delete a saved backup.

export_record(*backup*)

Export driver specific backup record information.

If backup backend needs additional driver specific information to import backup record back into the system it must overwrite this method and return it here as a dictionary so it can be serialized into a string.

Default backup driver implementation has no extra information.

Parameters
backup backup object to export

Returns
driver_info - dictionary with extra information

get_metadata(*volume_id*)

import_record(*backup*, *driver_info*)

Import driver specific backup record information.

If backup backend needs additional driver specific information to import backup record back into the system it must overwrite this method since it will be called with the extra information that was provided by export_record when exporting the backup.

Default backup driver implementation does nothing since it didnt export any specific data in export_record.

Parameters

- **backup** backup object to export
- **driver_info** dictionary with driver specific backup record information

Returns
nothing

`put_metadata`(*volume_id*, *json_metadata*)

`abstract restore`(*backup*, *volume_id*, *volume_file*, *volume_is_new*)

Restore a saved backup.

Some I/O operations may block greenthreads, so in order to prevent starvation parameter volume_file will be a proxy that will execute all methods in native threads, so the method implementation doesnt need to worry about that..

May raise BackupRestoreCancel to indicate that the restoration of a volume has been aborted by changing the backup status.

`class BackupMetadataAPI`(*context*)

Bases: `Base`

`TYPE_TAG_VOL_BASE_META = 'volume-base-metadata'`

`TYPE_TAG_VOL_GLANCE_META = 'volume-glance-metadata'`

`TYPE_TAG_VOL_META = 'volume-metadata'`

`get`(*volume_id*)

Get volume metadata.

Returns a json-encoded dict containing all metadata and the restore version i.e. the version used to decide what actually gets restored from this container when doing a backup restore.

`put`(*volume_id*, *json_metadata*)

Restore volume metadata to a volume.

The json container should contain a version that is supported here.

cinder.backup.manager module

Backup manager manages volume backups.

Volume Backups are full copies of persistent volumes stored in a backup store e.g. an object store or any other backup store if and when support is added. They are usable without the original object being available. A volume backup can be restored to the original volume it was created from or any other available volume with a minimum size of the original volume. Volume backups can be created, restored, deleted and listed.

Related Flags

backup_manager
The module name of a class derived from `manager.Manager` (default: `cinder.backup.manager.Manager`).

class BackupManager(**args*, ***kwargs*)

Bases: *SchedulerDependentManager*

Manages backup of block storage devices.

RPC_API_VERSION = '2.4'

check_support_to_force_delete(*context*)

Check if the backup driver supports force delete operation.

Parameters
context running context

continue_backup(*context*, *backup*, *backup_device*)

This is the callback from the volume manager to continue.

create_backup(*context*, *backup*)

Create volume backups using configured backup service.

delete_backup(*context*, *backup*)

Delete volume backup from configured backup service.

export_record(*context*, *backup*)

Export all volume backup metadata details to allow clean import.

Export backup metadata so it could be re-imported into the database without any prerequisite in the backup database.

Parameters

- **context** running context
- **backup** backup object to export

Returns
backup_record - a description of how to import the backup

Returns
contains backup_url - how to import the backup, and

Returns
backup_service describing the needed driver.

Raises
InvalidBackup

import_record(*context*, *backup*, *backup_service*, *backup_url*, *backup_hosts*)

Import all volume backup metadata details to the backup db.

Parameters

- **context** running context
- **backup** The new backup object for the import
- **backup_service** The needed backup driver for import
- **backup_url** An identifier string to locate the backup
- **backup_hosts** Potential hosts to execute the import

Raises

- *InvalidBackup*
- *ServiceNotFound*

init_host(***kwargs*)

Run initialization needed for a standalone service.

is_working()

Method indicating if service is working correctly.

This method is supposed to be overridden by subclasses and return if manager is working correctly.

publish_service_capabilities(*context*)

Collect driver status and then publish.

reset()

Method executed when SIGHUP is caught by the process.

Were utilizing it to reset RPC API version pins to avoid restart of the service when rolling upgrade is completed.

reset_status(*context*, *backup*, *status*)

Reset volume backup status.

Parameters

- **context** running context
- **backup** The backup object for reset status operation
- **status** The status to be set

Raises

- *InvalidBackup*
- **AttributeError**

restore_backup(*context*, *backup*, *volume_id*, *volume_is_new*)

Restore volume backups from configured backup service.

Parameters

- **context** RequestContext for the restore operation
- **backup** Backup that were restoring
- **volume_id** The ID of the volume into which were restoring
- **volume_is_new** The volume does not have stale data, so sparse backups can be restored as such.

setup_backup_backend(*ctxt*)

target = <Target version=2.4>

cinder.backup.rpcapi module

Client side of the volume backup RPC API.

class BackupAPI

Bases: RPCAPI

Client side of the volume rpc API.

API version history:

```
1.0 - Initial version.
1.1 - Changed methods to accept backup objects instead of IDs.
1.2 - A version that got in by mistake (without breaking anything).
1.3 - Dummy version bump to mark start of having cinder-backup service
      decoupled from cinder-volume.

... Mitaka supports messaging 1.3. Any changes to existing methods in
1.x after this point should be done so that they can handle version cap
set to 1.3.

2.0 - Remove 1.x compatibility
2.1 - Adds set_log_levels and get_log_levels
2.2 - Adds publish_service_capabilities
2.3 - Adds continue_backup call
2.4 - Add the volume_is_new flag to the restore_backup method
```

BINARY = 'cinder-backup'

RPC_API_VERSION = '2.4'

RPC_DEFAULT_VERSION = '2.0'

TOPIC = 'cinder-backup'

check_support_to_force_delete(*ctxt*, *host*) → bool

continue_backup(*ctxt*, *backup*, *backup_device*)

create_backup(*ctxt*, *backup*)

delete_backup(*ctxt*, *backup*)

export_record(*ctxt*, *backup*) → dict

get_log_levels(*context*, *service*, *log_request*)

import_record(*ctxt*, *host*, *backup*, *backup_service*, *backup_url*, *backup_hosts*) → None

publish_service_capabilities(*ctxt*)

reset_status(*ctxt*, *backup*, *status*)

restore_backup(*ctxt*, *backup_host*, *backup*, *volume_id*, *volume_is_new*)

set_log_levels(*context*, *service*, *log_request*)

Module contents

`API`(**args*, ***kwargs*)

cinder.brick package

Subpackages

cinder.brick.local_dev package

Submodules

cinder.brick.local_dev.lvm module

LVM class for performing LVM operations.

class LVM(*vg_name*, *root_helper*, *create_vg=False*, *physical_volumes=None*, *lvm_type='default'*, *executor=<function execute>*, *lvm_conf=None*, *suppress_fd_warn=False*)

Bases: `Executor`

LVM object to enable various LVM related operations.

LVM_CMD_PREFIX = ['env', 'LC_ALL=C']

activate_lv(*name*, *is_snapshot=False*, *permanent=False*)

Ensure that logical volume/snapshot logical volume is activated.

Parameters

- **name** Name of LV to activate
- **is_snapshot** whether LV is a snapshot
- **permanent** whether we should drop skipactivation flag

Raises

putils.ProcessExecutionError

create_lv_snapshot(*name*, *source_lv_name*, *lv_type='default'*)

Creates a snapshot of a logical volume.

Parameters

- **name** Name to assign to new snapshot
- **source_lv_name** Name of Logical Volume to snapshot
- **lv_type** Type of LV (default or thin)

create_thin_pool(*name=None*, *size_str=None*)

Creates a thin provisioning pool for this VG.

The syntax here is slightly different than the default lvcreate -T, so well just write a custom cmd here and do it.

Parameters

- **name** Name to use for pool, default is <vg-name>-pool
- **size_str** Size to allocate for pool, default is entire VG

Returns
The size string passed to the lvcreate command

`create_volume`(*name*, *size_str*, *lv_type='default'*, *mirror_count=0*)
Creates a logical volume on the objects VG.

Parameters

- `name` Name to use when creating Logical Volume
- `size_str` Size to use when creating Logical Volume
- `lv_type` Type of Volume (default or thin)
- `mirror_count` Use LVM mirroring with specified count

`deactivate_lv`(*name*)

`delete`(*name*)
Delete logical volume or snapshot.

Parameters
`name` Name of LV to delete

`extend_volume`(*lv_name*, *new_size*)
Extend the size of an existing volume.

static `get_all_physical_volumes`(*root_helper*, *vg_name=None*)
Static method to get all PVs on a system.

Parameters

- `root_helper` root_helper to use for execute
- `vg_name` optional, gathers info for only the specified VG

Returns
List of Dictionaries with PV info

static `get_all_volume_groups`(*root_helper*, *vg_name=None*)
Static method to get all VGs on a system.

Parameters

- `root_helper` root_helper to use for execute
- `vg_name` optional, gathers info for only the specified VG

Returns
List of Dictionaries with VG info

static `get_lv_info`(*root_helper*, *vg_name=None*, *lv_name=None*)
Retrieve info about LVs (all, in a VG, or a single LV).

Parameters

- `root_helper` root_helper to use for execute
- `vg_name` optional, gathers info for only the specified VG
- `lv_name` optional, gathers info for only the specified LV

Returns
List of Dictionaries with LV info

`static get_lvm_version(`*root_helper*`)`

Static method to get LVM version from system.

Parameters
`root_helper` root_helper to use for execute

Returns
version 3-tuple

`get_volume(`*name*`)`

Get reference object of volume specified by name.

Returns
dict representation of Logical Volume if exists

`get_volumes(`*lv_name=None*`)`

Get all LVs associated with this instantiation (VG).

Returns
List of Dictionaries with LV info

`lv_get_origin(`*name*`)`

Return the origin of an LV that is a snapshot, None otherwise.

`lv_has_snapshot(`*name*`)`

`lv_is_open(`*name*`)`

Return True if LV is currently open, False otherwise.

`lv_is_snapshot(`*name*`)`

Return True if LV is a snapshot, False otherwise.

`rename_volume(`*lv_name*, *new_name*`)`

Change the name of an existing volume.

`revert(`*snapshot_name*`)`

Revert an LV to snapshot.

Parameters
`snapshot_name` Name of snapshot to revert

`property supports_lvchange_ignoreskipactivation`

Property indicating whether lvchange can ignore skip activation.

Check for LVM version >= 2.02.99. (LVM2 git: ab789c1bc add ignoreactivationskip to lvchange)

`static supports_pvs_ignoreskippedcluster(`*root_helper*`)`

Property indicating whether pvs supports ignoreskippedcluster

Check for LVM version >= 2.02.103. (LVM2 git: baf95bbff cmdline: Add ignoreskipped-cluster.

property supports_snapshot_lv_activation

Property indicating whether snap activation changes are supported.

Check for LVM version >= 2.02.91. (LVM2 git: e8a40f6 Allow to activate snapshot)

Returns
True/False indicating support

static supports_thin_provisioning(*root_helper*)

Static method to check for thin LVM support on a system.

Parameters
root_helper root_helper to use for execute

Returns
True if supported, False otherwise

update_volume_group_info()

Update VG info for this instantiation.

Used to update member fields of object and provide a dict of info for caller.

Returns
Dictionaries of VG info

vg_mirror_free_space(*mirror_count*)

vg_mirror_size(*mirror_count*)

Module contents

Module contents

cinder.cmd package

Submodules

cinder.cmd.api module

Starter script for Cinder OS API.

main() → None

cinder.cmd.backup module

Starter script for Cinder Volume Backup.

main() → None

cinder.cmd.manage module

CLI interface for cinder management.

class BackupCommands

Bases: `object`

Methods for managing backups.

`list()` → None

List all backups.

List all backups (including ones in progress) and the host on which the backup operation is running.

`update_backup_host(currenthost: str, newhost: str)` → None

Modify the host name associated with a backup.

Particularly to recover from cases where one has moved their Cinder Backup node, and not set backup_use_same_backend.

`class BaseCommand`

Bases: `object`

`class ClusterCommands`

Bases: `BaseCommand`

Methods for managing clusters.

`list()` → None

Show a list of all cinder services.

`remove(recursive: bool, binary: str, cluster_name: str)` → int | None

Completely removes a cluster.

`rename(partial: bool, current: str | None, new: str | None)` → int | None

Rename cluster name for Volumes and Consistency Groups.

Useful when you want to rename a cluster, particularly when the backend_name has been modified in a multi-backend config or we have moved from a single backend to multi-backend.

`class ConfigCommands`

Bases: `object`

Class for exposing the flags defined by flag_file(s).

`list(param: str | None = None)` → None

List parameters configured for cinder.

Lists all parameters configured for cinder unless an optional argument is specified. If the parameter is specified we only print the requested parameter. If the parameter is not found an appropriate error is produced by .get*().

`class ConsistencyGroupCommands`

Bases: `object`

Methods for managing consistency groups.

`update_cg_host(currenthost: str, newhost: str)` → None

Modify the host name associated with a Consistency Group.

Particularly to recover from cases where one has moved a host from single backend to multi-backend, or changed the host configuration option, or modified the backend_name in a multi-backend config.

`class DbCommands`

Bases: `object`

Class for managing the database.

`online_data_migrations`(*max_count: int | None = None*) → None

Perform online data migrations for the release in batches.

`online_migrations: Tuple[Callable[[RequestContext, int], Tuple[int, int]], ...] = (<function remove_temporary_admin_metadata_data_migration>,)`

`purge`(*age_in_days: int*) → None

Purge deleted rows older than a given age from cinder tables.

`reset_active_backend`(*enable_replication: bool*, *active_backend_id: str | None*, *backend_host: str*) → None

Reset the active backend for a host.

`sync`(*version: int | None = None*, *bump_versions: bool = False*) → None

Sync the database up to the most recent version.

`version`() → None

Print the current database version.

`class HostCommands`

Bases: `object`

List hosts.

`list`(*zone: str | None = None*) → None

Show a list of all physical hosts.

Can be filtered by zone. args: [zone]

`class QuotaCommands`

Bases: `object`

Class for managing quota issues.

`check`(*project_id: str | None*) → None

Check if quotas and reservations are correct

This action checks quotas and reservations, for a specific project or for all projects, to see if they are out of sync.

The check will also look for duplicated entries.

One way to use this check in combination with the sync action is to run the check for all projects, take note of those that are out of sync, and then sync them one by one at intervals to reduce stress on the DB.

`sync`(*project_id: str | None*) → None

Fix quotas and reservations

This action refreshes existing quota usage and reservation count for a specific project or for all projects.

The refresh will also remove duplicated entries.

This operation is best executed when Cinder is not running, but it can be run with cinder services running as well.

A different transaction is used for each projects quota sync, so an action failure will only rollback the current projects changes.

class ServiceCommands

Bases: *BaseCommand*

Methods for managing services.

list()

Show a list of all cinder services.

remove(*binary: str*, *host_name: str*) → int | None

Completely removes a service.

class UtilCommands

Bases: `object`

Generic utils.

clean_locks(*online: bool*) → None

Clean file locks for vols, snaps, and backups on the current host.

Should be run on any host where we are running a Cinder service (API, Scheduler, Volume, Backup) and can be run with the Cinder services running or stopped.

If the services are running it will check existing resources in the Cinder database in order to know which resources are still available (its not safe to remove their file locks) and will only remove the file locks for the resources that are no longer present. Deleting locks while the services are offline is faster as theres no need to check the database.

For backups, the way to know if we can remove the startup lock is by checking if the PGRP in the file name is currently running cinder-backup.

Default assumes that services are online, must pass `--services-offline` to specify that they are offline.

Doesnt clean DLM locks (except when using file locks), as those dont leave lock leftovers.

class VersionCommands

Bases: `object`

Class for exposing the codebase version.

list()

class VolumeCommands

Bases: `object`

Methods for dealing with a cloud in an odd state.

delete(*volume_id: str*) → None

Delete a volume, bypassing the check that it must be available.

update_host(*currenthost: str*, *newhost: str*) → None

Modify the host name associated with a volume.

Particularly to recover from cases where one has moved their Cinder Volume node, or modified their backend_name in a multi-backend config.

update_service()

Modify the service uuid associated with a volume.

In certain upgrade cases, we create new cinder services and delete the records of old ones, however, the volumes created with old service still contain the service uuid of the old services.

add_command_parsers(*subparsers*)

args(**args*, ***kwargs*)

fetch_func_args(*func*)

get_arg_string(*args*)

main()

methods_of(*obj*) → list

Return non-private methods from an object.

Get all callable methods of an object that dont start with underscore :return: a list of tuples of the form (method_name, method)

missing_action(*help_func: Callable*) → Callable

cinder.cmd.rtstool module

exception RtstoolError

Bases: `Exception`

exception RtstoolImportError

Bases: `RtstoolError`

add_initiator(*target_iqn*, *initiator_iqn*, *userid*, *password*)

create(*backing_device*, *name*, *userid*, *password*, *iser_enabled*, *initiator_iqns=None*, *portals_ips=None*, *portals_port=3260*)

delete(*iqn*)

delete_initiator(*target_iqn*, *initiator_iqn*)

get_targets()

main(*argv=None*)

parse_optional_create(*argv*)

restore_from_file(*configuration_file*)

save_to_file(*destination_file*)

`usage()`

`verify_rtslib()`

cinder.cmd.scheduler module

Starter script for Cinder Scheduler.

`main()` → None

cinder.cmd.status module

CLI interface for cinder status commands.

`class Checks(*args, **kwargs)`

Bases: `UpgradeCommands`

Upgrade checks to run.

`main()`

cinder.cmd.volume module

Starter script for Cinder Volume.

`main()` → None

cinder.cmd.volume_usage_audit module

Cron script to generate usage notifications for volumes existing during the audit period.

Together with the notifications generated by volumes create/delete/resize, over that time period, this allows an external system consuming usage notification feeds to calculate volume usage for each tenant.

Time periods are specified as hour, month, day or year

- *hour* - previous hour. If run at 9:07am, will generate usage for 8-9am.
- *month* - previous month. If the script is run April 1, it will generate usages for March 1 through March 31.
- *day* - previous day. if run on July 4th, it generates usages for July 3rd.
- *year* - previous year. If run on Jan 1, it generates usages for Jan 1 through Dec 31 of the previous year.

`main()`

Module contents

cinder.common package

Submodules

cinder.common.config module

Command-line flag library.

Emulates gflags by wrapping cfg.ConfigOpts.

The idea is to move fully to cfg eventually, and this wrapper is a stepping stone.

set_external_library_defaults()

Set default configuration options for external openstack libraries.

set_middleware_defaults()

Update default configuration options for oslo.middleware.

cinder.common.constants module

cinder.common.sqlalchemyutils module

Implementation of paginate query.

paginate_query(*query*, *model*, *limit*, *sort_keys*, *marker=None*, *sort_dir=None*, *sort_dirs=None*, *offset=None*)

Returns a query with sorting / pagination criteria added.

Pagination works by requiring a unique sort_key, specified by sort_keys. (If sort_keys is not unique, then we risk looping through values.) We use the last row in the previous page as the marker for pagination. So we must return values that follow the passed marker in the order. With a single-valued sort_key, this would be easy: sort_key > X. With a compound-values sort_key, (k1, k2, k3) we must do this to repeat the lexicographical ordering: (k1 > X1) or (k1 == X1 && k2 > X2) or (k1 == X1 && k2 == X2 && k3 > X3)

We also have to cope with different sort_directions.

Typically, the id of the last row is used as the client-facing pagination marker, then the actual marker object must be fetched from the db and passed in to us as marker.

Parameters

- **query** the query object to which we should add paging/sorting
- **model** the ORM model class
- **limit** maximum number of items to return
- **sort_keys** array of attributes by which results should be sorted
- **marker** the last item of the previous page; we returns the next results after this value.
- **sort_dir** direction in which results should be sorted (asc, desc)
- **sort_dirs** per-column array of sort_dirs, corresponding to sort_keys
- **offset** the number of items to skip from the marker or from the first element.

Return type

sqlalchemy.orm.query.Query

Returns

The query with sorting/pagination added.

Module contents

cinder.compute package

Submodules

cinder.compute.nova module

Handles all requests to Nova.

class API

Bases: *Base*

API for interacting with novaclient.

exception NotFound(*code*, *message=None*, *details=None*, *request_id=None*, *url=None*, *method=None*)

Bases: `ClientException`

HTTP 404 - Not found

http_status = 404

message = 'Not found'

create_volume_snapshot(*context*, *volume_id*, *create_info*)

delete_volume_snapshot(*context*, *snapshot_id*, *delete_info*)

extend_volume(*context*, *server_ids*, *volume_id*)

get_server(*context*, *server_id*, *privileged_user=False*, *timeout=None*)

static get_server_volume(*context*, *server_id*, *volume_id*)

reimage_volume(*context*, *server_ids*, *volume_id*)

update_server_volume(*context*, *server_id*, *src_volid*, *new_volume_id*)

novaclient(*context*, *privileged_user=False*, *timeout=None*, *api_version=None*)

Returns a Nova client

@param privileged_user:
If True, use the account from configuration (requires auth_type and the other usual Keystone authentication options to be set in the [nova] section)

@param timeout:
Number of seconds to wait for an answer before raising a Timeout exception (None to disable)

@param api_version:
api version of nova

Module contents

API()

cinder.db package

Submodules

cinder.db.api module

Defines interface for DB access.

Functions in this module are imported into the cinder.db namespace. Call these functions from cinder.db namespace, not the cinder.db.api namespace.

All functions in this module return objects that implement a dictionary-like interface. Currently, many of these objects are sqlalchemy objects that implement a dictionary interface. However, a future goal is to have all of these objects be simple dictionaries.

Related Flags

connection
string specifying the sqlalchemy connection to use, like: *sqlite:///var/lib/cinder/cinder.sqlite*.

enable_new_services
when adding a new service to the database, is it in the pool of available hardware (Default: True)

`class Case`(*whens*, *value=None*, *else_=None*)

Bases: `object`

Class for conditional value selection for conditional_update.

`class Condition`(*value*, *field=None*)

Bases: `object`

Class for normal condition values for conditional_update.

`get_filter`(*model*, *field=None*)

`class Not`(*value*, *field=None*, *auto_none=True*)

Bases: `Condition`

Class for negated condition values for conditional_update.

By default NULL values will be treated like Python treats None instead of how SQL treats it.

So for example when values are (1, 2) it will evaluate to True when we have value 3 or NULL, instead of only with 3 like SQL does.

`get_filter`(*model*, *field=None*)

`attachment_destroy`(*context*, *attachment_id*)

Destroy the attachment or raise if it does not exist.

`attachment_specs_delete`(*context*, *attachment_id*, *key*)

DEPRECATED: Delete the given attachment specs item.

`attachment_specs_exist`(*context*)

Check if there are attachment specs left.

attachment_specs_get(*context*, *attachment_id*)
DEPRECATED: Get all specs for an attachment.

attachment_specs_update_or_create(*context*, *attachment_id*, *specs*)
DEPRECATED: Create or update attachment specs.

This adds or modifies the key/value pairs specified in the attachment specs dict argument.

backup_create(*context*, *values*)
Create a backup from the values dictionary.

backup_destroy(*context*, *backup_id*)
Destroy the backup or raise if it does not exist.

backup_get(*context*, *backup_id*, *read_deleted=None*, *project_only=True*)
Get a backup or raise if it does not exist.

backup_get_all(*context*, *filters=None*, *marker=None*, *limit=None*, *offset=None*, *sort_keys=None*, *sort_dirs=None*)
Get all backups.

backup_get_all_active_by_window(*context*, *begin*, *end=None*, *project_id=None*)
Get all the backups inside the window.

Specifying a project_id will filter for a certain project.

backup_get_all_by_host(*context*, *host*)
Get all backups belonging to a host.

backup_get_all_by_project(*context*, *project_id*, *filters=None*, *marker=None*, *limit=None*, *offset=None*, *sort_keys=None*, *sort_dirs=None*)
Get all backups belonging to a project.

backup_get_all_by_volume(*context*, *volume_id*, *vol_project_id*, *filters=None*)
Get all backups belonging to a volume.

backup_metadata_get(*context*, *backup_id*)

backup_metadata_update(*context*, *backup_id*, *metadata*, *delete*)

backup_update(*context*, *backup_id*, *values*)
Set the given properties on a backup and update it.

Raises NotFound if backup does not exist.

calculate_resource_count(*context*, *resource_type*, *filters*)

cg_creating_from_src(*cg_id=None*, *cgsnapshot_id=None*)
Return a filter to check if a CG is being used as creation source.

Returned filter is meant to be used in the Conditional Update mechanism and checks if provided CG ID or CG Snapshot ID is currently being used to create another CG.

This filter will not include CGs that have used the ID but have already finished their creation (status is no longer creating).

Filter uses a subquery that allows it to be used on updates to the consistencygroups table.

`cg_has_cgsnapshot_filter`()
Return a filter that checks if a CG has CG Snapshots.

`cg_has_volumes_filter`(*attached_or_with_snapshots=False*)
Return a filter to check if a CG has volumes.

When attached_or_with_snapshots parameter is given a True value only attached volumes or those with snapshots will be considered.

`cgsnapshot_create`(*context*, *values*)
Create a cgsnapshot from the values dictionary.

`cgsnapshot_creating_from_src`()
Get a filter that checks if a CGSnapshot is being created from a CG.

`cgsnapshot_destroy`(*context*, *cgsnapshot_id*)
Destroy the cgsnapshot or raise if it does not exist.

`cgsnapshot_get`(*context*, *cgsnapshot_id*)
Get a cgsnapshot or raise if it does not exist.

`cgsnapshot_get_all`(*context*, *filters=None*)
Get all cgsnapshots.

`cgsnapshot_get_all_by_group`(*context*, *group_id*, *filters=None*)
Get all cgsnapshots belonging to a consistency group.

`cgsnapshot_get_all_by_project`(*context*, *project_id*, *filters=None*)
Get all cgsnapshots belonging to a project.

`cgsnapshot_update`(*context*, *cgsnapshot_id*, *values*)
Set the given properties on a cgsnapshot and update it.

Raises NotFound if cgsnapshot does not exist.

`cleanup_expired_messages`(*context*)
Soft delete expired messages

`cluster_create`(*context*, *values*)
Create a cluster from the values dictionary.

`cluster_destroy`(*context*, *cluster_id*)
Destroy the cluster or raise if it does not exist or has hosts.

Raises
`ClusterNotFound` If cluster doesnt exist.

`cluster_get`(*context*, *id=None*, *is_up=None*, *get_services=False*, *services_summary=False*, *read_deleted='no'*, *name_match_level=None*, ***filters*)
Get a cluster that matches the criteria.

Parameters

- `id` Id of the cluster.
- `is_up` Boolean value to filter based on the clusters up status.
- `get_services` If we want to load all services from this cluster.

- **services_summary** If we want to load num_hosts and num_down_hosts fields.
- **read_deleted** Filtering based on delete status. Default value is no.
- **name_match_level** pool, backend, or host for name filter (as defined in _filter_host method)
- **filters** Field based filters in the form of key/value.

Raises
ClusterNotFound If cluster doesnt exist.

cluster_get_all(*context*, *is_up=None*, *get_services=False*, *services_summary=False*, *read_deleted='no'*, *name_match_level=None*, ***filters*)

Get all clusters that match the criteria.

Parameters

- **is_up** Boolean value to filter based on the clusters up status.
- **get_services** If we want to load all services from this cluster.
- **services_summary** If we want to load num_hosts and num_down_hosts fields.
- **read_deleted** Filtering based on delete status. Default value is no.
- **name_match_level** pool, backend, or host for name filter (as defined in _filter_host method)
- **filters** Field based filters in the form of key/value.

cluster_update(*context*, *cluster_id*, *values*)

Set the given properties on an cluster and update it.

Raises ClusterNotFound if cluster does not exist.

conditional_update(*context*, *model*, *values*, *expected_values*, *filters=None*, *include_deleted='no'*, *project_only=False*, *order=None*)

Compare-and-swap conditional update.

Update will only occur in the DB if conditions are met.

We have 4 different condition types we can use in expected_values:

- Equality: {status: available}
- Inequality: {status: vol_obj.Not(deleting)}
- In range: {status: [available, error]
- Not in range: {status: vol_obj.Not([in-use, attaching])

Method accepts additional filters, which are basically anything that can be passed to a sqlalchemy querys filter method, for example:

```
[~sql.exists().where(models.Volume.id == models.Snapshot.volume_id)]
```

We can select values based on conditions using Case objects in the values argument. For example:

```
has_snapshot_filter = sql.exists().where(
    models.Snapshot.volume_id == models.Volume.id
)
case_values = db.Case(
    [(has_snapshot_filter, 'has-snapshot')],
    else_='no-snapshot'
)
db.conditional_update(
    context,
    models.Volume,
    {'status': case_values},
    {'status': 'available'},
)
```

And we can use DB fields for example to store previous status in the corresponding field even though we dont know which value is in the db from those we allowed:

```
db.conditional_update(
    context,
    models.Volume,
    {'status': 'deleting', 'previous_status': models.Volume.status},
    {'status': ('available', 'error')},
)
```

Parameters

- **`values`** Dictionary of key-values to update in the DB.
- **`expected_values`** Dictionary of conditions that must be met for the update to be executed.
- **`filters`** Iterable with additional filters.
- **`include_deleted`** Should the update include deleted items, this is equivalent to read_deleted.
- **`project_only`** Should the query be limited to contexts project.
- **`order`** Specific order of fields in which to update the values

Returns

Boolean indicating whether db rows were updated.

`consistencygroup_create`(*context*, *values*, *cg_snap_id=None*, *cg_id=None*)

Create a consistencygroup from the values dictionary.

`consistencygroup_destroy`(*context*, *consistencygroup_id*)

Destroy the consistencygroup or raise if it does not exist.

`consistencygroup_get`(*context*, *consistencygroup_id*)

Get a consistencygroup or raise if it does not exist.

`consistencygroup_get_all`(*context*, *filters=None*, *marker=None*, *limit=None*, *offset=None*, *sort_keys=None*, *sort_dirs=None*)

Get all consistencygroups.

consistencygroup_get_all_by_project(*context*, *project_id*, *filters=None*, *marker=None*, *limit=None*, *offset=None*, *sort_keys=None*, *sort_dirs=None*)

Get all consistencygroups belonging to a project.

consistencygroup_include_in_cluster(*context*, *cluster*, *partial_rename=True*, ***filters*)

Include all consistency groups matching the filters into a cluster.

When partial_rename is set we will not set the cluster_name with cluster parameter value directly, well replace provided cluster_name or host filter value with cluster instead.

This is useful when we want to replace just the cluster name but leave the backend and pool information as it is. If we are using cluster_name to filter, well use that same DB field to replace the cluster value and leave the rest as it is. Likewise if we use the host to filter.

Returns the number of consistency groups that have been changed.

consistencygroup_update(*context*, *consistencygroup_id*, *values*)

Set the given properties on a consistencygroup and update it.

Raises NotFound if consistencygroup does not exist.

dispose_engine()

Force the engine to establish new connections.

driver_initiator_data_get(*context*, *initiator*, *namespace*)

Query for an DriverInitiatorData that has the specified key

driver_initiator_data_insert_by_key(*context*, *initiator*, *namespace*, *key*, *value*)

Updates DriverInitiatorData entry.

Sets the value for the specified key within the namespace.

get_all_projects_with_default_type(*context*, *volume_type_id*)

Get all the projects associated with a default type

get_booleans_for_table(*table_name*)

get_by_id(*context*, *model*, *id*, **args*, ***kwargs*)

get_model_for_versioned_object(*versioned_object*)

get_projects(*context*, *model*, *read_deleted='no'*)

get_snapshot_summary(*context*, *project_only*, *filters=None*)

Get snapshot summary.

get_volume_summary(*context*, *project_only*, *filters=None*)

Get volume summary.

group_create(*context*, *values*, *group_snapshot_id=None*, *source_group_id=None*)

Create a group from the values dictionary.

group_creating_from_src(*group_id=None*, *group_snapshot_id=None*)

Return a filter to check if a Group is being used as creation source.

Returned filter is meant to be used in the Conditional Update mechanism and checks if provided Group ID or Group Snapshot ID is currently being used to create another Group.

This filter will not include Groups that have used the ID but have already finished their creation (status is no longer creating).

Filter uses a subquery that allows it to be used on updates to the groups table.

`group_destroy`(*context*, *group_id*)

Destroy the group or raise if it does not exist.

`group_get`(*context*, *group_id*)

Get a group or raise if it does not exist.

`group_get_all`(*context*, *filters=None*, *marker=None*, *limit=None*, *offset=None*, *sort_keys=None*, *sort_dirs=None*)

Get all groups.

`group_get_all_by_project`(*context*, *project_id*, *filters=None*, *marker=None*, *limit=None*, *offset=None*, *sort_keys=None*, *sort_dirs=None*)

Get all groups belonging to a project.

`group_has_group_snapshot_filter`()

Return a filter that checks if a Group has Group Snapshots.

`group_has_volumes_filter`(*attached_or_with_snapshots=False*)

Return a filter to check if a Group has volumes.

When attached_or_with_snapshots parameter is given a True value only attached volumes or those with snapshots will be considered.

`group_include_in_cluster`(*context*, *cluster*, *partial_rename=True*, ***filters*)

Include all generic groups matching the filters into a cluster.

When partial_rename is set we will not set the cluster_name with cluster parameter value directly, well replace provided cluster_name or host filter value with cluster instead.

This is useful when we want to replace just the cluster name but leave the backend and pool information as it is. If we are using cluster_name to filter, well use that same DB field to replace the cluster value and leave the rest as it is. Likewise if we use the host to filter.

Returns the number of generic groups that have been changed.

`group_snapshot_create`(*context*, *values*)

Create a group snapshot from the values dictionary.

`group_snapshot_creating_from_src`()

Get a filter to check if a grp snapshot is being created from a grp.

`group_snapshot_destroy`(*context*, *group_snapshot_id*)

Destroy the group snapshot or raise if it does not exist.

`group_snapshot_get`(*context*, *group_snapshot_id*)

Get a group snapshot or raise if it does not exist.

`group_snapshot_get_all`(*context*, *filters=None*, *marker=None*, *limit=None*, *offset=None*, *sort_keys=None*, *sort_dirs=None*)

Get all group snapshots.

group_snapshot_get_all_by_group(*context*, *group_id*, *filters=None*, *marker=None*, *limit=None*, *offset=None*, *sort_keys=None*, *sort_dirs=None*)

Get all group snapshots belonging to a group.

group_snapshot_get_all_by_project(*context*, *project_id*, *filters=None*, *marker=None*, *limit=None*, *offset=None*, *sort_keys=None*, *sort_dirs=None*)

Get all group snapshots belonging to a project.

group_snapshot_update(*context*, *group_snapshot_id*, *values*)

Set the given properties on a group snapshot and update it.

Raises NotFound if group snapshot does not exist.

group_type_access_add(*context*, *type_id*, *project_id*)

Add group type access for project.

group_type_access_get_all(*context*, *type_id*)

Get all group type access of a group type.

group_type_access_remove(*context*, *type_id*, *project_id*)

Remove group type access for project.

group_type_create(*context*, *values*, *projects=None*)

Create a new group type.

group_type_destroy(*context*, *type_id*)

Delete a group type.

group_type_get(*context*, *id*, *inactive=False*, *expected_fields=None*)

Get group type by id.

Parameters

- `context` context to query under
- `id` Group type id to get.
- `inactive` Consider inactive group types when searching
- `expected_fields` Return those additional fields. Supported fields are: projects.

Returns

group type

group_type_get_all(*context*, *inactive=False*, *filters=None*, *marker=None*, *limit=None*, *sort_keys=None*, *sort_dirs=None*, *offset=None*, *list_result=False*)

Get all group types.

Parameters

- `context` context to query under
- `inactive` Include inactive group types to the result set
- `filters` Filters for the query in the form of key/value.
- `marker` the last item of the previous page, used to determine the next page of results to return

- `limit` maximum number of items to return
- `sort_keys` list of attributes by which results should be sorted, paired with corresponding item in sort_dirs
- `sort_dirs` list of directions in which results should be sorted, paired with corresponding item in sort_keys
- `list_result`

  For compatibility, if list_result = True, return a list
  instead of dict.

  is_public
  Filter group types based on visibility:

  - **True**: List public group types only
  - **False**: List private group types only
  - **None**: List both public and private group types

Returns
list/dict of matching group types

`group_type_get_by_name`(*context*, *name*)

Get group type by name.

`group_type_specs_delete`(*context*, *group_type_id*, *key*)

Delete the given group specs item.

`group_type_specs_get`(*context*, *group_type_id*)

Get all group specs for a group type.

`group_type_specs_update_or_create`(*context*, *group_type_id*, *group_specs*)

Create or update group type specs.

This adds or modifies the key/value pairs specified in the group specs dict argument.

`group_type_update`(*context*, *group_type_id*, *values*)

`group_types_get_by_name_or_id`(*context*, *group_type_list*)

Get group types by name or id.

`group_update`(*context*, *group_id*, *values*)

Set the given properties on a group and update it.

Raises NotFound if group does not exist.

`group_volume_type_mapping_create`(*context*, *group_id*, *volume_type_id*)

Create a group volume_type mapping entry.

`image_volume_cache_create`(*context*, *host*, *cluster_name*, *image_id*, *image_updated_at*, *volume_id*, *size*)

Create a new image volume cache entry.

`image_volume_cache_delete`(*context*, *volume_id*)

Delete an image volume cache entry specified by volume id.

image_volume_cache_get_all(*context*, ***filters*)

Query for all image volume cache entry for a host.

image_volume_cache_get_and_update_last_used(*context*, *image_id*, ***filters*)

Query for an image volume cache entry.

image_volume_cache_get_by_volume_id(*context*, *volume_id*)

Query to see if a volume id is an image-volume contained in the cache

image_volume_cache_include_in_cluster(*context*, *cluster*, *partial_rename=True*, ***filters*)

Include in cluster image volume cache entries matching the filters.

When partial_rename is set we will not set the cluster_name with cluster parameter value directly, well replace provided cluster_name or host filter value with cluster instead.

This is useful when we want to replace just the cluster name but leave the backend and pool information as it is. If we are using cluster_name to filter, well use that same DB field to replace the cluster value and leave the rest as it is. Likewise if we use the host to filter.

Returns the number of volumes that have been changed.

is_backend_frozen(*context*, *host*, *cluster_name*)

Check if a storage backend is frozen based on host and cluster_name.

is_orm_value(*obj*)

Check if object is an ORM field.

message_create(*context*, *values*)

Creates a new message with the specified values.

message_destroy(*context*, *message_id*)

Deletes message with the specified ID.

message_get(*context*, *message_id*)

Return a message with the specified ID.

message_get_all(*context*, *filters=None*, *marker=None*, *limit=None*, *offset=None*, *sort_keys=None*, *sort_dirs=None*)

project_default_volume_type_get(*context*, *project_id=None*)

Get default volume type for a project

project_default_volume_type_set(*context*, *volume_type_id*, *project_id*)

Set default volume type for a project

project_default_volume_type_unset(*context*, *project_id*)

Unset default volume type for a project (hard delete)

purge_deleted_rows(*context*, *age_in_days*)

Purge deleted rows older than given age from cinder tables

Raises InvalidParameterValue if age_in_days is incorrect. :returns: number of deleted rows

qos_specs_associate(*context*, *qos_specs_id*, *type_id*)

Associate qos_specs from volume type.

`qos_specs_associations_get`(*context*, *qos_specs_id*)
: Get all associated volume types for a given qos_specs.

`qos_specs_create`(*context*, *values*)
: Create a qos_specs.

`qos_specs_delete`(*context*, *qos_specs_id*)
: Delete the qos_specs.

`qos_specs_disassociate`(*context*, *qos_specs_id*, *type_id*)
: Disassociate qos_specs from volume type.

`qos_specs_disassociate_all`(*context*, *qos_specs_id*)
: Disassociate qos_specs from all entities.

`qos_specs_get`(*context*, *qos_specs_id*, *inactive=False*)
: Get all specification for a given qos_specs.

`qos_specs_get_all`(*context*, *filters=None*, *marker=None*, *limit=None*, *offset=None*, *sort_keys=None*, *sort_dirs=None*)
: Get all qos_specs.

`qos_specs_get_by_name`(*context*, *name*, *inactive=False*)
: Get all specification for a given qos_specs.

`qos_specs_item_delete`(*context*, *qos_specs_id*, *key*)
: Delete specified key in the qos_specs.

`qos_specs_update`(*context*, *qos_specs_id*, *values*)
: Update qos specs.

  This adds or modifies the key/value pairs specified in the specs dict argument for a given qos_specs.

`quota_class_create`(*context*, *class_name*, *resource*, *limit*)
: Create a quota class for the given name and resource.

`quota_class_destroy`(*context*, *class_name*, *resource*)
: Destroy the quota class or raise if it does not exist.

`quota_class_destroy_all_by_name`(*context*, *class_name*)
: Destroy all quotas associated with a given quota class.

`quota_class_get`(*context*, *class_name*, *resource*)
: Retrieve a quota class or raise if it does not exist.

`quota_class_get_all_by_name`(*context*, *class_name*)
: Retrieve all quotas associated with a given quota class.

`quota_class_get_defaults`(*context*)
: Retrieve all default quotas.

`quota_class_update`(*context*, *class_name*, *resource*, *limit*)
: Update a quota class or raise if it does not exist.

`quota_class_update_resource`(*context*, *old_res*, *new_res*)
: Update resource name in quota_class.

`quota_create`(*context*, *project_id*, *resource*, *limit*)
Create a quota for the given project and resource.

`quota_destroy`(*context*, *project_id*, *resource*)
Destroy the quota or raise if it does not exist.

`quota_destroy_by_project`(*context*, *project_id*)
Destroy all quotas associated with a given project.

`quota_get`(*context*, *project_id*, *resource*)
Retrieve a quota or raise if it does not exist.

`quota_get_all_by_project`(*context*, *project_id*)
Retrieve all quotas associated with a given project.

`quota_reserve`(*context*, *resources*, *quotas*, *deltas*, *expire*, *until_refresh*, *max_age*, *project_id=None*)
Check quotas and create appropriate reservations.

`quota_update`(*context*, *project_id*, *resource*, *limit*)
Update a quota or raise if it does not exist.

`quota_update_resource`(*context*, *old_res*, *new_res*)
Update resource of quotas.

`quota_usage_get`(*context*, *project_id*, *resource*)
Retrieve a quota usage or raise if it does not exist.

`quota_usage_get_all_by_project`(*context*, *project_id*)
Retrieve all usage associated with a given resource.

`quota_usage_update_resource`(*context*, *old_res*, *new_res*)
Update resource field in quota_usages.

`remove_temporary_admin_metadata_data_migration`(*context*, *max_count*)

`reservation_commit`(*context*, *reservations*, *project_id=None*)
Commit quota reservations.

`reservation_expire`(*context*)
Roll back any expired reservations.

`reservation_rollback`(*context*, *reservations*, *project_id=None*)
Roll back quota reservations.

`reset_active_backend`(*context*, *enable_replication*, *active_backend_id*, *backend_host*)
Reset the active backend for a host.

`resource_exists`(*context*, *model*, *resource_id*)

`service_create`(*context*, *values*)
Create a service from the values dictionary.

`service_destroy`(*context*, *service_id*)
Destroy the service or raise if it does not exist.

service_get(*context*, *service_id=None*, *backend_match_level=None*, ***filters*)

Get a service that matches the criteria.

A possible filter is is_up=True and it will filter nodes that are down.

Parameters

- **service_id** Id of the service.
- **filters** Filters for the query in the form of key/value.
- **backend_match_level** pool, backend, or host for host and cluster filters (as defined in _filter_host method)

Raises

ServiceNotFound If service doesnt exist.

service_get_all(*context*, *backend_match_level=None*, ***filters*)

Get all services that match the criteria.

A possible filter is is_up=True and it will filter nodes that are down, as well as host_or_cluster, that lets you look for services using both of these properties.

Parameters

- **filters** Filters for the query in the form of key/value arguments.
- **backend_match_level** pool, backend, or host for host and cluster filters (as defined in _filter_host method)

service_get_by_uuid(*context*, *service_uuid*)

Get a service by its uuid.

Return Service ref or raise if it does not exist.

service_update(*context*, *service_id*, *values*, *retry=True*)

Set the given properties on an service and update it.

Raises NotFound if service does not exist.

snapshot_create(*context*, *values*)

Create a snapshot from the values dictionary.

snapshot_data_get_for_project(*context*, *project_id*, *volume_type_id=None*, *host=None*)

Get count and gigabytes used for snapshots for specified project.

snapshot_destroy(*context*, *snapshot_id*)

Destroy the snapshot or raise if it does not exist.

snapshot_get(*context*, *snapshot_id*)

Get a snapshot or raise if it does not exist.

snapshot_get_all(*context*, *filters=None*, *marker=None*, *limit=None*, *sort_keys=None*, *sort_dirs=None*, *offset=None*)

Get all snapshots.

snapshot_get_all_active_by_window(*context*, *begin*, *end=None*, *project_id=None*)

Get all the snapshots inside the window.

Specifying a project_id will filter for a certain project.

`snapshot_get_all_by_host`(*context*, *host*, *filters=None*)

Get all snapshots belonging to a host.

Parameters

- **host** Include include snapshots only for specified host.
- **filters** Filters for the query in the form of key/value.

`snapshot_get_all_by_project`(*context*, *project_id*, *filters=None*, *marker=None*, *limit=None*, *sort_keys=None*, *sort_dirs=None*, *offset=None*)

Get all snapshots belonging to a project.

`snapshot_get_all_for_cgsnapshot`(*context*, *cgsnapshot_id*)

Get all snapshots belonging to a cgsnapshot.

`snapshot_get_all_for_group_snapshot`(*context*, *group_snapshot_id*)

Get all snapshots belonging to a group snapshot.

`snapshot_get_all_for_volume`(*context*, *volume_id*)

Get all snapshots for a volume.

`snapshot_get_latest_for_volume`(*context*, *volume_id*)

Get latest snapshot for a volume

`snapshot_metadata_delete`(*context*, *snapshot_id*, *key*)

Delete the given metadata item.

`snapshot_metadata_get`(*context*, *snapshot_id*)

Get all metadata for a snapshot.

`snapshot_metadata_update`(*context*, *snapshot_id*, *metadata*, *delete*)

Update metadata if it exists, otherwise create it.

`snapshot_update`(*context*, *snapshot_id*, *values*)

Set the given properties on an snapshot and update it.

Raises NotFound if snapshot does not exist.

`transfer_accept`(*context*, *transfer_id*, *user_id*, *project_id*, *no_snapshots=False*)

Accept a volume transfer.

`transfer_create`(*context*, *values*)

Create an entry in the transfers table.

`transfer_destroy`(*context*, *transfer_id*)

Destroy a record in the volume transfer table.

`transfer_get`(*context*, *transfer_id*)

Get a volume transfer record or raise if it does not exist.

`transfer_get_all`(*context*, *marker=None*, *limit=None*, *sort_keys=None*, *sort_dirs=None*, *filters=None*, *offset=None*)

Get all volume transfer records.

transfer_get_all_by_project(*context*, *project_id*, *marker=None*, *limit=None*, *sort_keys=None*, *sort_dirs=None*, *filters=None*, *offset=None*)

Get all volume transfer records for specified project.

volume_admin_metadata_delete(*context*, *volume_id*, *key*)

Delete the given metadata item.

volume_admin_metadata_get(*context*, *volume_id*)

Get all administration metadata for a volume.

volume_admin_metadata_update(*context*, *volume_id*, *metadata*, *delete*, *add=True*, *update=True*)

Update metadata if it exists, otherwise create it.

volume_attach(*context*, *values*)

Attach a volume.

volume_attached(*context*, *attachment_id*, *instance_uuid*, *host_name*, *mountpoint*, *attach_mode='rw'*, *mark_attached=True*)

Ensure that a volume is set as attached.

volume_attachment_get(*context*, *attachment_id*)

volume_attachment_get_all(*context*, *filters=None*, *marker=None*, *limit=None*, *offset=None*, *sort_keys=None*, *sort_dirs=None*)

volume_attachment_get_all_by_host(*context*, *host*, *filters=None*)

volume_attachment_get_all_by_instance_uuid(*context*, *instance_uuid*, *filters=None*)

volume_attachment_get_all_by_project(*context*, *project_id*, *filters=None*, *marker=None*, *limit=None*, *offset=None*, *sort_keys=None*, *sort_dirs=None*)

volume_attachment_get_all_by_volume_id(*context*, *volume_id*)

volume_attachment_update(*context*, *attachment_id*, *values*)

volume_create(*context*, *values*)

Create a volume from the values dictionary.

volume_data_get_for_host(*context*, *host*, *count_only=False*)

Get (volume_count, gigabytes) for project.

volume_data_get_for_project(*context*, *project_id*, *host=None*)

Get (volume_count, gigabytes) for project.

volume_destroy(*context*, *volume_id*)

Destroy the volume or raise if it does not exist.

volume_detached(*context*, *volume_id*, *attachment_id*)

Ensure that a volume is set as detached.

volume_encryption_metadata_get(*context*, *volume_id*)

volume_get(*context*, *volume_id*)

Get a volume or raise if it does not exist.

volume_get_all(*context*, *marker=None*, *limit=None*, *sort_keys=None*, *sort_dirs=None*, *filters=None*, *offset=None*)

Get all volumes.

volume_get_all_active_by_window(*context*, *begin*, *end=None*, *project_id=None*)

Get all the volumes inside the window.

Specifying a project_id will filter for a certain project.

volume_get_all_by_generic_group(*context*, *group_id*, *filters=None*)

Get all volumes belonging to a generic volume group.

volume_get_all_by_group(*context*, *group_id*, *filters=None*)

Get all volumes belonging to a consistency group.

volume_get_all_by_host(*context*, *host*, *filters=None*)

Get all volumes belonging to a host.

volume_get_all_by_project(*context*, *project_id*, *marker*, *limit*, *sort_keys=None*, *sort_dirs=None*, *filters=None*, *offset=None*)

Get all volumes belonging to a project.

volume_glance_metadata_bulk_create(*context*, *volume_id*, *metadata*)

Add Glance metadata for specified volume (multiple pairs).

volume_glance_metadata_copy_from_volume_to_volume(*context*, *src_volume_id*, *volume_id*)

Update the Glance metadata for a volume.

Update the Glance metadata for a volume by copying all of the key:value pairs from the originating volume.

This is so that a volume created from the volume (clone) will retain the original metadata.

volume_glance_metadata_copy_to_snapshot(*context*, *snapshot_id*, *volume_id*)

Update the Glance metadata for a snapshot.

This will copy all of the key:value pairs from the originating volume, to ensure that a volume created from the snapshot will retain the original metadata.

volume_glance_metadata_copy_to_volume(*context*, *volume_id*, *snapshot_id*)

Update the Glance metadata from a volume (created from a snapshot).

This will copy all of the key:value pairs from the originating snapshot, to ensure that the Glance metadata from the original volume is retained.

volume_glance_metadata_create(*context*, *volume_id*, *key*, *value*)

Update the Glance metadata for the specified volume.

volume_glance_metadata_delete_by_snapshot(*context*, *snapshot_id*)

Delete the glance metadata for a snapshot.

volume_glance_metadata_delete_by_volume(*context*, *volume_id*)

Delete the glance metadata for a volume.

volume_glance_metadata_get(*context*, *volume_id*)

Return the glance metadata for a volume.

volume_glance_metadata_get_all(*context*)

Return the glance metadata for all volumes.

volume_glance_metadata_list_get(*context*, *volume_id_list*)

Return the glance metadata for a volume list.

volume_has_attachments_filter()

volume_has_other_project_snp_filter()

volume_has_snapshots_filter()

volume_has_snapshots_in_a_cgsnapshot_filter()

volume_has_undeletable_snapshots_filter()

volume_include_in_cluster(*context*, *cluster*, *partial_rename=True*, ***filters*)

Include all volumes matching the filters into a cluster.

When partial_rename is set we will not set the cluster_name with cluster parameter value directly, well replace provided cluster_name or host filter value with cluster instead.

This is useful when we want to replace just the cluster name but leave the backend and pool information as it is. If we are using cluster_name to filter, well use that same DB field to replace the cluster value and leave the rest as it is. Likewise if we use the host to filter.

Returns the number of volumes that have been changed.

volume_metadata_delete(*context*, *volume_id*, *key*, *meta_type=METADATA_TYPES.user*)

Delete the given metadata item.

volume_metadata_get(*context*, *volume_id*)

Get all metadata for a volume.

volume_metadata_update(*context*, *volume_id*, *metadata*, *delete*, *meta_type=METADATA_TYPES.user*)

Update metadata if it exists, otherwise create it.

volume_qos_allows_retype(*new_vol_type*)

volume_snapshot_glance_metadata_get(*context*, *snapshot_id*)

Return the Glance metadata for the specified snapshot.

volume_type_access_add(*context*, *type_id*, *project_id*)

Add volume type access for project.

volume_type_access_get_all(*context*, *type_id*)

Get all volume type access of a volume type.

volume_type_access_remove(*context*, *type_id*, *project_id*)

Remove volume type access for project.

volume_type_create(*context*, *values*, *projects=None*)

Create a new volume type.

volume_type_destroy(*context*, *type_id*)

Delete a volume type.

`volume_type_encryption_create`(*context*, *volume_type_id*, *values*)

`volume_type_encryption_delete`(*context*, *volume_type_id*)

`volume_type_encryption_get`(*context*, *volume_type_id*)

`volume_type_encryption_update`(*context*, *volume_type_id*, *values*)

`volume_type_encryption_volume_get`(*context*, *volume_type_id*)

`volume_type_extra_specs_delete`(*context*, *volume_type_id*, *key*)

Delete the given extra specs item.

`volume_type_extra_specs_get`(*context*, *volume_type_id*)

Get all extra specs for a volume type.

`volume_type_extra_specs_update_or_create`(*context*, *volume_type_id*, *extra_specs*)

Create or update volume type extra specs.

This adds or modifies the key/value pairs specified in the extra specs dict argument.

`volume_type_get`(*context*, *id*, *inactive=False*, *expected_fields=None*)

Get volume type by id.

Parameters

- **context** context to query under
- **id** Volume type id to get.
- **inactive** Consider inactive volume types when searching
- **expected_fields** Return those additional fields. Supported fields are: projects.

Returns

volume type

`volume_type_get_all`(*context*, *inactive=False*, *filters=None*, *marker=None*, *limit=None*, *sort_keys=None*, *sort_dirs=None*, *offset=None*, *list_result=False*)

Get all volume types.

Parameters

- **context** context to query under
- **inactive** Include inactive volume types to the result set
- **filters** Filters for the query in the form of key/value.
- **marker** the last item of the previous page, used to determine the next page of results to return
- **limit** maximum number of items to return
- **sort_keys** list of attributes by which results should be sorted, paired with corresponding item in sort_dirs
- **sort_dirs** list of directions in which results should be sorted, paired with corresponding item in sort_keys
- **list_result**

For compatibility, if list_result = True, return a list
instead of dict.

is_public
Filter volume types based on visibility:

- **True**: List public volume types only
- **False**: List private volume types only
- **None**: List both public and private volume types

Returns
list/dict of matching volume types

`volume_type_get_all_by_group`(*context*, *group_id*)
Get all volumes in a group.

`volume_type_get_by_name`(*context*, *name*)
Get volume type by name.

`volume_type_qos_associate`(*context*, *type_id*, *qos_specs_id*)
Associate a volume type with specific qos specs.

`volume_type_qos_associations_get`(*context*, *qos_specs_id*, *inactive=False*)
Get volume types that are associated with specific qos specs.

`volume_type_qos_disassociate`(*context*, *qos_specs_id*, *type_id*)
Disassociate a volume type from specific qos specs.

`volume_type_qos_disassociate_all`(*context*, *qos_specs_id*)
Disassociate all volume types from specific qos specs.

`volume_type_qos_specs_get`(*context*, *type_id*)
Get all qos specs for given volume type.

`volume_type_update`(*context*, *volume_type_id*, *values*)

`volume_types_get_by_name_or_id`(*context*, *volume_type_list*)
Get volume types by name or id.

`volume_update`(*context*, *volume_id*, *values*)
Set the given properties on a volume and update it.

Raises NotFound if volume does not exist.

`volume_update_all_by_service`(*context*)
Update all volumes associated with an old service.

`volume_update_status_based_on_attachment`(*context*, *volume_id*)
Update volume status according to attached instance id

`volumes_update`(*context*, *values_list*)
Set the given properties on a list of volumes and update them.

Raises NotFound if a volume does not exist.

worker_claim_for_cleanup(*context*, *claimer_id*, *orm_worker*)
Soft delete a worker, change the service_id and update the worker.

worker_create(*context*, ***values*)
Create a worker entry from optional arguments.

worker_destroy(*context*, ***filters*)
Delete a worker (no soft delete).

worker_get(*context*, ***filters*)
Get a worker or raise exception if it does not exist.

worker_get_all(*context*, *until=None*, *db_filters=None*, ***filters*)
Get all workers that match given criteria.

worker_update(*context*, *id*, *filters=None*, *orm_worker=None*, ***values*)
Update a worker with given values.

cinder.db.base module

Base class for classes that need modular database access.

class Base
Bases: `object`

DB driver is injected in the init method.

cinder.db.migration module

Database setup and migration commands.

db_sync(*version=None*, *engine=None*)
Migrate the database to *version* or the most recent version.

Were currently straddling two migration systems, sqlalchemy-migrate and alembic. This handles both by ensuring we switch from one to the other at the appropriate moment.

db_version()
Get database version.

Module contents

DB abstraction for Cinder

cinder.group package

Submodules

cinder.group.api module

Handles all requests relating to groups.

class API
Bases: `Base`

API for interacting with the volume manager for groups.

create(*context*, *name*, *description*, *group_type*, *volume_types*, *availability_zone=None*)

create_from_src(*context*, *name*, *description=None*, *group_snapshot_id=None*, *source_group_id=None*)

create_group_snapshot(*context*, *group*, *name*, *description*)

delete(*context*, *group*, *delete_volumes=False*)

delete_group_snapshot(*context*, *group_snapshot*, *force=False*)

disable_replication(*context*, *group*)

enable_replication(*context*, *group*)

failover_replication(*context*, *group*, *allow_attached_volume=False*, *secondary_backend_id=None*)

get(*context*, *group_id*)

get_all(*context*, *filters=None*, *marker=None*, *limit=None*, *offset=None*, *sort_keys=None*, *sort_dirs=None*)

get_all_group_snapshots(*context*, *filters=None*, *marker=None*, *limit=None*, *offset=None*, *sort_keys=None*, *sort_dirs=None*)

get_group_snapshot(*context*, *group_snapshot_id*)

list_replication_targets(*context*, *group*)

reset_group_snapshot_status(*context*, *gsnapshot*, *status*)

Reset status of group snapshot

reset_status(*context*, *group*, *status*)

Reset status of generic group

update(*context*, *group*, *name*, *description*, *add_volumes*, *remove_volumes*)

Update group.

update_group_snapshot(*context*, *group_snapshot*, *fields*)

update_quota(*context*, *group*, *num*, *project_id=None*)

Module contents

cinder.image package

Subpackages

cinder.image.accelerators package

Submodules

cinder.image.accelerators.gzip module

class AccelGZIP

Bases: *AccelBase*

compress_img(*src*, *dest*, *run_as_root*)

decompress_img(*src*, *dest*, *run_as_root*)

is_accel_exist()

cinder.image.accelerators.qat module

class AccelQAT

Bases: *AccelBase*

compress_img(*src*, *dest*, *run_as_root*)

decompress_img(*src*, *dest*, *run_as_root*)

is_accel_exist()

Module contents

Submodules

cinder.image.accelerator module

class AccelBase

Bases: object

abstract compress_img(*src*, *dest*, *run_as_root*)

abstract decompress_img(*src*, *dest*, *run_as_root*)

abstract is_accel_exist()

class ImageAccel(*src*, *dest*)

Bases: object

compress_img(*run_as_root*)

decompress_img(*run_as_root*)

is_engine_ready()

is_gzip_compressed(*image_file*)

cinder.image.cache module

class ImageVolumeCache(*db*, *volume_api*, *max_cache_size_gb: int = 0*, *max_cache_size_count: int = 0*, *clone_across_pools: bool = False*)

Bases: object

create_cache_entry(*context: RequestContext*, *volume_ref: Volume*, *image_id: str*, *image_meta: dict*) → dict

Create a new cache entry for an image.

This assumes that the volume described by volume_ref has already been created and is in an available state.

ensure_space(*context: RequestContext*, *volume: Volume*) → bool

Makes room for a volume cache entry.

Returns True if successful, false otherwise.

evict(*context: RequestContext*, *cache_entry: dict*) → None

get_by_image_volume(*context: RequestContext*, *volume_id: str*)

get_entry(*context: RequestContext*, *volume_ref: Volume*, *image_id: str*, *image_meta: dict*) → dict | None

cinder.image.format_inspector module

This is a python implementation of virtual disk format inspection routines gathered from various public specification documents, as well as qemu disk driver code. It attempts to store and parse the minimum amount of data required, and in a streaming-friendly manner to collect metadata about complex-format images.

class CaptureRegion(*offset*, *length*)

Bases: `object`

Represents a region of a file we want to capture.

A region of a file we want to capture requires a byte offset into the file and a length. This is expected to be used by a data processing loop, calling capture() with the most recently-read chunk. This class handles the task of grabbing the desired region of data across potentially multiple fractional and unaligned reads.

Parameters

- **offset** Byte offset into the file starting the region
- **length** The length of the region

capture(*chunk*, *current_position*)

Process a chunk of data.

This should be called for each chunk in the read loop, at least until complete returns True.

Parameters

- **chunk** A chunk of bytes in the file
- **current_position** The position of the file processed by the read loop so far. Note that this will be the position in the file *after* the chunk being presented.

property complete

Returns True when we have captured the desired data.

class FileInspector(*tracing=False*)

Bases: object

A stream-based disk image inspector.

This base class works on raw images and is subclassed for more complex types. It is to be presented with the file to be examined one chunk at a time, during read processing and will only store as much data as necessary to determine required attributes of the file.

property actual_size

Returns the total size of the file.

This is usually smaller than virtual_size. NOTE: this will only be accurate if the entire file is read and processed.

property complete

Returns True if we have all the information needed.

property context_info

Return info on amount of data held in memory for auditing.

This is a dict of region:sizeinbytes items that the inspector uses to examine the file.

eat_chunk(*chunk*)

Call this to present chunks of the file to the inspector.

property format_match

Returns True if the file appears to be the expected format.

classmethod from_file(*filename*)

Read as much of a file as necessary to complete inspection.

NOTE: Because we only read as much of the file as necessary, the actual_size property will not reflect the size of the file, but the amount of data we read before we satisfied the inspector.

Raises ImageFormatError if we cannot parse the file.

has_region(*name*)

Returns True if named region has been defined.

new_region(*name, region*)

Add a new CaptureRegion by name.

post_process()

Post-read hook to process what has been read so far.

This will be called after each chunk is read and potentially captured by the defined regions. If any regions are defined by this call, those regions will be presented with the current chunk in case it is within one of the new regions.

region(*name*)

Get a CaptureRegion by name.

safety_check()

Perform some checks to determine if this file is safe.

Returns True if safe, False otherwise. It may raise ImageFormatError if safety cannot be guaranteed because of parsing or other errors.

property virtual_size

Returns the virtual size of the disk image, or zero if unknown.

exception ImageFormatError

Bases: `Exception`

An unrecoverable image format error that aborts the process.

class InfoWrapper(*source*, *fmt*)

Bases: `object`

A file-like object that wraps another and updates a format inspector.

This passes chunks to the format inspector while reading. If the inspector fails, it logs the error and stops calling it, but continues proxying data from the source to its user.

close()

read(*size*)

class QEDInspector(*tracing=False*)

Bases: *FileInspector*

property format_match

Returns True if the file appears to be the expected format.

safety_check()

Perform some checks to determine if this file is safe.

Returns True if safe, False otherwise. It may raise ImageFormatError if safety cannot be guaranteed because of parsing or other errors.

class QcowInspector(**a*, ***k*)

Bases: *FileInspector*

QEMU QCOW2 Format

This should only require about 32 bytes of the beginning of the file to determine the virtual size, and 104 bytes to perform the safety check.

BF_OFFSET = 8

BF_OFFSET_LEN = 8

I_FEATURES = 72

I_FEATURES_DATAFILE_BIT = 3

I_FEATURES_LEN = 8

I_FEATURES_MAX_BIT = 4

property format_match

Returns True if the file appears to be the expected format.

property has_backing_file

property has_data_file

property has_header

property has_unknown_features

safety_check()

Perform some checks to determine if this file is safe.

Returns True if safe, False otherwise. It may raise ImageFormatError if safety cannot be guaranteed because of parsing or other errors.

safety_check_allow_backing_file()

property virtual_size

Returns the virtual size of the disk image, or zero if unknown.

class TraceDisabled

Bases: `object`

A logger-like thing that swallows tracing when we do not want it.

debug(**a*, ***k*)

error(**a*, ***k*)

info(**a*, ***k*)

warning(**a*, ***k*)

class VDIInspector(**a*, ***k*)

Bases: *FileInspector*

VirtualBox VDI format

This only needs to store the first 512 bytes of the image.

property format_match

Returns True if the file appears to be the expected format.

property virtual_size

Returns the virtual size of the disk image, or zero if unknown.

class VHDInspector(**a*, ***k*)

Bases: *FileInspector*

Connectix/MS VPC VHD Format

This should only require about 512 bytes of the beginning of the file to determine the virtual size.

property format_match

Returns True if the file appears to be the expected format.

property virtual_size

Returns the virtual size of the disk image, or zero if unknown.

class VHDXInspector(**a*, ***k*)

Bases: *FileInspector*

MS VHDX Format

This requires some complex parsing of the stream. The first 256KiB of the image is stored to get the header and region information, and then we capture the first metadata region to read those records, find the location of the virtual size data and parse it. This needs to store the metadata table entries up until the VDS record, which may consist of up to 2047 32-byte entries at max. Finally, it must store a chunk of data at the offset of the actual VDS uint64.

`METAREGION = '8B7CA206-4790-4B9A-B8FE-575F050F886E'`

`VHDX_METADATA_TABLE_MAX_SIZE = 65536`

`VIRTUAL_DISK_SIZE = '2FA54224-CD1B-4876-B211-5DBED83BF4B8'`

`property format_match`

Returns True if the file appears to be the expected format.

`post_process()`

Post-read hook to process what has been read so far.

This will be called after each chunk is read and potentially captured by the defined regions. If any regions are defined by this call, those regions will be presented with the current chunk in case it is within one of the new regions.

`property virtual_size`

Returns the virtual size of the disk image, or zero if unknown.

`class VMDKInspector(*a, **k)`

Bases: *FileInspector*

vmware VMDK format (monolithicSparse and streamOptimized variants only)

This needs to store the 512 byte header and the descriptor region which should be just after that. The descriptor region is some variable number of 512 byte sectors, but is just text defining the layout of the disk.

`DESC_MAX_SIZE = 1048575`

`DESC_OFFSET = 512`

`GD_AT_END = 18446744073709551615`

`property format_match`

Returns True if the file appears to be the expected format.

`post_process()`

Post-read hook to process what has been read so far.

This will be called after each chunk is read and potentially captured by the defined regions. If any regions are defined by this call, those regions will be presented with the current chunk in case it is within one of the new regions.

`safety_check()`

Perform some checks to determine if this file is safe.

Returns True if safe, False otherwise. It may raise ImageFormatError if safety cannot be guaranteed because of parsing or other errors.

`property virtual_size`

Returns the virtual size of the disk image, or zero if unknown.

chunked_reader(*fileobj*, *chunk_size=512*)

detect_file_format(*filename*)

Attempts to detect the format of a file.

This runs through a file one time, running all the known inspectors in parallel. It stops reading the file once one of them matches or all of them are sure they dont match.

Returns the FileInspector that matched, if any. None if raw.

get_inspector(*format_name*)

Returns a FormatInspector class based on the given name.

Parameters

format_name The name of the disk_format (raw, qcow2, etc).

Returns

A FormatInspector or None if unsupported.

cinder.image.glance module

Implementation of an image service that uses Glance as the backend

class GlanceClientWrapper(*context: RequestContext | None = None*, *netloc: str | None = None*, *use_ssl: bool = False*)

Bases: `object`

Glance client wrapper class that implements retries.

call(*context: RequestContext*, *method: str*, **args: Any*, ***kwargs: str*) → Any

Call a glance client method.

If we get a connection error, retry the request according to CONF.glance_num_retries.

class GlanceImageService(*client: Any | None = None*)

Bases: `object`

Provides storage and retrieval of disk image objects within Glance.

add_location(*context: RequestContext*, *image_id: str*, *url: str*, *metadata: dict*) → dict

Add a backend location url to an image.

Returns a dict containing image metadata on success.

create(*context: RequestContext*, *image_meta: dict[str, Any]*, *data=None*) → dict[str, Any]

Store the image data and return the new image object.

delete(*context: RequestContext*, *image_id: str*) → bool

Delete the given image.

Raises

- ***ImageNotFound*** if the image does not exist.
- ***NotAuthorized*** if the user is not an owner.

detail(*context: RequestContext*, ***kwargs: str*) → list[dict]

Calls out to Glance for a list of detailed image information.

download(*context: RequestContext*, *image_id: str*, *data=None*)

Calls out to Glance for data and writes data.

get_location(*context: RequestContext*, *image_id: str*) → tuple[str | None, Any]

Get backend storage location url.

Returns a tuple containing the direct url and locations representing the backend storage location, or (None, None) if these attributes are not shown by Glance.

get_stores(*context: RequestContext*)

Returns a list of dicts with stores information.

list_members(*context: RequestContext*, *image_id: str*) → list[dict]

Returns a list of dicts with image member data.

show(*context: RequestContext*, *image_id: str*) → dict[str, Any]

Returns a dict with image data for the given opaque image id.

update(*context: RequestContext*, *image_id: str*, *image_meta: dict*, *data=None*, *purge_props: bool = True*, *store_id: str | None = None*, *base_image_ref: str | None = None*) → dict

Modify the given image with the new data.

get_api_servers(*context: RequestContext*) → Iterable

Return Iterable over shuffled api servers.

Shuffle a list of glance_api_servers and return an iterator that will cycle through the list, looping around to the beginning if necessary. If CONF.glance_api_servers is None then they will be retrieved from the catalog.

get_default_image_service() → *GlanceImageService*

get_remote_image_service(*context: RequestContext*, *image_href*) → tuple[*GlanceImageService*, str]

Create an image_service and parse the id from the given image_href.

The image_href param can be an href of the form http://example.com:9292/v1/images/b8b2c6f7-7345-4e2f-afa2-eedaba9cbbe3, or just an id such as b8b2c6f7-7345-4e2f-afa2-eedaba9cbbe3. If the image_href is a standalone id, then the default image service is returned.

Parameters

image_href href that describes the location of an image

Returns

a tuple of the form (image_service, image_id)

cinder.image.image_utils module

Helper methods to deal with images.

This is essentially a copy from nova.virt.images.py Some slight modifications, but at some point we should look at maybe pushing this up to Oslo

class TemporaryImages(*image_service: GlanceImageService*)

Bases: `object`

Manage temporarily downloaded images to avoid downloading it twice.

In the with TemporaryImages.fetch(image_service, ctx, image_id) as tmp clause, tmp can be used as the downloaded image path. In addition, image_utils.fetch() will use the pre-fetched image by the TemporaryImages. This is useful to inspect image contents before conversion.

classmethod fetch(*image_service: GlanceImageService*, *context: RequestContext*, *image_id: str*, *suffix: str | None = ''*) → Generator[str, None, None]

static for_image_service(*image_service: GlanceImageService*) → *TemporaryImages*

get(*context: RequestContext*, *image_id: str*)

check_available_space(*dest: str*, *image_size: int*, *image_id: str*) → None

check_image_conversion_disable(*disk_format*, *volume_format*, *image_id*, *upload=False*)

check_image_format(*source: str*, *src_format: str | None = None*, *image_id: str | None = None*, *data: QemuImgInfo | None = None*, *run_as_root: bool = True*) → None

Do some image format checks.

Verifies that the src_format matches what qemu-img thinks the image format is, and does some vmdk subformat checks. See Bug #1996188.

- Does not check for a qcow2 backing file.
- Will make a call out to qemu_img if data is None.

Parameters

- **source** filename of the image to check
- **src_format** source image format recognized by qemu_img, or None
- **image_id** the image ID if this is a Glance image, or None
- **data** a imageutils.QemuImgInfo object from this image, or None
- **run_as_root** when data is None, call qemu-img info as root

Raises

- *ImageUnacceptable* when the image fails some format checks
- **ProcessExecutionError** if qemu-img info fails

check_qcow2_image(*image_id: str*, *data: QemuImgInfo*) → None

Check some rules about qcow2 images.

Does not check for a backing_file, because cinder has some legitimate use cases for qcow2 backing files.

Makes sure the image:

- does not have a data_file

Parameters

- **image_id** the image id
- **data** an imageutils.QemuImgInfo object

Raises
ImageUnacceptable when the image fails the check

check_qemu_img_version(*minimum_version: str*) → None

check_virtual_size(*virtual_size: float*, *volume_size: int*, *image_id: str*) → int

check_vmdk_image(*image_id: str*, *data: QemuImgInfo*) → None

Check some rules about VMDK images.

Make sure the VMDK subformat (the createType in vmware docs) is one that we allow as determined by the vmdk_allowed_types configuration option. The default set includes only types that do not reference files outside the VMDK file, which can otherwise be used in exploits to expose host information.

Parameters

- **image_id** the image id
- **data** an imageutils.QemuImgInfo object

Raises
ImageUnacceptable when the VMDK createType is not in the allowed list

chown_if_needed(*volume_path: str*) → Generator[None, None, None]

cleanup_temporary_file(*backend_name: str*) → None

coalesce_chain(*vhd_chain: list[str]*) → str

coalesce_vhd(*vhd_path: str*) → None

convert_image(*source: str*, *dest: str*, *out_format: str*, *out_subformat: str | None = None*, *src_format: str | None = None*, *run_as_root: bool = True*, *throttle=None*, *cipher_spec: dict | None = None*, *passphrase_file: str | None = None*, *compress: bool = False*, *src_passphrase_file: str | None = None*, *image_id: str | None = None*, *data: QemuImgInfo | None = None*, *disable_sparse: bool = False*) → None

Convert image to other format.

NOTE: If the qemu-img convert command fails and this function raises an exception, a non-empty dest file may be left in the filesystem. It is the responsibility of the caller to decide what to do with this file.

Parameters

- **source** source filename
- **dest** destination filename
- **out_format** output image format of qemu-img
- **out_subformat** output image subformat
- **src_format** source image format (use image_utils.fixup_disk_format() to translate from a Glance format to one recognizable by qemu_img)
- **run_as_root** run qemu-img as root
- **throttle** a cinder.throttling.Throttle object, or None
- **cipher_spec** encryption details

- **passphrase_file** filename containing luks passphrase
- **compress** compress w/ qemu-img when possible (best effort)
- **src_passphrase_file** filename containing source volumes luks passphrase
- **image_id** the image ID if this is a Glance image, or None
- **data** a imageutils.QemuImgInfo object from this image, or None

Raises

- ***ImageUnacceptable*** when the image fails some format checks
- **ProcessExecutionError** when something goes wrong during conversion

create_temporary_file(**args: str*, ***kwargs: str*) → str

decode_cipher(*cipher_spec: str*, *key_size: int*) → dict[str, str]

Decode a dm-crypt style cipher specification string

The assumed format being cipher-chainmode-ivmode, similar to that documented under linux/Documentation/admin-guide/device-mapper/dm-crypt.txt in the kernel source tree. Cinder does not support the [:keycount] or [:ivopts] options.

discover_vhd_chain(*directory: str*) → list[str]

extract_targz(*archive_name: str*, *target: str*) → None

fetch(*context: RequestContext*, *image_service: GlanceImageService*, *image_id: str*, *path: str*, *_user_id*, *_project_id*) → None

fetch_to_raw(*context: RequestContext*, *image_service: GlanceImageService*, *image_id: str*, *dest: str*, *blocksize: int*, *user_id: str | None = None*, *project_id: str | None = None*, *size: int | None = None*, *run_as_root: bool = True*, *disable_sparse: bool = False*) → None

fetch_to_vhd(*context: RequestContext*, *image_service: GlanceImageService*, *image_id: str*, *dest: str*, *blocksize: int*, *volume_subformat: str | None = None*, *user_id: str | None = None*, *project_id: str | None = None*, *run_as_root: bool = True*, *disable_sparse: bool = False*) → None

fetch_to_volume_format(*context: RequestContext*, *image_service: GlanceImageService*, *image_id: str*, *dest: str*, *volume_format: str*, *blocksize: int*, *volume_subformat: str | None = None*, *user_id: str | None = None*, *project_id: str | None = None*, *size: int | None = None*, *run_as_root: bool = True*, *disable_sparse: bool = False*) → None

fetch_verify_image(*context: RequestContext*, *image_service: GlanceImageService*, *image_id: str*, *dest: str*) → None

filter_out_reserved_namespaces_metadata(*metadata: dict[str, str] | None*) → dict[str, str]

fix_vhd_chain(*vhd_chain: list[str]*) → None

fixup_disk_format(*disk_format: str*) → str

Return the format to be provided to qemu-img convert.

from_qemu_img_disk_format(*disk_format: str*) → str

Return the conventional format derived from qemu-img format.

get_qemu_data(*image_id: str*, *has_meta: bool*, *disk_format_raw: bool*, *dest: str*, *run_as_root: bool*, *force_share: bool = False*) → QemuImgInfo

get_qemu_img_version() → list[int] | None

The qemu-img version will be cached until the process is restarted.

get_vhd_size(*vhd_path: str*) → int

is_xenserver_format(*image_meta: dict*) → bool

qemu_img_info(*path: str*, *run_as_root: bool = True*, *force_share: bool = False*, *allow_qcow2_backing_file: bool = False*) → QemuImgInfo

Return an object containing the parsed output from qemu-img info.

replace_xenserver_image_with_coalesced_vhd(*image_file: str*) → None

resize_image(*source: str*, *size: int*, *run_as_root: bool = False*, *file_format: str | None = None*) → None

Changes the virtual size of the image.

resize_vhd(*vhd_path: str*, *size: int*, *journal: str*) → None

set_vhd_parent(*vhd_path: str*, *parentpath: str*) → None

temporary_dir() → ContextManager[str]

temporary_file(**args: str*, ***kwargs*) → Generator[str, None, None]

upload_volume(*context: RequestContext*, *image_service: GlanceImageService*, *image_meta: dict*, *volume_path: str*, *volume_fd=None*, *volume_format: str = 'raw'*, *run_as_root: bool = True*, *compress: bool = True*, *store_id: str | None = None*, *base_image_ref: str | None = None*) → None

validate_stores_id(*context: RequestContext*, *image_service_store_id: str*) → None

verify_glance_image_signature(*context: RequestContext*, *image_service: GlanceImageService*, *image_id: str*, *path: str*) → bool

Module contents

cinder.interface package

Submodules

cinder.interface.backup_chunked_driver module

Backup driver with chunked backup operations.

class BackupChunkedDriver

Bases: `BackupDriver`

Backup driver that supports chunked backups.

`delete_object`(*container*, *object_name*)

Delete object from container.

Parameters

- `container` The container to modify.
- `object_name` The object name to delete.

`get_container_entries`(*container*, *prefix*)

Get container entry names.

Parameters

- `container` The container from which to get entries.
- `prefix` The prefix used to match entries.

`get_extra_metadata`(*backup*, *volume*)

Return extra metadata to use in prepare_backup.

This method allows for collection of extra metadata in prepare_backup() which will be passed to get_object_reader() and get_object_writer(). Subclass extensions can use this extra information to optimize data transfers.

returns
json serializable object

`get_object_reader`(*container*, *object_name*, *extra_metadata=None*)

Returns a reader object for the backed up chunk.

Parameters

- `container` The container to read from.
- `object_name` The object name to read.
- `extra_metadata` Extra metadata to be included.

`get_object_writer`(*container*, *object_name*, *extra_metadata=None*)

Returns a writer which stores the chunk data in backup repository.

Parameters

- `container` The container to write to.
- `object_name` The object name to write.
- `extra_metadata` Extra metadata to be included.

Returns
A context handler that can be used in a with context.

`put_container`(*container*)

Create the container if needed. No failure if it pre-exists.

Parameters
`container` The container to write into.

update_container_name(*backup*, *container*)

Allows sub-classes to override container name.

This method exists so that sub-classes can override the container name as it comes in to the driver in the backup object. Implementations should return None if no change to the container name is desired.

cinder.interface.backup_driver module

Core backup driver interface.

All backup drivers should support this interface as a bare minimum.

class BackupDriver

Bases: *CinderInterface*

Backup driver required interface.

backup(*backup*, *volume_file*, *backup_metadata=False*)

Start a backup of a specified volume.

If backup[parent_id] is given, then an incremental backup should be performed.

If the parent backup is of different size, a full backup should be performed to ensure all data is included.

Parameters

- **backup** The backup information.
- **volume_file** The volume or file to write the backup to.
- **backup_metadata** Whether to include volume metadata in the backup.

The variable structure of backup in the following format:

```
{
   'id': id,
   'availability_zone': availability_zone,
   'service': driver_name,
   'user_id': context.user_id,
   'project_id': context.project_id,
   'display_name': name,
   'display_description': description,
   'volume_id': volume_id,
   'status': fields.BackupStatus.CREATING,
   'container': container,
   'parent_id': parent_id,
   'size': size,
   'host': host,
   'snapshot_id': snapshot_id,
   'data_timestamp': data_timestamp,
}
```

service: backup driver parent_id: parent backup id size: equal to volume size data_timestamp: backup creation time `check_for_setup_error()`

Method for checking if backup backend is successfully installed.

Depends on storage backend limitations and driver implementation this method could check if all needed config options are configurated well or try to connect to the storage to verify driver can do it without any issues.

A dummy default is provided. This method can be omitted from driver.

Returns
None

Raises

- *InvalidConfigurationValue* raise this if you detect a problem during a configuration check
- *BackupDriverException* raise this or one of its more specific subclasses if you detect setup problems other than invalid configuration
- **Exception** refrain from raising generic exceptions, although we catch them for the benefit of legacy code

`delete_backup(backup)`

Delete a backup from the backup store.

Parameters
backup The backup to be deleted.

`export_record(backup)`

Export driver specific backup record information.

If backup backend needs additional driver specific information to import backup record back into the system it must override this method and return it as a dictionary so it can be serialized into a string.

Default backup driver implementation has no extra information.

Parameters
backup backup object to export

Returns
driver_info - dictionary with extra information

`get_metadata(volume_id)`

Get volume metadata.

Returns a json-encoded dict containing all metadata and the restore version i.e. the version used to decide what actually gets restored from this container when doing a backup restore.

Typically best to use py:class:*BackupMetadataAPI* for this.

Parameters
volume_id The ID of the volume.

Returns
json-encoded dict of metadata.

import_record(*backup*, *driver_info*)

Import driver specific backup record information.

If backup backend needs additional driver specific information to import backup record back into the system it must override this method since it will be called with the extra information that was provided by export_record when exporting the backup.

Default backup driver implementation does nothing since it didnt export any specific data in export_record.

Parameters

- **backup** backup object to export
- **driver_info** dictionary with driver specific backup record information

Returns
None

put_metadata(*volume_id*, *json_metadata*)

Set volume metadata.

Typically best to use py:class:*BackupMetadataAPI* for this.

Parameters

- **volume_id** The ID of the volume.
- **json_metadata** The json-encoded dict of metadata.

restore(*backup*, *volume_id*, *volume_file*)

Restore volume from a backup.

Parameters

- **backup** The backup information.
- **volume_id** The volume to be restored.
- **volume_file** The volume or file to read the data from.

cinder.interface.base module

class CinderInterface

Bases: `object`

Interface base class for Cinder.

Cinder interfaces should inherit from this class to support indirect inheritance evaluation.

This can be used to validate compliance to an interface without requiring that the class actually be inherited from the same base class.

cinder.interface.fczm_driver module

Core fibre channel zone manager driver interface.

All fczm drivers should support this interface as a bare minimum.

class FibreChannelZoneManagerDriver

Bases: *CinderInterface*

FCZM driver required interface.

add_connection(*fabric*, *initiator_target_map*, *host_name=None*, *storage_system=None*)

Add a new initiator<>target connection.

All implementing drivers should provide concrete implementation for this API.

Parameters

- **fabric** Fabric name from cinder.conf file
- **initiator_target_map** Mapping of initiator to list of targets

```
Example initiator_target_map:

{
    '10008c7cff523b01': ['20240002ac000a50', '20240002ac000a40']
}
```

Note that WWPN can be in lower or upper case and can be : separated strings.

delete_connection(*fabric*, *initiator_target_map*, *host_name=None*, *storage_system=None*)

Delete an initiator<>target connection.

Parameters

- **fabric** Fabric name from cinder.conf file
- **initiator_target_map** Mapping of initiator to list of targets

```
Example initiator_target_map:

{
    '10008c7cff523b01': ['20240002ac000a50', '20240002ac000a40']
}
```

Note that WWPN can be in lower or upper case and can be : separated strings.

get_san_context(*target_wwn_list*)

Get SAN context for end devices.

Parameters

target_wwn_list Mapping of initiator to list of targets

Example initiator_target_map: [20240002ac000a50, 20240002ac000a40] Note that WWPN can be in lower or upper case and can be : separated strings.

cinder.interface.util module

class DriverInfo(*cls*)

Bases: `object`

Information about driver implementations.

get_backup_drivers()

Get a list of all backup drivers.

get_fczm_drivers()

Get a list of all fczm drivers.

get_volume_drivers()

Get a list of all volume drivers.

cinder.interface.volume_consistencygroup_driver module

Consistency group volume driver interface.

class VolumeConsistencyGroupDriver

Bases: *CinderInterface*

Interface for drivers that support consistency groups.

create_cgsnapshot(*context*, *cgsnapshot*, *snapshots*)

Creates a cgsnapshot.

Parameters

- **context** the context of the caller.
- **cgsnapshot** the dictionary of the cgsnapshot to be created.
- **snapshots** a list of snapshot dictionaries in the cgsnapshot.

Returns

model_update, snapshots_model_update

param snapshots is retrieved directly from the db. It is a list of cinder.db.sqlalchemy.models.Snapshot to be precise. It cannot be assigned to snapshots_model_update. snapshots_model_update is a list of dictionaries. It has to be built by the driver. An entry will be in this format: {id: xxx, status: xxx, }. model_update will be in this format: {status: xxx, }.

The driver should populate snapshots_model_update and model_update and return them.

The manager will check snapshots_model_update and update db accordingly for each snapshot. If the driver successfully deleted some snapshots but failed to delete others, it should set statuses of the snapshots accordingly so that the manager can update db correctly.

If the status in any entry of snapshots_model_update is error, the status in model_update will be set to the same if it is not already error.

If the status in model_update is error, the manager will raise an exception and the status of cgsnapshot will be set to error in the db. If snapshots_model_update is not returned by the driver, the manager will set the status of every snapshot to error in the except block.

If the driver raises an exception during the operation, it will be caught by the try-except block in the manager and the statuses of cgsnapshot and all snapshots will be set to error.

For a successful operation, the driver can either build the model_update and snapshots_model_update and return them or return None, None. The statuses of cgsnapshot and all snapshots will be set to available at the end of the manager function.

`create_consistencygroup`(*context*, *group*)

Creates a consistencygroup.

Parameters

- `context` the context of the caller.
- `group` the dictionary of the consistency group to be created.

Returns
model_update

model_update will be in this format: {status: xxx, }.

If the status in model_update is error, the manager will throw an exception and it will be caught in the try-except block in the manager. If the driver throws an exception, the manager will also catch it in the try-except block. The group status in the db will be changed to error.

For a successful operation, the driver can either build the model_update and return it or return None. The group status will be set to available.

`create_consistencygroup_from_src`(*context*, *group*, *volumes*, *cgsnapshot=None*, *snapshots=None*, *source_cg=None*, *source_vols=None*)

Creates a consistencygroup from source.

Parameters

- `context` the context of the caller.
- `group` the dictionary of the consistency group to be created.
- `volumes` a list of volume dictionaries in the group.
- `cgsnapshot` the dictionary of the cgsnapshot as source.
- `snapshots` a list of snapshot dictionaries in the cgsnapshot.
- `source_cg` the dictionary of a consistency group as source.
- `source_vols` a list of volume dictionaries in the source_cg.

Returns
model_update, volumes_model_update

The source can be cgsnapshot or a source cg.

param volumes is retrieved directly from the db. It is a list of cinder.db.sqlalchemy.models.Volume to be precise. It cannot be assigned to volumes_model_update. volumes_model_update is a list of dictionaries. It has to be built by the driver. An entry will be in this format: {id: xxx, status: xxx, }. model_update will be in this format: {status: xxx, }.

To be consistent with other volume operations, the manager will assume the operation is successful if no exception is thrown by the driver. For a successful operation, the driver can either build the model_update and volumes_model_update and return them or return None, None.

`delete_cgsnapshot`(*context*, *cgsnapshot*, *snapshots*)

Deletes a cgsnapshot.

Parameters

- `context` the context of the caller.
- `cgsnapshot` the dictionary of the cgsnapshot to be deleted.
- `snapshots` a list of snapshot dictionaries in the cgsnapshot.

Returns
model_update, snapshots_model_update

param snapshots is retrieved directly from the db. It is a list of cinder.db.sqlalchemy.models.Snapshot to be precise. It cannot be assigned to snapshots_model_update. snapshots_model_update is a list of dictionaries. It has to be built by the driver. An entry will be in this format: {id: xxx, status: xxx, }. model_update will be in this format: {status: xxx, }.

The driver should populate snapshots_model_update and model_update and return them.

The manager will check snapshots_model_update and update db accordingly for each snapshot. If the driver successfully deleted some snapshots but failed to delete others, it should set statuses of the snapshots accordingly so that the manager can update db correctly.

If the status in any entry of snapshots_model_update is error_deleting or error, the status in model_update will be set to the same if it is not already error_deleting or error.

If the status in model_update is error_deleting or error, the manager will raise an exception and the status of cgsnapshot will be set to error in the db. If snapshots_model_update is not returned by the driver, the manager will set the status of every snapshot to error in the except block.

If the driver raises an exception during the operation, it will be caught by the try-except block in the manager and the statuses of cgsnapshot and all snapshots will be set to error.

For a successful operation, the driver can either build the model_update and snapshots_model_update and return them or return None, None. The statuses of cgsnapshot and all snapshots will be set to deleted after the manager deletes them from db.

`delete_consistencygroup`(*context*, *group*, *volumes*)

Deletes a consistency group.

Parameters

- `context` the context of the caller.
- `group` the dictionary of the consistency group to be deleted.
- `volumes` a list of volume dictionaries in the group.

Returns
model_update, volumes_model_update

param volumes is retrieved directly from the db. It is a list of cinder.db.sqlalchemy.models.Volume to be precise. It cannot be assigned to volumes_model_update. volumes_model_update is a list of dictionaries. It has to be built by the driver. An entry will be in this format: {id: xxx, status: xxx, }. model_update will be in this format: {status: xxx, }.

The driver should populate volumes_model_update and model_update and return them.

The manager will check volumes_model_update and update db accordingly for each volume. If the driver successfully deleted some volumes but failed to delete others, it should set statuses of the volumes accordingly so that the manager can update db correctly.

If the status in any entry of volumes_model_update is error_deleting or error, the status in model_update will be set to the same if it is not already error_deleting or error.

If the status in model_update is error_deleting or error, the manager will raise an exception and the status of the group will be set to error in the db. If volumes_model_update is not returned by the driver, the manager will set the status of every volume in the group to error in the except block.

If the driver raises an exception during the operation, it will be caught by the try-except block in the manager. The statuses of the group and all volumes in it will be set to error.

For a successful operation, the driver can either build the model_update and volumes_model_update and return them or return None, None. The statuses of the group and all volumes will be set to deleted after the manager deletes them from db.

update_consistencygroup(*context*, *group*, *add_volumes=None*, *remove_volumes=None*)

Updates a consistency group.

Parameters

- **context** the context of the caller.
- **group** the dictionary of the consistency group to be updated.
- **add_volumes** a list of volume dictionaries to be added.
- **remove_volumes** a list of volume dictionaries to be removed.

Returns

model_update, add_volumes_update, remove_volumes_update

model_update is a dictionary that the driver wants the manager to update upon a successful return. If None is returned, the manager will set the status to available.

add_volumes_update and remove_volumes_update are lists of dictionaries that the driver wants the manager to update upon a successful return. Note that each entry requires a {id: xxx} so that the correct volume entry can be updated. If None is returned, the volume will remain its original status. Also note that you cannot directly assign add_volumes to add_volumes_update as add_volumes is a list of cinder.db.sqlalchemy.models.Volume objects and cannot be used for db update directly. Same with remove_volumes.

If the driver throws an exception, the status of the group as well as those of the volumes to be added/removed will be set to error.

cinder.interface.volume_driver module

Core backend volume driver interface.

All backend drivers should support this interface as a bare minimum, but some methods (marked as optional in their description) can rely on the default implementation.

class VolumeDriverCore

Bases: *CinderInterface*

Core backend driver required interface.

after_volume_copy(*context*, *src_vol*, *dest_vol*, *remote=None*)

Driver-specific actions executed after copying a volume.

This method will be called after _copy_volume_data during volume migration.

Parameters

- `context` Context
- `src_volume` Source volume in the copy operation.
- `dest_volume` Destination volume in the copy operation.
- `remote` Whether the copy operation is local.

Returns
There is no return value for this method.

`before_volume_copy`(*context*, *src_vol*, *dest_vol*, *remote=None*)

Driver-specific actions executed before copying a volume.

This method will be called before _copy_volume_data during volume migration.

Parameters

- `context` Context
- `src_volume` Source volume in the copy operation.
- `dest_volume` Destination volume in the copy operation.
- `remote` Whether the copy operation is local.

Returns
There is no return value for this method.

`check_for_setup_error`()

Validate there are no issues with the driver configuration.

Called after do_setup(). Driver initialization can occur there or in this call, but must be complete by the time this returns.

If this method raises an exception, the driver will be left in an uninitialized state by the volume manager, which means that it will not be sent requests for volume operations.

This method typically checks things like whether the configured credentials can be used to log in the storage backend, and whether any external dependencies are present and working.

Raises

- *VolumeBackendAPIException* in case of setup error.
- *InvalidConfigurationValue* raise this if you detect a problem during a configuration check

`clone_image`(*context*, *volume*, *image_location*, *image_meta*, *image_service*)

Create a volume efficiently from an existing image.

Drivers that, always or under some circumstances, can efficiently create a volume from a Glance image can implement this method to be given a chance to try to do the volume creation as efficiently as possible.

If the driver cannot do it efficiently on a specific call it can return `(None, False)` to let Cinder try other mechanisms.

This method is optional and most drivers wont need to implement it and can leverage the default driver implementation that returns `(None, False)` to indicate that this optimization is not possible on this driver.

Examples where drivers can do this optimization:

- When images are stored on the same storage system and the driver can locate them and efficiently create a volume. For example the RBD driver can efficiently create a volume if the image is stored on the same Ceph cluster and the image format is `raw`. Another example is the GPFS driver.
- When volumes are locally accessible and accessing them that way is more efficient than going through the remote connection mechanism. For example in the GPFS driver if the cloning feature doesnt work it will copy the file without using os-brick to connect to the volume.

Parameters

- **context** Security/policy info for the request.
- **volume** The volume to create, as an OVO instance. Drivers should use attributes to access its values instead of using the dictionary compatibility interface it provides.
- **image_location** Tuple with (`direct_url`, `locations`) from the image metadata fields. `direct_url`, when present, is a string whose format depends on the image services external storage in use. Any, or both, tuple positions can be None, depending on the image service configuration. `locations`, when present, is a list of dictionaries where the value of the `url` key contains the direct urls (including the one from `direct_url`).
- **image_meta** Dictionary containing information about the image, including basic attributes and custom properties. Some transformations have been applied, such as converting timestamps (from `created_at`, `updated_at`, and `deleted_at`) to datetimes, and deserializing JSON values from `block_device_mapping` and `mappings` keys if present. Base properties, as per the images schema, will be stored on the base dictionary and the rest will be stored under the `properties` key. An important field to check in this method is the `disk_format` (e.g. raw, qcow2).
- **image_service** The image service to use (`GlanceImageService` instance). Can fetch image data directly using it.

Returns

Tuple of (model_update, boolean) where the boolean specifies whether the clone occurred.

copy_image_to_volume(*context*, *volume*, *image_service*, *image_id*, *disable_sparse=False*)

Fetch the image from image_service and write it to the volume.

Parameters

- **context** Security/policy info for the request.
- **volume** The volume to create.
- **image_service** The image service to use.
- **image_id** The image identifier.
- **disable_sparse** Enable or disable sparse copy. Default=False.

Returns
Model updates.

`copy_volume_to_image`(*context*, *volume*, *image_service*, *image_meta*)

Copy the volume to the specified image.

Parameters

- `context` Security/policy info for the request.
- `volume` The volume to copy.
- `image_service` The image service to use.
- `image_meta` Information about the image.

Returns
Model updates.

`create_snapshot`(*snapshot*)

Creates a snapshot.

Parameters
`snapshot` Information for the snapshot to be created.

`create_volume`(*volume*)

Create a new volume on the backend.

This method is responsible only for storage allocation on the backend. It should not export a LUN or actually make this storage available for use, this is done in a later call.

TODO(smcginnis): Add example data structure of volume object.

Parameters
`volume` Volume object containing specifics to create.

Returns
(Optional) dict of database updates for the new volume.

Raises
VolumeBackendAPIException if creation failed.

`create_volume_from_snapshot`(*volume*, *snapshot*)

Creates a volume from a snapshot.

If volume_type extra specs includes replication: <is> True the driver needs to create a volume replica (secondary), and setup replication between the newly created volume and the secondary volume.

An optional larger size for the new volume can be specified. Drivers should check this value and create or expand the new volume to match.

Parameters

- `volume` The volume to be created.
- `snapshot` The snapshot from which to create the volume.

Returns
A dict of database updates for the new volume.

delete_snapshot(*snapshot*)

Deletes a snapshot.

Parameters
snapshot The snapshot to delete.

delete_volume(*volume*)

Delete a volume from the backend.

If the driver can talk to the backend and detects that the volume is no longer present, this call should succeed and allow Cinder to complete the process of deleting the volume.

It is imperative that this operation ensures that the data from the deleted volume cannot leak into new volumes when they are created, as new volumes are likely to belong to a different tenant/project.

Parameters
volume The volume to delete.

Raises
VolumeIsBusy if the volume is still attached or has snapshots. VolumeBackendAPIException on error.

do_setup(*context*)

Any initialization the volume driver needs to do while starting.

Called once by the manager after the driver is loaded. Can be used to set up clients, check licenses, set up protocol specific helpers, etc.

If you choose to raise an exception here, the setup is considered failed already and the check_for_setup_error() will not be called.

Parameters
context The admin context of type context.RequestContext.

Raises

- *InvalidConfigurationValue* raise this if you detect a problem during a configuration check
- *VolumeDriverException* raise this or one of its more specific subclasses if you detect setup problems other than invalid configuration

extend_volume(*volume*, *new_size*)

Extend the size of a volume.

Parameters

- **volume** The volume to extend.
- **new_size** The new desired size of the volume.

Note that if the volume backend doesnt support extending an in-use volume, the driver should report online_extend_support=False.

get_volume_stats(*refresh=False*)

Collects volume backend stats.

The get_volume_stats method is used by the volume manager to collect information from the driver instance related to information about the driver, available and used space, and driver/backend capabilities.

stats are stored in self._stats field, which could be updated in _update_volume_stats method.

It returns a dict with the following required fields:

- **volume_backend_name**
  This is an identifier for the backend taken from cinder.conf. Useful when using multi-backend.
- **vendor_name**
  Vendor/author of the driver who serves as the contact for the drivers development and support.
- **driver_version**
  The driver version is logged at cinder-volume startup and is useful for tying volume service logs to a specific release of the code. There are currently no rules for how or when this is updated, but it tends to follow typical major.minor.revision ideas.
- **storage_protocol**
  The protocol used to connect to the storage, this should be a short string such as: iSCSI, FC, NFS, ceph, etc. Available protocols are present in cinder.common.constants and they must be used instead of string literals. Variant values only exist for older drivers that were already reporting those values. New drivers must use non variant versions. In some cases this may be the same value as the driver_volume_type returned by the initialize_connection method, but they are not the same thing, since this one is meant to be used by the scheduler, while the latter is the os-brick connector identifier used in the factory method.
- **total_capacity_gb**
  The total capacity in gigabytes (GiB) of the storage backend being used to store Cinder volumes. Use keyword unknown if the backend cannot report the value or infinite if there is no upper limit. But, it is recommended to report real values as the Cinder scheduler assigns lowest weight to any storage backend reporting unknown or infinite.
- **free_capacity_gb**
  The free capacity in gigabytes (GiB). Use keyword unknown if the backend cannot report the value or infinite if there is no upper limit. But, it is recommended to report real values as the Cinder scheduler assigns lowest weight to any storage backend reporting unknown or infinite.

And the following optional fields:

- **reserved_percentage (integer)**
  Percentage of backend capacity which is not used by the scheduler.
- **location_info (string)**
  Driver-specific information used by the driver and storage backend to correlate Cinder volumes and backend LUNs/files.
- **QoS_support (Boolean)**
  Whether the backend supports quality of service.
- **provisioned_capacity_gb**
  The total provisioned capacity on the storage backend, in gigabytes (GiB), including

space consumed by any user other than Cinder itself.

- **max_over_subscription_ratio**
  The maximum amount a backend can be over subscribed.
- **thin_provisioning_support (Boolean)**
  Whether the backend is capable of allocating thinly provisioned volumes.
- **thick_provisioning_support (Boolean)**
  Whether the backend is capable of allocating thick provisioned volumes. (Typically True.)
- **total_volumes (integer)**
  Total number of volumes on the storage backend. This can be used in custom driver filter functions.
- **filter_function (string)**
  A custom function used by the scheduler to determine whether a volume should be allocated to this backend or not. Example:

  capabilities.total_volumes < 10
- **goodness_function (string)**
  Similar to filter_function, but used to weigh multiple volume backends. Example:

  capabilities.capacity_utilization < 0.6 ? 100 : 25
- **multiattach (Boolean)**
  Whether the backend supports multiattach or not. Defaults to False.
- **sparse_copy_volume (Boolean)**
  Whether copies performed by the volume manager for operations such as migration should attempt to preserve sparseness.
- **online_extend_support (Boolean)**
  Whether the backend supports in-use volume extend or not. Defaults to True.
- **clone_across_pools (Boolean)**
  Whether the backend supports cloning a volume across different pools. Defaults to False.

The returned dict may also contain a list, pools, which has a similar dict for each pool being used with the backend.

Parameters
`refresh` Whether to discard any cached values and force a full refresh of stats.

Returns
dict of appropriate values (see above).

`init_capabilities()`

Fetch and merge capabilities of the driver.

Do not override this, implement _init_vendor_properties instead.

`initialize_connection`(*volume*, *connector*, *initiator_data=None*)

Allow connection to connector and return connection info.

Parameters

- `volume` The volume to be attached.

- **connector** Dictionary containing information about what is being connected to.
- **initiator_data** (Optional) A dictionary of driver_initiator_data objects with key-value pairs that have been saved for this initiator by a driver in previous initialize_connection calls.

Returns
A dictionary of connection information. This can optionally include a initiator_updates field.

The initiator_updates field must be a dictionary containing a set_values and/or remove_values field. The set_values field must be a dictionary of key-value pairs to be set/updated in the db. The remove_values field must be a list of keys, previously set with set_values, that will be deleted from the db.

May be called multiple times to get connection information after a volume has already been attached.

initialized()

Getter for drivers initialized status.

Do not implement this in a driver. Rely on the default implementation.

migrate_volume(*context*, *volume*, *host*)

Migrate the volume to the specified host.

Parameters

- **context** Context
- **volume** A dictionary describing the volume to migrate
- **host** A dictionary describing the host to migrate to, where host[host] is its name, and host[capabilities] is a dictionary of its reported capabilities.

Returns
Tuple of (model_update, boolean) where the boolean specifies whether the migration occurred.

retype(*context*, *volume*, *new_type*, *diff*, *host*)

Change the type of a volume.

This operation occurs on the same backend and the return value indicates whether it was successful. If migration is required to satisfy a retype, that will be handled by the volume manager.

Parameters

- **context** Context
- **volume** The volume to retype
- **new_type** The target type for the volume
- **diff** The differences between the two types
- **host** The host that contains this volume

Returns
Tuple of (boolean, model_update) where the boolean specifies whether the retype occurred.

`set_initialized()`

Mark driver as initialized.

Do not implement this in a driver. Rely on the default implementation.

`set_throttle()`

Hook for initialization of cinder.volume.throttle.

This has not been necessary to re-implement or override in any drivers thus far. The generic implementation does nothing unless explicitly enabled.

`supported()`

Getter for drivers supported status.

Do not implement this in a driver. Rely on the default implementation.

`terminate_connection`(*volume*, *connector*)

Remove access to a volume.

Note: If `connector` is `None`, then all connections to the volume should be terminated.

Parameters

- **volume** The volume to remove.
- **connector** The Dictionary containing information about the connection. This is optional when doing a force-detach and can be None.

`update_migrated_volume`(*context*, *volume*, *new_volume*, *original_volume_status*)

Return model update for migrated volume.

Each driver implementing this method needs to be responsible for the values of _name_id and provider_location. If None is returned or either key is not set, it means the volume table does not need to change the value(s) for the key(s). The return format is {_name_id: value, provider_location: value}.

Parameters

- **context** Context
- **volume** The original volume that was migrated to this backend
- **new_volume** The migration volume object that was created on this backend as part of the migration process
- **original_volume_status** The status of the original volume

Returns
model_update to update DB with any needed changes

`update_provider_info`(*volumes*, *snapshots*)

Get provider info updates from driver.

This retrieves a list of volumes and a list of snapshots that changed their providers thanks to the initialization of the host, so that Cinder can update this information in the volume database.

This is only implemented by drivers where such migration is possible.

Parameters

- **volumes** List of Cinder volumes to check for updates
- **snapshots** List of Cinder snapshots to check for updates

Returns
tuple (volume_updates, snapshot_updates)

where volume updates {id: uuid, provider_id: <provider-id>} and snapshot updates {id: uuid, provider_id: <provider-id>}

cinder.interface.volume_group_driver module

Generic volume group volume driver interface.

class VolumeGroupDriver

Bases: *CinderInterface*

Interface for drivers that support groups.

create_group(*context*, *group*)

Creates a group.

Parameters

- **context** the context of the caller.
- **group** the Group object to be created.

Returns
model_update

model_update will be in this format: {status: xxx, }.

If the status in model_update is error, the manager will throw an exception and it will be caught in the try-except block in the manager. If the driver throws an exception, the manager will also catch it in the try-except block. The group status in the db will be changed to error.

For a successful operation, the driver can either build the model_update and return it or return None. The group status will be set to available.

create_group_from_src(*context*, *group*, *volumes*, *group_snapshot=None*, *snapshots=None*, *source_group=None*, *source_vols=None*)

Creates a group from source.

Parameters

- **context** the context of the caller.
- **group** the Group object to be created.
- **volumes** a list of Volume objects in the group.
- **group_snapshot** the GroupSnapshot object as source.
- **snapshots** a list of Snapshot objects in the group_snapshot.
- **source_group** a Group object as source.
- **source_vols** a list of Volume objects in the source_group.

Returns
model_update, volumes_model_update

The source can be group_snapshot or a source group.

param volumes is a list of objects retrieved from the db. It cannot be assigned to volumes_model_update. volumes_model_update is a list of dictionaries. It has to be built by the driver. An entry will be in this format: {id: xxx, status: xxx, }. model_update will be in this format: {status: xxx, }.

To be consistent with other volume operations, the manager will assume the operation is successful if no exception is thrown by the driver. For a successful operation, the driver can either build the model_update and volumes_model_update and return them or return None, None.

`create_group_snapshot`(*context*, *group_snapshot*, *snapshots*)

Creates a group_snapshot.

Parameters

- **context** the context of the caller.
- **group_snapshot** the GroupSnapshot object to be created.
- **snapshots** a list of Snapshot objects in the group_snapshot.

Returns
model_update, snapshots_model_update

param snapshots is a list of Snapshot objects. It cannot be assigned to snapshots_model_update. snapshots_model_update is a list of dictionaries. It has to be built by the driver. An entry will be in this format: {id: xxx, status: xxx, }. model_update will be in this format: {status: xxx, }.

The driver should populate snapshots_model_update and model_update and return them.

The manager will check snapshots_model_update and update db accordingly for each snapshot. If the driver successfully deleted some snapshots but failed to delete others, it should set statuses of the snapshots accordingly so that the manager can update db correctly.

If the status in any entry of snapshots_model_update is error, the status in model_update will be set to the same if it is not already error.

If the status in model_update is error, the manager will raise an exception and the status of group_snapshot will be set to error in the db. If snapshots_model_update is not returned by the driver, the manager will set the status of every snapshot to error in the except block.

If the driver raises an exception during the operation, it will be caught by the try-except block in the manager and the statuses of group_snapshot and all snapshots will be set to error.

For a successful operation, the driver can either build the model_update and snapshots_model_update and return them or return None, None. The statuses of group_snapshot and all snapshots will be set to available at the end of the manager function.

`delete_group`(*context*, *group*, *volumes*)

Deletes a group.

Parameters

- **context** the context of the caller.

- **group** the Group object to be deleted.
- **volumes** a list of Volume objects in the group.

Returns
model_update, volumes_model_update

param volumes is a list of objects retrieved from the db. It cannot be assigned to volumes_model_update. volumes_model_update is a list of dictionaries. It has to be built by the driver. An entry will be in this format: {id: xxx, status: xxx, }. model_update will be in this format: {status: xxx, }.

The driver should populate volumes_model_update and model_update and return them.

The manager will check volumes_model_update and update db accordingly for each volume. If the driver successfully deleted some volumes but failed to delete others, it should set statuses of the volumes accordingly so that the manager can update db correctly.

If the status in any entry of volumes_model_update is error_deleting or error, the status in model_update will be set to the same if it is not already error_deleting or error.

If the status in model_update is error_deleting or error, the manager will raise an exception and the status of the group will be set to error in the db. If volumes_model_update is not returned by the driver, the manager will set the status of every volume in the group to error in the except block.

If the driver raises an exception during the operation, it will be caught by the try-except block in the manager. The statuses of the group and all volumes in it will be set to error.

For a successful operation, the driver can either build the model_update and volumes_model_update and return them or return None, None. The statuses of the group and all volumes will be set to deleted after the manager deletes them from db.

delete_group_snapshot(*context*, *group_snapshot*, *snapshots*)

Deletes a group_snapshot.

Parameters

- **context** the context of the caller.
- **group_snapshot** the GroupSnapshot object to be deleted.
- **snapshots** a list of Snapshot objects in the group_snapshot.

Returns
model_update, snapshots_model_update

param snapshots is a list of objects. It cannot be assigned to snapshots_model_update. snapshots_model_update is a list of of dictionaries. It has to be built by the driver. An entry will be in this format: {id: xxx, status: xxx, }. model_update will be in this format: {status: xxx, }.

The driver should populate snapshots_model_update and model_update and return them.

The manager will check snapshots_model_update and update db accordingly for each snapshot. If the driver successfully deleted some snapshots but failed to delete others, it should set statuses of the snapshots accordingly so that the manager can update db correctly.

If the status in any entry of snapshots_model_update is error_deleting or error, the status in model_update will be set to the same if it is not already error_deleting or error.

If the status in model_update is error_deleting or error, the manager will raise an exception and the status of group_snapshot will be set to error in the db. If snapshots_model_update is not returned by the driver, the manager will set the status of every snapshot to error in the except block.

If the driver raises an exception during the operation, it will be caught by the try-except block in the manager and the statuses of group_snapshot and all snapshots will be set to error.

For a successful operation, the driver can either build the model_update and snapshots_model_update and return them or return None, None. The statuses of group_snapshot and all snapshots will be set to deleted after the manager deletes them from db.

update_group(*context*, *group*, *add_volumes=None*, *remove_volumes=None*)

Updates a group.

Parameters

- **context** the context of the caller.
- **group** the Group object to be updated.
- **add_volumes** a list of Volume objects to be added.
- **remove_volumes** a list of Volume objects to be removed.

Returns

model_update, add_volumes_update, remove_volumes_update

model_update is a dictionary that the driver wants the manager to update upon a successful return. If None is returned, the manager will set the status to available.

add_volumes_update and remove_volumes_update are lists of dictionaries that the driver wants the manager to update upon a successful return. Note that each entry requires a {id: xxx} so that the correct volume entry can be updated. If None is returned, the volume will remain its original status. Also note that you cannot directly assign add_volumes to add_volumes_update as add_volumes is a list of volume objects and cannot be used for db update directly. Same with remove_volumes.

If the driver throws an exception, the status of the group as well as those of the volumes to be added/removed will be set to error.

cinder.interface.volume_manageable_driver module

Manage/unmanage existing volume driver interface.

class VolumeListManageableDriver

Bases: *VolumeManagementDriver*

Interface to support listing manageable snapshots and volumes.

get_manageable_snapshots(*cinder_snapshots*, *marker*, *limit*, *offset*, *sort_keys*, *sort_dirs*)

List snapshots on the backend available for management by Cinder.

Returns a list of dictionaries, each specifying a snapshot in the host, with the following keys:

- reference (dictionary): The reference for a snapshot, which can be passed to manage_existing_snapshot.
- size (int): The size of the snapshot according to the storage backend, rounded up to the nearest GB.

- safe_to_manage (boolean): Whether or not this snapshot is safe to manage according to the storage backend. For example, is the snapshot in use or invalid for any reason.
- reason_not_safe (string): If safe_to_manage is False, the reason why.
- cinder_id (string): If already managed, provide the Cinder ID.
- extra_info (string): Any extra information to return to the user
- source_reference (string): Similar to reference, but for the snapshots source volume.

Parameters

- **`cinder_snapshots`** A list of snapshots in this host that Cinder currently manages, used to determine if a snapshot is manageable or not.
- **`marker`** The last item of the previous page; we return the next results after this value (after sorting)
- **`limit`** Maximum number of items to return
- **`offset`** Number of items to skip after marker
- **`sort_keys`** List of keys to sort results by (valid keys are identifier and size)
- **`sort_dirs`** List of directions to sort by, corresponding to sort_keys (valid directions are asc and desc)

`get_manageable_volumes`(*cinder_volumes*, *marker*, *limit*, *offset*, *sort_keys*, *sort_dirs*)

List volumes on the backend available for management by Cinder.

Returns a list of dictionaries, each specifying a volume in the host, with the following keys:

- reference (dictionary): The reference for a volume, which can be passed to manage_existing.
- size (int): The size of the volume according to the storage backend, rounded up to the nearest GB.
- safe_to_manage (boolean): Whether or not this volume is safe to manage according to the storage backend. For example, is the volume in use or invalid for any reason.
- reason_not_safe (string): If safe_to_manage is False, the reason why.
- cinder_id (string): If already managed, provide the Cinder ID.
- extra_info (string): Any extra information to return to the user

Parameters

- **`cinder_volumes`** A list of volumes in this host that Cinder currently manages, used to determine if a volume is manageable or not.
- **`marker`** The last item of the previous page; we return the next results after this value (after sorting)
- **`limit`** Maximum number of items to return
- **`offset`** Number of items to skip after marker
- **`sort_keys`** List of keys to sort results by (valid keys are identifier and size)

- **sort_dirs** List of directions to sort by, corresponding to sort_keys (valid directions are asc and desc)

class VolumeManagementDriver

Bases: *CinderInterface*

Interface for drivers that support managing existing volumes.

manage_existing(*volume*, *existing_ref*)

Brings an existing backend storage object under Cinder management.

existing_ref is passed straight through from the API requests manage_existing_ref value, and it is up to the driver how this should be interpreted. It should be sufficient to identify a storage object that the driver should somehow associate with the newly-created cinder volume structure.

There are two ways to do this:

1. Rename the backend storage object so that it matches the, volume[name] which is how drivers traditionally map between a cinder volume and the associated backend storage object.
2. Place some metadata on the volume, or somewhere in the backend, that allows other driver requests (e.g. delete, clone, attach, detach) to locate the backend storage object when required.

If the existing_ref doesnt make sense, or doesnt refer to an existing backend storage object, raise a ManageExistingInvalidReference exception.

The volume may have a volume_type, and the driver can inspect that and compare against the properties of the referenced backend storage object. If they are incompatible, raise a ManageExistingVolumeTypeMismatch, specifying a reason for the failure.

Parameters

- **volume** Cinder volume to manage
- **existing_ref** Dictionary with keys source-id, source-name with driver-specific values to identify a backend storage object.

Raises

- ***ManageExistingInvalidReference*** If the existing_ref doesnt make sense, or doesnt refer to an existing backend storage object.
- ***ManageExistingVolumeTypeMismatch*** If there is a mismatch between the volume type and the properties of the existing backend storage object.

manage_existing_get_size(*volume*, *existing_ref*)

Return size of volume to be managed by manage_existing.

When calculating the size, round up to the next GB.

Parameters

- **volume** Cinder volume to manage
- **existing_ref** Dictionary with keys source-id, source-name with driver-specific values to identify a backend storage object.

Raises

ManageExistingInvalidReference If the existing_ref doesnt make sense, or doesnt refer to an existing backend storage object.

unmanage(*volume*)

Removes the specified volume from Cinder management.

Does not delete the underlying backend storage object.

For most drivers, this will not need to do anything. However, some drivers might use this call as an opportunity to clean up any Cinder-specific configuration that they have associated with the backend storage object.

Parameters

volume Cinder volume to unmanage

cinder.interface.volume_snapshot_revert module

Revert to snapshot capable volume driver interface.

class VolumeSnapshotRevertDriver

Bases: *CinderInterface*

Interface for drivers that support revert to snapshot.

revert_to_snapshot(*context*, *volume*, *snapshot*)

Revert volume to snapshot.

Note: the revert process should not change the volumes current size, that means if the driver shrank the volume during the process, it should extend the volume internally.

Parameters

- **context** the context of the caller.
- **volume** The volume to be reverted.
- **snapshot** The snapshot used for reverting.

cinder.interface.volume_snapshotmanagement_driver module

Manage/unmanage existing volume snapshots driver interface.

class VolumeSnapshotManagementDriver

Bases: *CinderInterface*

Interface for drivers that support managing existing snapshots.

manage_existing_snapshot(*snapshot*, *existing_ref*)

Brings an existing backend storage object under Cinder management.

existing_ref is passed straight through from the API requests manage_existing_ref value, and it is up to the driver how this should be interpreted. It should be sufficient to identify a storage object that the driver should somehow associate with the newly-created cinder snapshot structure.

There are two ways to do this:

1. Rename the backend storage object so that it matches the snapshot[name] which is how drivers traditionally map between a cinder snapshot and the associated backend storage object.
2. Place some metadata on the snapshot, or somewhere in the backend, that allows other driver requests (e.g. delete) to locate the backend storage object when required.

Parameters

- **snapshot** The snapshot to manage.
- **existing_ref** Dictionary with keys source-id, source-name with driver-specific values to identify a backend storage object.

Raises

ManageExistingInvalidReference If the existing_ref doesnt make sense, or doesnt refer to an existing backend storage object.

manage_existing_snapshot_get_size(*snapshot*, *existing_ref*)

Return size of snapshot to be managed by manage_existing.

When calculating the size, round up to the next GB.

Parameters

- **snapshot** The snapshot to manage.
- **existing_ref** Dictionary with keys source-id, source-name with driver-specific values to identify a backend storage object.

Raises

ManageExistingInvalidReference If the existing_ref doesnt make sense, or doesnt refer to an existing backend storage object.

unmanage_snapshot(*snapshot*)

Removes the specified snapshot from Cinder management.

Does not delete the underlying backend storage object.

For most drivers, this will not need to do anything. However, some drivers might use this call as an opportunity to clean up any Cinder-specific configuration that they have associated with the backend storage object.

Parameters

snapshot The snapshot to unmanage.

Module contents

backupdriver(*cls*)

Decorator for concrete backup driver implementations.

fczmdriver(*cls*)

Decorator for concrete fibre channel zone manager drivers.

volumedriver(*cls*)

Decorator for concrete volume driver implementations.

cinder.keymgr package

Submodules

cinder.keymgr.conf_key_mgr module

An implementation of a key manager that reads its key from the projects configuration options.

This key manager implementation provides limited security, assuming that the key remains secret. Using the volume encryption feature as an example, encryption provides protection against a lost or stolen disk, assuming that the configuration file that contains the key is not stored on the disk. Encryption also protects the confidentiality of data as it is transmitted via iSCSI from the compute host to the storage host (again assuming that an attacker who intercepts the data does not know the secret key).

Because this implementation uses a single, fixed key, it proffers no protection once that key is compromised. In particular, different volumes encrypted with a key provided by this key manager actually share the same encryption key so *any* volume can be decrypted once the fixed key is known.

class ConfKeyManager(*configuration*)

Bases: `KeyManager`

Key Manager that supports one key defined by the fixed_key conf option.

This key manager implementation supports all the methods specified by the key manager interface. This implementation creates a single key in response to all invocations of create_key. Side effects (e.g., raising exceptions) for each method are handled as specified by the key manager interface.

add_consumer(*context*, *managed_object_id*, *consumer_data*)

Add a consumer to a managed object.

Implementations should verify that the caller has permission to add a consumer to the managed object by checking the context object (context). A NotAuthorized exception should be raised if the caller lacks permission.

If the specified object does not exist, then a KeyError should be raised. Implementations should preclude users from discerning the UUIDs of objects that belong to other users by repeatedly calling this method. That is, objects that belong to other users should be considered non-existent and completely invisible.

create_key(*context*, ***kwargs*)

Creates a symmetric key.

This implementation returns a UUID for the key read from the configuration file. A NotAuthorized exception is raised if the specified context is None.

create_key_pair(*context*, ***kwargs*)

Creates an asymmetric key pair.

This method creates an asymmetric key pair and returns the pair of key UUIDs. If the specified context does not permit the creation of keys, then a NotAuthorized exception should be raised. The order of the UUIDs will be (private, public).

delete(*context*, *managed_object_id*)

Represents deleting the key.

Because the ConfKeyManager has only one key, which is read from the configuration file, the key is not actually deleted when this is called.

get(*context*, *managed_object_id*)

Retrieves the key identified by the specified id.

This implementation returns the key that is associated with the specified UUID. A NotAuthorized exception is raised if the specified context is None; a KeyError is raised if the UUID is invalid.

list(*context*, *object_type=None*, *metadata_only=False*)

Retrieves a list of managed objects that match the criteria.

Note: Required abstract method starting with Castellan 0.13.0

Parameters

- **context** Contains information of the user and the environment for the request.
- **object_type** The type of object to retrieve.
- **metadata_only** Whether secret data should be included.

Raises

NotAuthorized If no user context.

remove_consumer(*context*, *managed_object_id*, *consumer_data*)

Remove a consumer from a managed object.

Implementations should verify that the caller has permission to remove a consumer to the managed object by checking the context object (context). A NotAuthorized exception should be raised if the caller lacks permission.

If the specified object does not exist, then a KeyError should be raised. Implementations should preclude users from discerning the UUIDs of objects that belong to other users by repeatedly calling this method. That is, objects that belong to other users should be considered non-existent and completely invisible.

store(*context*, *managed_object*, ***kwargs*)

Stores (i.e., registers) a key with the key manager.

warning_logged = False

cinder.keymgr.migration module

class KeyMigrator(*conf*)

Bases: `object`

handle_key_migration(*volumes*, *backups*)

migrate_fixed_key(*volumes=None*, *backups=None*, *conf=<oslo_config.cfg.ConfigOpts object>*)

cinder.keymgr.transfer module

class KeyTransfer(*conf: ConfigOpts*)

Bases: `object`

property service_context

Returns the cinder services context.

transfer_key(*volume: Volume*, *src_context: RequestContext*, *dst_context: RequestContext*) → None

Transfer the key from the src_context to the dst_context.

transfer_accept(*context: ~cinder.context.RequestContext*, *volume: ~cinder.objects.volume.Volume*, *conf: ~oslo_config.cfg.ConfigOpts = <oslo_config.cfg.ConfigOpts object>*) → None

Transfer the key from the cinder service to the recipient.

transfer_create(*context: ~cinder.context.RequestContext*, *volume: ~cinder.objects.volume.Volume*, *conf: ~oslo_config.cfg.ConfigOpts = <oslo_config.cfg.ConfigOpts object>*) → None

Transfer the key from the owner to the cinder service.

transfer_delete(*context: ~cinder.context.RequestContext*, *volume: ~cinder.objects.volume.Volume*, *conf: ~oslo_config.cfg.ConfigOpts = <oslo_config.cfg.ConfigOpts object>*) → None

Transfer the key from the cinder service back to the owner.

Module contents

cinder.message package

Submodules

cinder.message.api module

Handles all requests related to user facing messages.

class API

Bases: *Base*

API for handling user messages.

Cinder Messages describe the outcome of a user action using predefined fields that are members of objects defined in the cinder.message.message_field package. They are intended to be exposed to end users. Their primary purpose is to provide end users with a means of discovering what went wrong when an asynchronous action in the Volume REST API (for which theyve already received a 2xx response) fails.

Messages contain an expires_at field based on the creation time plus the value of the message_ttl configuration option. They are periodically reaped by a task of the SchedulerManager class whose periodicity is given by the message_reap_interval configuration option.

cleanup_expired_messages(*context*)

create(*context*, *action*, *resource_type='VOLUME'*, *resource_uuid=None*, *exception=None*, *detail=None*, *level='ERROR'*)

Create a message record with the specified information.

Parameters

- **context** current context object
- **action** a message_field.Action field describing what was taking place when this message was created

- **resource_type** a message_field.Resource field describing the resource this message applies to. Default is message_field.Resource.VOLUME
- **resource_uuid** the resource ID if this message applies to an existing resource. Default is None
- **exception** if an exception has occurred, you can pass it in and it will be translated into an appropriate message detail ID (possibly message_field.Detail.UNKNOWN_ERROR). The message in the exception itself is ignored in order not to expose sensitive information to end users. Default is None
- **detail** a message_field.Detail field describing the event the message is about. Default is None, in which case message_field.Detail.UNKNOWN_ERROR will be used for the message unless an exception in the message_field.EXCEPTION_DETAIL_MAPPINGS is passed; in that case the message_field.Detail field thats mapped to the exception is used.
- **level** a string describing the severity of the message. Suggested values are INFO, ERROR, WARNING. Default is ERROR.

`create_from_request_context`(*context*, *exception=None*, *detail=None*, *level='ERROR'*)

Create a message record with the specified information.

Parameters

- **context** current context object which we must have populated with the message_action, message_resource_type and message_resource_id fields
- **exception** if an exception has occurred, you can pass it in and it will be translated into an appropriate message detail ID (possibly message_field.Detail.UNKNOWN_ERROR). The message in the exception itself is ignored in order not to expose sensitive information to end users. Default is None
- **detail** a message_field.Detail field describing the event the message is about. Default is None, in which case message_field.Detail.UNKNOWN_ERROR will be used for the message unless an exception in the message_field.EXCEPTION_DETAIL_MAPPINGS is passed; in that case the message_field.Detail field thats mapped to the exception is used.
- **level** a string describing the severity of the message. Suggested values are INFO, ERROR, WARNING. Default is ERROR.

`delete`(*context*, *id*)

Delete message with the specified id.

`get`(*context*, *id*)

Return message with the specified id.

`get_all`(*context*, *filters=None*, *marker=None*, *limit=None*, *offset=None*, *sort_keys=None*, *sort_dirs=None*)

Return all messages for the given context.

cinder.message.defined_messages module

Event ID and user visible message mapping.

Event IDs are used to look up the message to be displayed for an API Message object. All defined messages should be appropriate for any API user to see and not contain any sensitive information. A good rule-of-thumb is to be very general in error messages unless the issue is due to a bad user action, then be specific.

class EventIds

Bases: object

ATTACH_READONLY_VOLUME = 'VOLUME_000003'

IMAGE_FROM_VOLUME_OVER_QUOTA = 'VOLUME_000004'

UNABLE_TO_ALLOCATE = 'VOLUME_000002'

UNKNOWN_ERROR = 'VOLUME_000001'

UNMANAGE_ENCRYPTED_VOLUME_UNSUPPORTED = 'VOLUME_000005'

get_message_text(*event_id*)

cinder.message.message_field module

Message Resource, Action, Detail and user visible message.

Use Resource, Action and Details combination to indicate the Event in the format of:

EVENT: VOLUME_RESOURCE_ACTION_DETAIL

Also, use exception-to-detail mapping to decrease the workload of classifying event in cinders task code.

class Action

Bases: object

ALL = (('001', 'schedule allocate volume'), ('002', 'attach volume'), ('003', 'copy volume to image'), ('004', 'update attachment'), ('005', 'copy image to volume'), ('006', 'unmanage volume'), ('007', 'extend volume'), ('008', 'create volume from backend storage'), ('009', 'create snapshot'), ('010', 'delete snapshot'), ('011', 'update snapshot'), ('012', 'update snapshot metadata'), ('013', 'create backup'), ('014', 'delete backup'), ('015', 'restore backup'), ('016', 'reimage volume'))

ATTACH_VOLUME = ('002', 'attach volume')

BACKUP_CREATE = ('013', 'create backup')

BACKUP_DELETE = ('014', 'delete backup')

BACKUP_RESTORE = ('015', 'restore backup')

COPY_IMAGE_TO_VOLUME = ('005', 'copy image to volume')

COPY_VOLUME_TO_IMAGE = ('003', 'copy volume to image')

CREATE_VOLUME_FROM_BACKEND = ('008', 'create volume from backend storage')

EXTEND_VOLUME = ('007', 'extend volume')

REIMAGE_VOLUME = ('016', 'reimage volume')

SCHEDULE_ALLOCATE_VOLUME = ('001', 'schedule allocate volume')

SNAPSHOT_CREATE = ('009', 'create snapshot')

SNAPSHOT_DELETE = ('010', 'delete snapshot')

SNAPSHOT_METADATA_UPDATE = ('012', 'update snapshot metadata')

SNAPSHOT_UPDATE = ('011', 'update snapshot')

UNMANAGE_VOLUME = ('006', 'unmanage volume')

UPDATE_ATTACHMENT = ('004', 'update attachment')

class Detail

Bases: object

ALL = (('001', 'An unknown error occurred.'), ('002', 'Driver is not initialized at present.'), ('003', 'Could not find any available weighted backend.'), ('004', 'Failed to upload volume to image at backend.'), ('005', "Volume's attach mode is invalid."), ('006', 'Not enough quota resource for operation.'), ('007', 'Image used for creating volume exceeds available space.'), ('008', 'Unmanaging encrypted volumes is not supported.'), ('009', 'Compute service failed to extend volume.'), ('010', 'Volume Driver failed to extend volume.'), ('011', 'Image signature verification failed.'), ('012', 'Driver failed to create the volume.'), ('013', 'Snapshot failed to create.'), ('014', 'Volume snapshot update metadata failed.'), ('015', 'Snapshot is busy.'), ('016', 'Snapshot failed to delete.'), ('017', 'Backup status is invalid.'), ('018', 'Backup service is down.'), ('019', 'Failed to get backup device from the volume service.'), ('020', 'Backup driver failed to create backup.'), ('021', 'Failed to attach volume.'), ('022', 'Failed to detach volume.'), ('023', 'Cleanup of temporary volume/snapshot failed.'), ('024', 'Backup failed to schedule. Service not found for creating backup.'), ('025', 'Backup driver failed to delete backup.'), ('026', 'Backup driver failed to restore backup.'), ('027', 'Volume status is invalid.'), ('028', 'Compute service failed to reimage volume.'), ('029', 'The image disk format must be the same as the volume format for the volume type you are requesting.'), ('030', 'Incremental backup not possible, forcing full backup.'))

ATTACH_ERROR = ('021', 'Failed to attach volume.')

BACKUP_CREATE_CLEANUP_ERROR = ('023', 'Cleanup of temporary volume/snapshot failed.')

BACKUP_CREATE_DEVICE_ERROR = ('019', 'Failed to get backup device from the volume service.')

BACKUP_CREATE_DRIVER_ERROR = ('020', 'Backup driver failed to create backup.')

BACKUP_DELETE_DRIVER_ERROR = ('025', 'Backup driver failed to delete backup.')

BACKUP_INVALID_STATE = ('017', 'Backup status is invalid.')

BACKUP_RESTORE_ERROR = ('026', 'Backup driver failed to restore backup.')

BACKUP_SCHEDULE_ERROR = ('024', 'Backup failed to schedule. Service not found for creating backup.')

BACKUP_SERVICE_DOWN = ('018', 'Backup service is down.')

DETACH_ERROR = ('022', 'Failed to detach volume.')

DRIVER_FAILED_CREATE = ('012', 'Driver failed to create the volume.')

DRIVER_FAILED_EXTEND = ('010', 'Volume Driver failed to extend volume.')

DRIVER_NOT_INITIALIZED = ('002', 'Driver is not initialized at present.')

EXCEPTION_DETAIL_MAPPINGS = {('002', 'Driver is not initialized at present.'): ['DriverNotInitialized'], ('003', 'Could not find any available weighted backend.'): ['NoValidBackend'], ('005', "Volume's attach mode is invalid."): ['InvalidVolumeAttachMode'], ('006', 'Not enough quota resource for operation.'): ['ImageLimitExceeded', 'BackupLimitExceeded', 'SnapshotLimitExceeded'], ('007', 'Image used for creating volume exceeds available space.'): ['ImageTooBig'], ('015', 'Snapshot is busy.'): ['SnapshotIsBusy']}

FAILED_TO_UPLOAD_VOLUME = ('004', 'Failed to upload volume to image at backend.')

IMAGE_FORMAT_UNACCEPTABLE = ('029', 'The image disk format must be the same as the volume format for the volume type you are requesting.')

INCREMENTAL_BACKUP_FORCES_FULL_BACKUP = ('030', 'Incremental backup not possible, forcing full backup.')

NOTIFY_COMPUTE_SERVICE_FAILED = ('009', 'Compute service failed to extend volume.')

NOT_ENOUGH_SPACE_FOR_IMAGE = ('007', 'Image used for creating volume exceeds available space.')

NO_BACKEND_AVAILABLE = ('003', 'Could not find any available weighted backend.')

QUOTA_EXCEED = ('006', 'Not enough quota resource for operation.')

REIMAGE_VOLUME_FAILED = ('028', 'Compute service failed to reimage volume.')

`SIGNATURE_VERIFICATION_FAILED = ('011', 'Image signature verification failed.')`

`SNAPSHOT_CREATE_ERROR = ('013', 'Snapshot failed to create.')`

`SNAPSHOT_DELETE_ERROR = ('016', 'Snapshot failed to delete.')`

`SNAPSHOT_IS_BUSY = ('015', 'Snapshot is busy.')`

`SNAPSHOT_UPDATE_METADATA_FAILED = ('014', 'Volume snapshot update metadata failed.')`

`UNKNOWN_ERROR = ('001', 'An unknown error occurred.')`

`UNMANAGE_ENC_NOT_SUPPORTED = ('008', 'Unmanaging encrypted volumes is not supported.')`

`VOLUME_ATTACH_MODE_INVALID = ('005', "Volume's attach mode is invalid.")`

`VOLUME_INVALID_STATE = ('027', 'Volume status is invalid.')`

class Resource

Bases: `object`

`VOLUME = 'VOLUME'`

`VOLUME_BACKUP = 'VOLUME_BACKUP'`

`VOLUME_SNAPSHOT = 'VOLUME_SNAPSHOT'`

translate_action(*action_id*)

translate_detail(*detail_id*)

translate_detail_id(*exception*, *detail*)

Get a detail_id to use for a message.

If exception is in the EXCEPTION_DETAIL_MAPPINGS, returns the detail_id of the mapped Detail field. If exception is not in the mapping or is None, returns the detail_id of the passed-in Detail field. Otherwise, returns the detail_id of Detail.UNKNOWN_ERROR.

Parameters

- **exception** an Exception (or None)
- **detail** a message_field.Detail field (or None)

Returns

string

Returns

the detail_id of a message_field.Detail field

Module contents

cinder.objects package

Submodules

cinder.objects.backup module

class Backup(**args, **kwargs*)

Bases: *CinderPersistentObject*, *CinderObject*, *CinderObjectDictCompat*, *CinderComparableObject*

OPTIONAL_FIELDS = ('metadata', 'parent')

VERSION = '1.7'

property availability_zone

property container

create() → None

property created_at

property data_timestamp

static decode_record(*backup_url*) → dict

Deserialize backup metadata from string into a dictionary.

Raises

InvalidInput

property deleted

property deleted_at

destroy() → None

property display_description

property display_name

encode_record(***kwargs*) → str

Serialize backup object, with optional extra info, into a string.

property encryption_key_id

property fail_reason

```
fields = {'availability_zone': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'container': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'created_at': DateTime(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'data_timestamp': DateTime(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'deleted': Boolean(default=False,nullable=True), 'deleted_at':
DateTime(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'display_description': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'display_name': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'encryption_key_id': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'fail_reason': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'host':
String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'id':
UUID(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False),
'metadata': Dict(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'num_dependent_backups': Integer(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'object_count': Integer(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'parent': Object(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'parent_id': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'project_id': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False),
'restore_volume_id': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'service': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'service_metadata': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'size':
Integer(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'snapshot_id': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'status': BackupStatus(default=<class 'oslo_versionedobjects.fields.
UnspecifiedDefault'>,nullable=True,valid_values=('error',
'error_deleting', 'creating', 'available', 'deleting', 'deleted',
'restoring')), 'temp_snapshot_id': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'temp_volume_id': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'updated_at': DateTime(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'user_id': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False),
'volume_id': UUID(default=<class
```

property has_dependent_backups: bool

property host

property id

property is_incremental: bool

property metadata

model
alias of Backup

property name

property num_dependent_backups

obj_extra_fields = ['name', 'is_incremental', 'has_dependent_backups']

obj_load_attr(*attrname*)
Load an additional attribute from the real object.

This should load self.$attrname and cache any data that might be useful for future load operations.

obj_reset_changes(*fields=None*)
Reset the list of fields that have been changed.

Parameters

- **fields** List of fields to reset, or all if None.
- **recursive** Call obj_reset_changes(recursive=True) on any sub-objects within the list of fields being reset.

This is NOT revert to previous values.

Specifying fields on recursive resets will only be honored at the top level. Everything below the top will reset all.

obj_what_changed()
Returns a set of fields that have been modified.

property object_count

property parent

property parent_id

property project_id

property restore_volume_id

save() → None
Save the changed fields back to the store.

This is optional for subclasses, but is presented here in the base class for consistency among those that do.

property service

property service_metadata

property size

property snapshot_id

property status

property temp_snapshot_id

property temp_volume_id

property updated_at

property user_id

property volume_id

class BackupDeviceInfo(*context=None*, ***kwargs*)

Bases: *CinderObject*, *CinderObjectDictCompat*, *CinderComparableObject*

VERSION = '1.0'

property device_obj

fields = {'secure_enabled': Boolean(default=False,nullable=False), 'snapshot': Object(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'volume': Object(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True)}

classmethod from_primitive(*primitive*, *context*, *expected_attrs=None*)

property is_snapshot

obj_extra_fields = ['is_snapshot', 'device_obj']

property secure_enabled

property snapshot

to_primitive(*context*)

property volume

class BackupImport(**args*, ***kwargs*)

Bases: *Backup*

Special object for Backup Imports.

This class should not be used for anything but Backup creation when importing backups to the DB.

On creation it allows to specify the ID for the backup, since its the reference used in parent_id it is imperative that this is preserved.

Backup Import objects get promoted to standard Backups when the import is completed.

property availability_zone

property container

create()

property created_at

property data_timestamp

property deleted

property deleted_at

property display_description

property display_name

property encryption_key_id

property fail_reason

fields = {'availability_zone': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'container': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'created_at': DateTime(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'data_timestamp': DateTime(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'deleted': Boolean(default=False,nullable=True), 'deleted_at': DateTime(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'display_description': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'display_name': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'encryption_key_id': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'fail_reason': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'host': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'id': UUID(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'metadata': Dict(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'num_dependent_backups': Integer(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'object_count': Integer(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'parent': Object(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'parent_id': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'project_id': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'restore_volume_id': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'service': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'service_metadata': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'size': Integer(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'snapshot_id': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'status': BackupStatus(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True,valid_values=('error', 'error_deleting', 'creating', 'available', 'deleting', 'deleted', 'restoring')), 'temp_snapshot_id': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'temp_volume_id': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'updated_at': DateTime(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'user_id': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'volume_id': UUID(default=<class

property host

property id

property metadata

model

alias of Backup

property num_dependent_backups

property object_count

property parent

property parent_id

property project_id

property restore_volume_id

property service

property service_metadata

property size

property snapshot_id

property status

property temp_snapshot_id

property temp_volume_id

property updated_at

property user_id

property volume_id

class BackupList(**args*, ***kwargs*)

Bases: `ObjectListBase`, `CinderObject`

VERSION = '1.0'

fields = {'objects': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}

classmethod get_all(*context:* RequestContext, *filters=None*, *marker=None*, *limit=None*, *offset=None*, *sort_keys=None*, *sort_dirs=None*) → *BackupList*

classmethod get_all_active_by_window(*context*, *begin*, *end*)

classmethod get_all_by_host(*context:* RequestContext, *host: str*) → *BackupList*

classmethod get_all_by_project(*context*, *project_id*, *filters=None*, *marker=None*, *limit=None*, *offset=None*, *sort_keys=None*, *sort_dirs=None*)

classmethod get_all_by_volume(*context: RequestContext*, *volume_id: str*, *vol_project_id: str*, *filters=None*) → *BackupList*

property objects

cinder.objects.base module

Cinder common internal object model

class CinderComparableObject

Bases: `ComparableVersionedObject`

class CinderObject(*context=None*, ***kwargs*)

Bases: `VersionedObject`

OBJ_PROJECT_NAMESPACE = 'cinder'

cinder_obj_get_changes()

Returns a dict of changed fields with tz unaware datetimes.

Any timezone aware datetime field will be converted to UTC timezone and returned as timezone unaware datetime.

This will allow us to pass these fields directly to a db update method as they cant have timezone information.

obj_make_compatible(*primitive*, *target_version*)

Make an object representation compatible with a target version.

This is responsible for taking the primitive representation of an object and making it suitable for the given target_version. This may mean converting the format of object attributes, removing attributes that have been added since the target version, etc. In general:

- If a new version of an object adds a field, this routine should remove it for older versions.
- If a new version changed or restricted the format of a field, this should convert it back to something a client knowing only of the older version will tolerate.
- If an object that this object depends on is bumped, then this object should also take a version bump. Then, this routine should backlevel the dependent object (by calling its obj_make_compatible()) if the requested version of this object is older than the version where the new dependent object was added.

Parameters

- **primitive** The result of `obj_to_primitive()`
- **target_version** The version string requested by the recipient of the object

Raises

`oslo_versionedobjects.exception.UnsupportedObjectError` if conversion is not possible for some reason

class CinderObjectDictCompat

Bases: `VersionedObjectDictCompat`

Mix-in to provide dictionary key access compat.

If an object needs to support attribute access using dictionary items instead of object attributes, inherit from this class. This should only be used as a temporary measure until all callers are converted to use modern attribute access.

NOTE(berrange) This class will eventually be deleted.

get(*key*, *value=<class 'oslo_versionedobjects.base._NotSpecifiedSentinel'>*)

For backwards-compatibility with dict-based objects.

NOTE(danms): May be removed in the future.

class CinderObjectRegistry(**args*, ***kwargs*)

Bases: `VersionedObjectRegistry`

registration_hook(*cls*, *index*)

Hook called when registering a class.

This method takes care of adding the class to cinder.objects namespace.

Should registering class have a method called cinder_ovo_cls_init it will be called to support class initialization. This is convenient for all persistent classes that need to register their models.

class CinderObjectSerializer(*version_cap=None*)

Bases: `VersionedObjectSerializer`

OBJ_BASE_CLASS

alias of `CinderObject`

serialize_entity(*context*, *entity*)

Serialize something to primitive form.

Parameters

- **ctxt** Request context, in deserialized form
- **entity** Entity to be serialized

Returns

Serialized form of entity

class CinderObjectVersionsHistory

Bases: `dict`

Helper class that maintains objects version history.

Current state of object versions is aggregated in a single version number that explicitly identifies a set of object versions. That way a service is able to report what objects it supports using a single string and all the newer services will know exactly what that mean for a single object.

add(*ver*, *updates*)

get_current()

`get_current_versions()`

class CinderPersistentObject

Bases: `object`

Mixin class for Persistent objects.

This adds the fields that we use in common for all persistent objects.

class Case(*whens*, *value=None*, *else_=None*)

Bases: `object`

Class for conditional value selection for conditional_update.

class Not(*value*, *field=None*, *auto_none=True*)

Bases: `Condition`

Class for negated condition values for conditional_update.

By default NULL values will be treated like Python treats None instead of how SQL treats it.

So for example when values are (1, 2) it will evaluate to True when we have value 3 or NULL, instead of only with 3 like SQL does.

get_filter(*model*, *field=None*)

OPTIONAL_FIELDS = ()

as_read_deleted(*mode='yes'*)

Context manager to make OVO with modified read deleted context.

This temporarily modifies the context embedded in an object to have a different *read_deleted* parameter.

Parameter mode accepts most of the same parameters as our *model_query* DB method. We support yes, no, and only.

usage:

with obj.as_read_deleted():
obj.refresh()

if obj.status = deleted:

classmethod cinder_ovo_cls_init()

This method is called on OVO registration and sets the DB model.

conditional_update(*values*, *expected_values=None*, *filters=()*, *save_all=False*, *reflect_changes=True*, *order=None*)

Compare-and-swap update.

A conditional object update that, unlike normal update, will SAVE the contents of the update to the DB.

Update will only occur in the DB and the object if conditions are met.

If no expected_values are passed in we will default to make sure that all fields have not been changed in the DB. Since we cannot know the original value in the DB for dirty fields in the object those will be excluded.

We have 4 different condition types we can use in expected_values:

- Equality: {status: available}
- Inequality: {status: vol_obj.Not(deleting)}
- In range: {status: [available, error]
- Not in range: {status: vol_obj.Not([in-use, attaching])

Method accepts additional filters, which are basically anything that can be passed to a sqlalchemy querys filter method, for example:

```
[~sql.exists().where(models.Volume.id == models.Snapshot.volume_id)]
```

We can select values based on conditions using Case objects in the values argument. For example:

```
has_snapshot_filter = sql.exists().where(
    models.Snapshot.volume_id == models.Volume.id)
case_values = volume.Case([(has_snapshot_filter, 'has-snapshot')],
                          else_='no-snapshot')
volume.conditional_update({'status': case_values},
                          {'status': 'available'}))
```

And we can use DB fields using model class attribute for example to store previous status in the corresponding field even though we dont know which value is in the db from those we allowed:

```
volume.conditional_update({'status': 'deleting',
                           'previous_status': volume.model.status},
                          {'status': ('available', 'error')})
```

Parameters

- **`values`** Dictionary of key-values to update in the DB.
- **`expected_values`** Dictionary of conditions that must be met for the update to be executed.
- **`filters`** Iterable with additional filters
- **`save_all`** Object may have changes that are not in the DB, this will say whether we want those changes saved as well.
- **`reflect_changes`** If we want changes made in the database to be reflected in the versioned object. This may mean in some cases that we have to reload the object from the database.
- **`order`** Specific order of fields in which to update the values

Returns

Boolean indicating whether db rows were updated. It will be False if we couldnt update the DB and True if we could.

classmethod exists(*context*, *id_*)

fields = {'created_at': DateTime(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'deleted': Boolean(default=False,nullable=True), 'deleted_at': DateTime(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'updated_at': DateTime(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True)}

classmethod **get_by_id**(*context*, *id*, **args*, ***kwargs*)

obj_as_admin()

Context manager to make an object call as an admin.

This temporarily modifies the context embedded in an object to be elevated() and restores it after the call completes. Example usage:

with obj.obj_as_admin():
obj.save()

refresh()

update_single_status_where(*new_status*, *expected_status*, *filters=()*)

class **ClusteredObject**

Bases: object

assert_not_frozen()

property **is_clustered**

property **resource_backend**

property **service_topic_queue**

class **ObjectListBase**(**args*, ***kwargs*)

Bases: ObjectListBase

obj_make_compatible(*primitive*, *target_version*)

cinder.objects.cgsnapshot module

class **CGSnapshot**(*context=None*, ***kwargs*)

Bases: *CinderPersistentObject*, *CinderObject*, *CinderObjectDictCompat*, *ClusteredObject*

OPTIONAL_FIELDS = ('consistencygroup', 'snapshots')

VERSION = '1.1'

property **cluster_name**

property **consistencygroup**

property **consistencygroup_id**

create()

property created_at

property deleted

property deleted_at

property description

destroy()

fields = {'consistencygroup': Object(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'consistencygroup_id': UUID(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'created_at': DateTime(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'deleted': Boolean(default=False,nullable=True), 'deleted_at': DateTime(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'description': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'id': UUID(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'name': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'project_id': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'snapshots': Object(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'status': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'updated_at': DateTime(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'user_id': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}

from_group_snapshot(*group_snapshot*)

Convert a generic volume group object to a cg object.

property host

property id

model

alias of CGSnapshot

property name

obj_load_attr(*attrname*)

Load an additional attribute from the real object.

This should load self.$attrname and cache any data that might be useful for future load operations.

property project_id

save()

Save the changed fields back to the store.

This is optional for subclasses, but is presented here in the base class for consistency among those that do.

property snapshots

property status

property updated_at

property user_id

class CGSnapshotList(**args*, ***kwargs*)

Bases: *ObjectListBase*, *CinderObject*

VERSION = '1.0'

fields = {'objects': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}

classmethod get_all(*context*, *filters=None*)

classmethod get_all_by_group(*context*, *group_id*, *filters=None*)

classmethod get_all_by_project(*context*, *project_id*, *filters=None*)

property objects

cinder.objects.cleanable module

class CinderCleanableObject

Bases: *CinderPersistentObject*

Base class for cleanable OVO resources.

All cleanable objects must have a host property/attribute.

classmethod cinder_ovo_cls_init()

Called on OVO registration, sets set of cleanable resources.

cleanable_resource_types = {'Snapshot', 'Volume'}

create_worker(*pinned=True*)

Create a worker entry at the API.

static decorate(*func*, *caller*, *extras=()*, *kwsyntax=False*)

Decorates a function/generator/coroutine using a caller. If kwsyntax is True calling the decorated functions with keyword syntax will pass the named arguments inside the `kw` dictionary, even if such argument are positional, similarly to what functools.wraps does. By default kwsyntax is False and the the arguments are untouched.

classmethod get_pinned_version()

classmethod get_rpc_api()

is_cleanable(*pinned=False*)

Check if cleanable VO status is cleanable.

Parameters
pinned (*bool*) If we should check against pinned version or current version.

Returns
Whether this needs a workers DB entry or not

refresh()

set_worker()

static set_workers(**decorator_args*)

Decorator that adds worker DB rows for cleanable versioned objects.

By default will take care of all cleanable objects, but we can limit which objects we want by passing the name of the arguments we want to be added.

unset_worker()

worker = None

cinder.objects.cleanup_request module

class CleanupRequest(*context=None*, ***kwargs*)

Bases: `CinderObject`, `ClusteredObject`

Versioned Object to send cleanup requests.

VERSION = '1.0'

property binary

property cluster_name

property disabled

fields = {'binary': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'cluster_name': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'disabled': Boolean(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'host': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'is_up': Boolean(default=False,nullable=True), 'resource_id': UUID(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'resource_type': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'service_id': Integer(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'until': DateTime(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True)}

property host

property is_up

property resource_id

property resource_type

property service_id

property until

cinder.objects.cluster module

class Cluster(*context=None*, ***kwargs*)

Bases: *CinderPersistentObject*, *CinderObject*, *CinderComparableObject*

Cluster Versioned Object.

Method get_by_id supports as additional named arguments:

- get_services: If we want to load all services from this cluster.
- **services_summary: If we want to load num_nodes and num_down_nodes** fields.
- is_up: Boolean value to filter based on the clusters up status.
- read_deleted: Filtering based on delete status. Default value no.
- Any other cluster field will be used as a filter.

OPTIONAL_FIELDS = ('num_hosts', 'num_down_hosts', 'services')

VERSION = '1.1'

property active_backend_id

property binary

create()

property created_at

property deleted

property deleted_at

destroy()

property disabled

property disabled_reason

fields = {'active_backend_id': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'binary': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'created_at': DateTime(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'deleted': Boolean(default=False,nullable=True), 'deleted_at': DateTime(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'disabled': Boolean(default=False,nullable=True), 'disabled_reason': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'frozen': Boolean(default=False,nullable=False), 'id': Integer(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'last_heartbeat': DateTime(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'name': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'num_down_hosts': Integer(default=0,nullable=False), 'num_hosts': Integer(default=0,nullable=False), 'replication_status': ReplicationStatus(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True,valid_values=('error', 'enabled', 'disabled', 'not-capable', 'failover-error', 'failing-over', 'failed-over', 'enabling', 'disabling')), 'services': Object(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'updated_at': DateTime(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True)}

property frozen

property id

property is_up

property last_heartbeat

model

alias of Cluster

property name

property num_down_hosts

property num_hosts

obj_load_attr(*attrname*)

Lazy load services attribute.

property replication_status

reset_service_replication()

Reset service replication flags on promotion.

When an admin promotes a cluster, each service member requires an update to maintain database consistency.

save()

Save the changed fields back to the store.

This is optional for subclasses, but is presented here in the base class for consistency among those that do.

property services

property updated_at

class ClusterList(**args*, ***kwargs*)

Bases: *ObjectListBase*, *CinderObject*

VERSION = '1.0'

fields = {'objects': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}

classmethod get_all(*context*, *is_up=None*, *get_services=False*, *services_summary=False*, *read_deleted='no'*, ***filters*)

Get all clusters that match the criteria.

Parameters

- **is_up** Boolean value to filter based on the clusters up status.
- **get_services** If we want to load all services from this cluster.
- **services_summary** If we want to load num_nodes and num_down_nodes fields.
- **read_deleted** Filtering based on delete status. Default value is no.
- **filters** Field based filters in the form of key/value.

property objects

cinder.objects.consistencygroup module

class ConsistencyGroup(*context=None*, ***kwargs*)

Bases: *CinderPersistentObject*, *CinderObject*, *CinderObjectDictCompat*, *ClusteredObject*

OPTIONAL_FIELDS = ('cgsnapshots', 'volumes', 'cluster')

VERSION = '1.4'

property availability_zone

property cgsnapshot_id

property cgsnapshots

property cluster

property cluster_name

create(*cg_snap_id=None*, *cg_id=None*)

Create a consistency group.

If cg_snap_id or cg_id are specified then volume_type_id, availability_zone, and host will be taken from the source Consistency Group.

property created_at

property deleted

property deleted_at

property description

destroy()

```
fields = {'availability_zone': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'cgsnapshot_id': UUID(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'cgsnapshots': Object(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'cluster': Object(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'cluster_name': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'created_at': DateTime(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'deleted': Boolean(default=False,nullable=True), 'deleted_at':
DateTime(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'description': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'host':
String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'id':
UUID(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False),
'name': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'project_id': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False),
'source_cgid': UUID(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'status': ConsistencyGroupStatus(default=<class 'oslo_versionedobjects.
fields.UnspecifiedDefault'>,nullable=True,valid_values=('error',
'available', 'creating', 'deleting', 'deleted', 'updating',
'error_deleting')), 'updated_at': DateTime(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'user_id': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False),
'volume_type_id': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'volumes': Object(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True)}
```

`from_group`(*group*)

Convert a generic volume group object to a cg object.

`property host`

`property id`

`model`

alias of `ConsistencyGroup`

`property name`

`obj_load_attr`(*attrname*)

Load an additional attribute from the real object.

This should load self.$attrname and cache any data that might be useful for future load operations.

property project_id

save()

Save the changed fields back to the store.

This is optional for subclasses, but is presented here in the base class for consistency among those that do.

property source_cgid

property status

property updated_at

property user_id

property volume_type_id

property volumes

class ConsistencyGroupList(**args*, ***kwargs*)

Bases: `ObjectListBase`, `CinderObject`

VERSION = '1.1'

fields = {'objects': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}

classmethod get_all(*context*, *filters=None*, *marker=None*, *limit=None*, *offset=None*, *sort_keys=None*, *sort_dirs=None*)

classmethod get_all_by_project(*context*, *project_id*, *filters=None*, *marker=None*, *limit=None*, *offset=None*, *sort_keys=None*, *sort_dirs=None*)

static include_in_cluster(*context*, *cluster*, *partial_rename=True*, ***filters*)

Include all consistency groups matching the filters into a cluster.

When partial_rename is set we will not set the cluster_name with cluster parameter value directly, well replace provided cluster_name or host filter value with cluster instead.

This is useful when we want to replace just the cluster name but leave the backend and pool information as it is. If we are using cluster_name to filter, well use that same DB field to replace the cluster value and leave the rest as it is. Likewise if we use the host to filter.

Returns the number of consistency groups that have been changed.

property objects

cinder.objects.dynamic_log module

class LogLevel(*context=None*, ***kwargs*)

Bases: `CinderObject`

Versioned Object to send log change requests.

VERSION = '1.0'

fields = {'level': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'prefix': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True)}

property level

property prefix

class LogLevelList(*args, **kwargs)

Bases: *ObjectListBase*, *CinderObject*

VERSION = '1.0'

fields = {'objects': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}

property objects

cinder.objects.fields module

Custom fields for Cinder objects.

class BackupStatus

Bases: *BaseCinderEnum*

ALL = ('error', 'error_deleting', 'creating', 'available', 'deleting', 'deleted', 'restoring')

AVAILABLE = 'available'

CREATING = 'creating'

DELETED = 'deleted'

DELETING = 'deleting'

ERROR = 'error'

ERROR_DELETING = 'error_deleting'

RESTORING = 'restoring'

class BackupStatusField(**kwargs)

Bases: BaseEnumField

AUTO_TYPE = <cinder.objects.fields.BackupStatus object>

class BaseCinderEnum

Bases: Enum

class ConsistencyGroupStatus

Bases: *BaseCinderEnum*

ALL = ('error', 'available', 'creating', 'deleting', 'deleted', 'updating', 'error_deleting')

AVAILABLE = 'available'

CREATING = 'creating'

DELETED = 'deleted'

DELETING = 'deleting'

ERROR = 'error'

ERROR_DELETING = 'error_deleting'

UPDATING = 'updating'

class ConsistencyGroupStatusField(***kwargs*)

Bases: `BaseEnumField`

AUTO_TYPE = <cinder.objects.fields.ConsistencyGroupStatus object>

class DictOfNullableField(***kwargs*)

Bases: `AutoTypedField`

AUTO_TYPE = <oslo_versionedobjects.fields.Dict object>

class GroupSnapshotStatus

Bases: *BaseCinderEnum*

ALL = ('error', 'available', 'creating', 'deleting', 'deleted', 'updating', 'error_deleting')

AVAILABLE = 'available'

CREATING = 'creating'

DELETED = 'deleted'

DELETING = 'deleting'

ERROR = 'error'

ERROR_DELETING = 'error_deleting'

UPDATING = 'updating'

class GroupSnapshotStatusField(***kwargs*)

Bases: `BaseEnumField`

AUTO_TYPE = <cinder.objects.fields.GroupSnapshotStatus object>

class GroupStatus

Bases: *BaseCinderEnum*

ALL = ('error', 'available', 'creating', 'deleting', 'deleted', 'updating', 'in-use', 'error_deleting')

AVAILABLE = 'available'

CREATING = 'creating'

DELETED = 'deleted'

DELETING = 'deleting'

ERROR = 'error'

ERROR_DELETING = 'error_deleting'

IN_USE = 'in-use'

UPDATING = 'updating'

class GroupStatusField(***kwargs*)

Bases: BaseEnumField

AUTO_TYPE = <cinder.objects.fields.GroupStatus object>

class QoSConsumerField(***kwargs*)

Bases: BaseEnumField

AUTO_TYPE = <cinder.objects.fields.QoSConsumerValues object>

class QoSConsumerValues

Bases: *BaseCinderEnum*

ALL = ('back-end', 'front-end', 'both')

BACK_END = 'back-end'

BOTH = 'both'

FRONT_END = 'front-end'

class ReplicationStatus

Bases: *BaseCinderEnum*

ALL = ('error', 'enabled', 'disabled', 'not-capable', 'failover-error', 'failing-over', 'failed-over', 'enabling', 'disabling')

DISABLED = 'disabled'

DISABLING = 'disabling'

ENABLED = 'enabled'

ENABLING = 'enabling'

ERROR = 'error'

FAILED_OVER = 'failed-over'

FAILING_OVER = 'failing-over'

FAILOVER_ERROR = 'failover-error'

NOT_CAPABLE = 'not-capable'

class ReplicationStatusField(**kwargs)

Bases: BaseEnumField

AUTO_TYPE = <cinder.objects.fields.ReplicationStatus object>

class SnapshotStatus

Bases: *BaseCinderEnum*

ALL = ('error', 'available', 'creating', 'deleting', 'deleted', 'updating', 'error_deleting', 'unmanaging', 'backing-up', 'restoring')

AVAILABLE = 'available'

BACKING_UP = 'backing-up'

CREATING = 'creating'

DELETED = 'deleted'

DELETING = 'deleting'

ERROR = 'error'

ERROR_DELETING = 'error_deleting'

RESTORING = 'restoring'

UNMANAGING = 'unmanaging'

UPDATING = 'updating'

class SnapshotStatusField(**kwargs)

Bases: BaseEnumField

AUTO_TYPE = <cinder.objects.fields.SnapshotStatus object>

class VolumeAttachStatus

Bases: *BaseCinderEnum*

ALL = ('attached', 'attaching', 'detached', 'error_attaching', 'error_detaching', 'reserved', 'deleted')

ATTACHED = 'attached'

ATTACHING = 'attaching'

DELETED = 'deleted'

DETACHED = 'detached'

ERROR_ATTACHING = 'error_attaching'

ERROR_DETACHING = 'error_detaching'

RESERVED = 'reserved'

class VolumeAttachStatusField(***kwargs*)

Bases: BaseEnumField

AUTO_TYPE = <cinder.objects.fields.VolumeAttachStatus object>

class VolumeMigrationStatus

Bases: *BaseCinderEnum*

ALL = ('migrating', 'error', 'success', 'completing', 'none', 'starting')

COMPLETING = 'completing'

ERROR = 'error'

MIGRATING = 'migrating'

NONE = 'none'

STARTING = 'starting'

SUCCESS = 'success'

class VolumeMigrationStatusField(***kwargs*)

Bases: BaseEnumField

AUTO_TYPE = <cinder.objects.fields.VolumeMigrationStatus object>

class VolumeStatus

Bases: *BaseCinderEnum*

ALL = ('creating', 'available', 'deleting', 'error', 'error_deleting', 'error_managing', 'managing', 'attaching', 'in-use', 'detaching', 'maintenance', 'restoring-backup', 'error_restoring', 'reserved', 'awaiting-transfer', 'backing-up', 'error_backing-up', 'error_extending', 'downloading', 'uploading', 'retyping', 'extending')

ATTACHING = 'attaching'

AVAILABLE = 'available'

AWAITING_TRANSFER = 'awaiting-transfer'

BACKING_UP = 'backing-up'

CREATING = 'creating'

DELETING = 'deleting'

DETACHING = 'detaching'

DOWNLOADING = 'downloading'

ERROR = 'error'

ERROR_BACKING_UP = 'error_backing-up'

ERROR_DELETING = 'error_deleting'

ERROR_EXTENDING = 'error_extending'

ERROR_MANAGING = 'error_managing'

ERROR_RESTORING = 'error_restoring'

EXTENDING = 'extending'

IN_USE = 'in-use'

MAINTENANCE = 'maintenance'

MANAGING = 'managing'

RESERVED = 'reserved'

RESTORING_BACKUP = 'restoring-backup'

RETYPING = 'retyping'

UPLOADING = 'uploading'

class VolumeStatusField(**kwargs*)

Bases: BaseEnumField

AUTO_TYPE = <cinder.objects.fields.VolumeStatus object>

cinder.objects.group module

class Group(*context=None*, ***kwargs*)

Bases: *CinderPersistentObject*, *CinderObject*, *CinderObjectDictCompat*, *ClusteredObject*

OPTIONAL_FIELDS = ('volumes', 'volume_types', 'group_snapshots')

VERSION = '1.2'

property availability_zone

property cluster_name

create(*group_snapshot_id=None*, *source_group_id=None*)

property created_at

property deleted

property deleted_at

property description

destroy()

fields = {'availability_zone': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'cluster_name': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'created_at': DateTime(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'deleted': Boolean(default=False,nullable=True), 'deleted_at': DateTime(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'description': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'group_snapshot_id': UUID(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'group_snapshots': Object(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'group_type_id': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'host': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'id': UUID(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'name': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'project_id': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'replication_status': ReplicationStatus(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True, valid_values=('error', 'enabled', 'disabled', 'not-capable', 'failover-error', 'failing-over', 'failed-over', 'enabling', 'disabling')), 'source_group_id': UUID(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'status': GroupStatus(default=<class 'oslo_versionedobjects.fields. UnspecifiedDefault'>,nullable=True,valid_values=('error', 'available', 'creating', 'deleting', 'deleted', 'updating', 'in-use', 'error_deleting')), 'updated_at': DateTime(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'user_id': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'volume_type_ids': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'volume_types': Object(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'volumes': Object(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True)}

property group_snapshot_id

property group_snapshots

property group_type_id

property host

property id

property is_replicated

model

alias of Group

property name

obj_load_attr(*attrname*)

Load an additional attribute from the real object.

This should load self.$attrname and cache any data that might be useful for future load operations.

property project_id

property replication_status

save()

Save the changed fields back to the store.

This is optional for subclasses, but is presented here in the base class for consistency among those that do.

property source_group_id

property status

property updated_at

property user_id

property volume_type_ids

property volume_types

property volumes

class GroupList(**args*, ***kwargs*)

Bases: ObjectListBase, CinderObject

VERSION = '1.0'

fields = {'objects': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}

classmethod get_all(*context*, *filters=None*, *marker=None*, *limit=None*, *offset=None*, *sort_keys=None*, *sort_dirs=None*)

classmethod get_all_by_project(*context*, *project_id*, *filters=None*, *marker=None*, *limit=None*, *offset=None*, *sort_keys=None*, *sort_dirs=None*)

classmethod get_all_replicated(*context*, *filters=None*, *marker=None*, *limit=None*, *offset=None*, *sort_keys=None*, *sort_dirs=None*)

static include_in_cluster(*context*, *cluster*, *partial_rename=True*, ***filters*)

Include all generic groups matching the filters into a cluster.

When partial_rename is set we will not set the cluster_name with cluster parameter value directly, well replace provided cluster_name or host filter value with cluster instead.

This is useful when we want to replace just the cluster name but leave the backend and pool information as it is. If we are using cluster_name to filter, well use that same DB field to replace the cluster value and leave the rest as it is. Likewise if we use the host to filter.

Returns the number of generic groups that have been changed.

property objects

cinder.objects.group_snapshot module

class GroupSnapshot(*context=None*, ***kwargs*)

Bases: *CinderPersistentObject*, *CinderObject*, *CinderObjectDictCompat*, *ClusteredObject*

OPTIONAL_FIELDS = ('group', 'snapshots')

VERSION = '1.0'

property cluster_name

create()

property created_at

property deleted

property deleted_at

property description

destroy()

fields = {'created_at': DateTime(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'deleted': Boolean(default=False,nullable=True), 'deleted_at': DateTime(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'description': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'group': Object(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'group_id': UUID(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'group_type_id': UUID(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'id': UUID(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'name': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'project_id': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'snapshots': Object(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'status': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'updated_at': DateTime(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'user_id': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True)}

property group

property group_id

property group_type_id

property host

property id

model

alias of GroupSnapshot

property name

obj_load_attr(*attrname*)

Load an additional attribute from the real object.

This should load self.$attrname and cache any data that might be useful for future load operations.

property project_id

save()

Save the changed fields back to the store.

This is optional for subclasses, but is presented here in the base class for consistency among those that do.

property snapshots

property status

property updated_at

property user_id

class GroupSnapshotList(**args*, ***kwargs*)

Bases: `ObjectListBase`, `CinderObject`

VERSION = '1.0'

fields = {'objects': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}

classmethod get_all(*context*, *filters=None*, *marker=None*, *limit=None*, *offset=None*, *sort_keys=None*, *sort_dirs=None*)

classmethod get_all_by_group(*context*, *group_id*, *filters=None*, *marker=None*, *limit=None*, *offset=None*, *sort_keys=None*, *sort_dirs=None*)

classmethod get_all_by_project(*context*, *project_id*, *filters=None*, *marker=None*, *limit=None*, *offset=None*, *sort_keys=None*, *sort_dirs=None*)

property objects

cinder.objects.group_type module

class GroupType(*context=None*, ***kwargs*)

Bases: `CinderPersistentObject`, `CinderObject`, `CinderObjectDictCompat`, `CinderComparableObject`

OPTIONAL_FIELDS = ('group_specs', 'projects')

VERSION = '1.0'

create()

property created_at

property deleted

property deleted_at

property description

destroy()

fields = {'created_at': DateTime(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'deleted': Boolean(default=False,nullable=True), 'deleted_at': DateTime(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'description': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'group_specs': Dict(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'id': UUID(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'is_public': Boolean(default=True,nullable=True), 'name': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'projects': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'updated_at': DateTime(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True)}

property group_specs

property id

property is_public

model

alias of GroupType

property name

property projects

save()

Save the changed fields back to the store.

This is optional for subclasses, but is presented here in the base class for consistency among those that do.

property updated_at

class GroupTypeList(*args, **kwargs)

Bases: *ObjectListBase*, *CinderObject*

VERSION = '1.0'

fields = {'objects': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}

classmethod get_all(*context*, *inactive=0*, *filters=None*, *marker=None*, *limit=None*, *sort_keys=None*, *sort_dirs=None*, *offset=None*)

property objects

cinder.objects.manageableresources module

class ManageableObject

Bases: object

fields = {'cinder_id': UUID(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'extra_info': Dict(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'reason_not_safe': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'reference': Dict(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'safe_to_manage': Boolean(default=False,nullable=True), 'size': Integer(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True)}

classmethod from_primitives(*context*, *dict_resource*)

class ManageableSnapshot(*context=None*, ***kwargs*)

Bases: *CinderObject*, *CinderObjectDictCompat*, *ManageableObject*

VERSION = '1.0'

property cinder_id

property extra_info

fields = {'cinder_id': UUID(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'extra_info': Dict(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'reason_not_safe': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'reference': Dict(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'safe_to_manage': Boolean(default=False,nullable=True), 'size': Integer(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'source_reference': Dict(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}

property reason_not_safe

property reference

property safe_to_manage

property size

property source_reference

class ManageableSnapshotList(**args*, ***kwargs*)

Bases: *ObjectListBase*, *CinderObject*

VERSION = '1.0'

fields = {'objects': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}

classmethod from_primitives(*context*, *data*)

property objects

class ManageableVolume(*context=None*, ***kwargs*)

Bases: *CinderObject*, *CinderObjectDictCompat*, *CinderComparableObject*, *ManageableObject*

VERSION = '1.0'

property cinder_id

property extra_info

fields = {'cinder_id': UUID(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'extra_info': Dict(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'reason_not_safe': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'reference': Dict(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'safe_to_manage': Boolean(default=False,nullable=True), 'size': Integer(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True)}

property reason_not_safe

property reference

property safe_to_manage

property size

class ManageableVolumeList(**args*, ***kwargs*)

Bases: *ObjectListBase*, *CinderObject*

VERSION = '1.0'

fields = {'objects': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}

classmethod from_primitives(*context*, *data*)

property objects

cinder.objects.qos_specs module

class QualityOfServiceSpecs(**args*, ***kwargs*)

Bases: *CinderPersistentObject*, *CinderObject*, *CinderObjectDictCompat*, *CinderComparableObject*

`OPTIONAL_FIELDS = ('volume_types',)`

`VERSION = '1.0'`

`property consumer`

`create()`

`property created_at`

`property deleted`

`property deleted_at`

`destroy`(*force=False*)

Deletes the QoS spec.

Parameters

`force` when force is True, all volume_type mappings for this QoS are deleted. When force is False and volume_type mappings still exist, a QoSSpecsInUse exception is thrown

`fields = {'consumer': QoSConsumerValues(default=back-end,nullable=False, valid_values=('back-end', 'front-end', 'both')), 'created_at': DateTime(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'deleted': Boolean(default=False,nullable=True), 'deleted_at': DateTime(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'id': UUID(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'name': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'specs': Dict(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'updated_at': DateTime(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'volume_types': Object(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True)}`

`property id`

`model`

alias of `QualityOfServiceSpecs`

`property name`

`obj_get_changes()`

Returns a dict of changed fields and their new values.

`obj_load_attr`(*attrname*)

Load an additional attribute from the real object.

This should load self.$attrname and cache any data that might be useful for future load operations.

obj_reset_changes(*fields=None*, *recursive=False*)
Reset the list of fields that have been changed.

Parameters

- **fields** List of fields to reset, or all if None.
- **recursive** Call obj_reset_changes(recursive=True) on any sub-objects within the list of fields being reset.

This is NOT revert to previous values.

Specifying fields on recursive resets will only be honored at the top level. Everything below the top will reset all.

obj_what_changed()
Returns a set of fields that have been modified.

save()
Save the changed fields back to the store.

This is optional for subclasses, but is presented here in the base class for consistency among those that do.

property specs

property updated_at

property volume_types

class QualityOfServiceSpecsList(**args*, ***kwargs*)
Bases: *ObjectListBase*, *CinderObject*

VERSION = '1.0'

fields = {'objects': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}

classmethod get_all(*context*, **args*, ***kwargs*)

property objects

cinder.objects.request_spec module

class RequestSpec(*context=None*, ***kwargs*)
Bases: *CinderObject*, *CinderObjectDictCompat*, *CinderComparableObject*

property CG_backend

VERSION = '1.5'

property availability_zones

property backup_id

property cgsnapshot_id

property consistencygroup_id

fields = {'CG_backend': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'availability_zones': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'backup_id': UUID(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'cgsnapshot_id': UUID(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'consistencygroup_id': UUID(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'group_backend': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'group_id': UUID(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'image_id': UUID(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'operation': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'resource_backend': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'snapshot_id': UUID(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'source_replicaid': UUID(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'source_volid': UUID(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'volume': Object(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'volume_id': UUID(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'volume_properties': Object(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'volume_type': Object(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True)}

classmethod from_primitives(*spec*)

Returns RequestSpec object creating it from legacy dictionary.

FIXME(dulek): This should go away in early O as we stop supporting backward compatibility with M.

property group_backend

property group_id

property image_id

obj_extra_fields = ['resource_properties']

property operation

property resource_backend

property resource_properties

property snapshot_id

property source_replicaid

property source_volid

property volume

property volume_id

property volume_properties

property volume_type

class VolumeProperties(*context=None*, ***kwargs*)

Bases: *CinderObject*, *CinderObjectDictCompat*

VERSION = '1.1'

property attach_status

property availability_zone

property cgsnapshot_id

property consistencygroup_id

property display_description

property display_name

property encryption_key_id

fields = {'attach_status': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'availability_zone': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'cgsnapshot_id': UUID(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'consistencygroup_id': UUID(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'display_description': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'display_name': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'encryption_key_id': UUID(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'group_id': UUID(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'group_type_id': UUID(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'metadata': Dict(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'multiattach': Boolean(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'project_id': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'qos_specs': Dict(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'replication_status': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'reservations': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'size': Integer(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'snapshot_id': UUID(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'source_replicaid': UUID(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'source_volid': UUID(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'status': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'user_id': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'volume_type_id': UUID(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True)}

property group_id

property group_type_id

property metadata

property multiattach

property project_id

property qos_specs

property replication_status

property reservations

property size

property snapshot_id

property source_replicaid

property source_volid

property status

property user_id

property volume_type_id

cinder.objects.service module

class Service(*context=None*, ***kwargs*)

Bases: *CinderPersistentObject*, *CinderObject*, *CinderObjectDictCompat*, *CinderComparableObject*, *ClusteredObject*

OPTIONAL_FIELDS = ('cluster',)

VERSION = '1.6'

property active_backend_id

property availability_zone

property binary

property cluster

property cluster_name

create()

property created_at

property deleted

property deleted_at

destroy()

property disabled

property disabled_reason

fields = {'active_backend_id': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'availability_zone': String(default=cinder,nullable=True), 'binary': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'cluster': Object(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'cluster_name': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'created_at': DateTime(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'deleted': Boolean(default=False,nullable=True), 'deleted_at': DateTime(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'disabled': Boolean(default=False,nullable=True), 'disabled_reason': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'frozen': Boolean(default=False,nullable=False), 'host': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'id': Integer(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'modified_at': DateTime(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'object_current_version': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'replication_status': ReplicationStatus(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True, valid_values=('error', 'enabled', 'disabled', 'not-capable', 'failover-error', 'failing-over', 'failed-over', 'enabling', 'disabling')), 'report_count': Integer(default=0,nullable=False), 'rpc_current_version': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'topic': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'updated_at': DateTime(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'uuid': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}

property frozen

classmethod get_by_args(*context*, *host*, *binary_key*)

classmethod get_by_host_and_topic(*context*, *host*, *topic*, *disabled=False*)

classmethod get_by_uuid(*context*, *service_uuid*)

classmethod get_minimum_obj_version(*context*, *binary=None*)

classmethod get_minimum_rpc_version(*context*, *binary*)

property host

property id

property is_up

Check whether a service is up based on last heartbeat.

model

alias of Service

property modified_at

obj_load_attr(*attrname*)

Load an additional attribute from the real object.

This should load self.$attrname and cache any data that might be useful for future load operations.

property object_current_version

property replication_status

property report_count

property rpc_current_version

save(*retry=True*)

Save the changed fields back to the store.

This is optional for subclasses, but is presented here in the base class for consistency among those that do.

property topic

property updated_at

property uuid

class ServiceList(**args*, ***kwargs*)

Bases: ObjectListBase, CinderObject

VERSION = '1.1'

fields = {'objects': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}

classmethod get_all(*context*, *filters=None*)

classmethod get_all_by_binary(*context*, *binary*, *disabled=None*)

classmethod get_all_by_topic(*context*, *topic*, *disabled=None*)

property objects

cinder.objects.snapshot module

class Snapshot(**args*, ***kwargs*)

Bases: CinderCleanableObject, CinderObject, CinderObjectDictCompat, CinderComparableObject, ClusteredObject

`OPTIONAL_FIELDS = ('volume', 'metadata', 'cgsnapshot', 'group_snapshot')`

`VERSION = '1.6'`

`property cgsnapshot`

`property cgsnapshot_id`

`property cluster_name`

`create()`

`property created_at`

`delete_metadata_key`(*context*, *key*)

`property deleted`

`property deleted_at`

`destroy()`

`property display_description`

`property display_name`

`property encryption_key_id`

```
fields = {'cgsnapshot': Object(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'cgsnapshot_id': UUID(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'created_at': DateTime(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'deleted': Boolean(default=False,nullable=True), 'deleted_at':
DateTime(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'display_description': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'display_name': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'encryption_key_id': UUID(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'group_snapshot': Object(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'group_snapshot_id': UUID(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'id':
UUID(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False),
'metadata': Dict(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False),
'progress': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'project_id': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'provider_auth': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'provider_id': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'provider_location': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'status': SnapshotStatus(default=<class 'oslo_versionedobjects.fields.
UnspecifiedDefault'>,nullable=True,valid_values=('error', 'available',
'creating', 'deleting', 'deleted', 'updating', 'error_deleting',
'unmanaging', 'backing-up', 'restoring')), 'updated_at':
DateTime(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'use_quota': Boolean(default=True,nullable=False), 'user_id':
String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'volume': Object(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'volume_id': UUID(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'volume_size': Integer(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'volume_type_id': UUID(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True)}
```

property group_snapshot

property group_snapshot_id

property host

All cleanable VO must have a host property/attribute.

property id

property metadata

model

alias of `Snapshot`

property name

obj_extra_fields = ['name', 'volume_name']

obj_load_attr(*attrname*)

Load an additional attribute from the real object.

This should load self.$attrname and cache any data that might be useful for future load operations.

obj_reset_changes(*fields=None*)

Reset the list of fields that have been changed.

Parameters

- **fields** List of fields to reset, or all if None.
- **recursive** Call obj_reset_changes(recursive=True) on any sub-objects within the list of fields being reset.

This is NOT revert to previous values.

Specifying fields on recursive resets will only be honored at the top level. Everything below the top will reset all.

obj_what_changed()

Returns a set of fields that have been modified.

property progress

property project_id

property provider_auth

property provider_id

property provider_location

save()

Save the changed fields back to the store.

This is optional for subclasses, but is presented here in the base class for consistency among those that do.

classmethod snapshot_data_get_for_project(*context*, *project_id*, *volume_type_id=None*, *host=None*)

property status

property updated_at

property use_quota

property user_id

property volume

property volume_id

property volume_name

property volume_size

property volume_type_id

class SnapshotList(**args*, ***kwargs*)

Bases: ObjectListBase, CinderObject

VERSION = '1.0'

fields = {'objects': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}

classmethod get_all(*context*, *filters*, *marker=None*, *limit=None*, *sort_keys=None*, *sort_dirs=None*, *offset=None*)

Get all snapshot given some search_opts (filters).

Special filters accepted are host and cluster_name, that refer to the volumes fields.

classmethod get_all_active_by_window(*context*, *begin*, *end*)

classmethod get_all_by_project(*context*, *project_id*, *search_opts*, *marker=None*, *limit=None*, *sort_keys=None*, *sort_dirs=None*, *offset=None*)

classmethod get_all_for_cgsnapshot(*context*, *cgsnapshot_id*)

classmethod get_all_for_group_snapshot(*context*, *group_snapshot_id*)

classmethod get_all_for_volume(*context*, *volume_id*)

classmethod get_by_host(*context*, *host*, *filters=None*)

classmethod get_snapshot_summary(*context*, *project_only*, *filters=None*)

property objects

cinder.objects.volume module

class MetadataObject(*key=None*, *value=None*)

Bases: dict

class Volume(**args*, ***kwargs*)

Bases: *CinderCleanableObject*, *CinderObject*, *CinderObjectDictCompat*, *CinderComparableObject*, *ClusteredObject*

OPTIONAL_FIELDS = ('metadata', 'admin_metadata', 'glance_metadata', 'volume_type', 'volume_attachment', 'consistencygroup', 'snapshots', 'cluster', 'group')

VERSION = '1.9'

property admin_metadata

admin_metadata_update(*metadata*, *delete*, *add=True*, *update=True*)

property attach_status

property availability_zone

begin_attach(*attach_mode*)

property bootable

property cluster

property cluster_name

property consistencygroup

property consistencygroup_id

create()

property created_at

delete_metadata_key(*key*)

property deleted

property deleted_at

destroy()

property display_description

property display_name

property ec2_id

property encryption_key_id

```
fields = {'_name_id': UUID(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'admin_metadata': Dict(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'attach_status': VolumeAttachStatus(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True,
valid_values=('attached', 'attaching', 'detached', 'error_attaching',
'error_detaching', 'reserved', 'deleted')), 'availability_zone':
String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'bootable': Boolean(default=False,nullable=True), 'cluster':
Object(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'cluster_name': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'consistencygroup': Object(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'consistencygroup_id': UUID(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'created_at': DateTime(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'deleted': Boolean(default=False,nullable=True), 'deleted_at':
DateTime(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'display_description': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'display_name': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'ec2_id': UUID(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'encryption_key_id': UUID(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'glance_metadata': Dict(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'group': Object(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'group_id': UUID(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'host':
String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'id':
UUID(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False),
'launched_at': DateTime(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'metadata': Dict(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'migration_status': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'multiattach': Boolean(default=False,nullable=True), 'previous_status':
String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'project_id': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'provider_auth': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'provider_geometry': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
```

finish_detach(*attachment_id*)

finish_volume_migration(*dest_volume*)

get_latest_snapshot()

Get volumes latest snapshot

property glance_metadata

property group

property group_id

property host

property id

is_migration_target()

is_multiattach()

is_replicated()

property launched_at

property metadata

property migration_status

model

alias of `Volume`

property multiattach

property name

property name_id

Actual volumes UUID for driver usage.

There may be two different UUIDs for the same volume, the user facing one, and the one the driver should be using.

When a volume is created these two are the same, but when doing a generic migration (create new volume, then copying data) they will be different if we were unable to rename the new volume in the final migration steps.

So the volume will have been created using the new volumes UUID and the driver will have to look for it using that UUID, but the user on the other hand will keep referencing the volume with the original UUID.

This property facilitates using the right UUID in the drivers code.

obj_extra_fields = ['name', 'name_id', 'volume_metadata', 'volume_admin_metadata', 'volume_glance_metadata']

`obj_load_attr`(*attrname*)

Load an additional attribute from the real object.

This should load self.$attrname and cache any data that might be useful for future load operations.

`obj_reset_changes`(*fields=None*)

Reset the list of fields that have been changed.

Parameters

- **fields** List of fields to reset, or all if None.
- **recursive** Call obj_reset_changes(recursive=True) on any sub-objects within the list of fields being reset.

This is NOT revert to previous values.

Specifying fields on recursive resets will only be honored at the top level. Everything below the top will reset all.

`obj_what_changed`()

Returns a set of fields that have been modified.

`populate_consistencygroup`()

Populate CG fields based on group fields.

Method assumes that consistencygroup_id and consistencygroup fields have not already been set.

This is a hack to support backward compatibility of consistencygroup, where we set the fields but dont want to write them to the DB, so we mark them as not changed, so they wont be stored on the next save().

`property previous_status`

`property project_id`

`property provider_auth`

`property provider_geometry`

`property provider_id`

`property provider_location`

`property replication_driver_data`

`property replication_extended_status`

`property replication_status`

`save`()

Save the changed fields back to the store.

This is optional for subclasses, but is presented here in the base class for consistency among those that do.

`property scheduled_at`

property service_uuid

property shared_targets

property size

property snapshot_id

property snapshots

property source_volid

property status

property terminated_at

property updated_at

property use_quota

property user_id

property volume_admin_metadata

property volume_attachment

property volume_glance_metadata

property volume_metadata

property volume_type

property volume_type_id

class VolumeList(*args, **kwargs)

Bases: ObjectListBase, CinderObject

VERSION = '1.1'

fields = {'objects': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}

classmethod get_all(*context*, *marker=None*, *limit=None*, *sort_keys=None*, *sort_dirs=None*, *filters=None*, *offset=None*)

classmethod get_all_active_by_window(*context*, *begin*, *end*)

classmethod get_all_by_generic_group(*context*, *group_id*, *filters=None*)

classmethod get_all_by_group(*context*, *group_id*, *filters=None*)

classmethod get_all_by_host(*context*, *host*, *filters=None*)

classmethod get_all_by_project(*context*, *project_id*, *marker=None*, *limit=None*, *sort_keys=None*, *sort_dirs=None*, *filters=None*, *offset=None*)

classmethod get_volume_summary(*context*, *project_only*, *filters=None*)

static include_in_cluster(*context*, *cluster*, *partial_rename=True*, ***filters*)

Include all volumes matching the filters into a cluster.

When partial_rename is set we will not set the cluster_name with cluster parameter value directly, well replace provided cluster_name or host filter value with cluster instead.

This is useful when we want to replace just the cluster name but leave the backend and pool information as it is. If we are using cluster_name to filter, well use that same DB field to replace the cluster value and leave the rest as it is. Likewise if we use the host to filter.

Returns the number of volumes that have been changed.

property objects

cinder.objects.volume_attachment module

class VolumeAttachment(*context=None*, ***kwargs*)

Bases: *CinderPersistentObject*, *CinderObject*, *CinderObjectDictCompat*, *CinderComparableObject*

OPTIONAL_FIELDS = ('volume',)

VERSION = '1.3'

property attach_mode

property attach_status

property attach_time

property attached_host

property connection_info

property connector

create()

property created_at

property deleted

property deleted_at

destroy()

property detach_time

```
fields = {'attach_mode': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'attach_status': VolumeAttachStatus(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True,
valid_values=('attached', 'attaching', 'detached', 'error_attaching',
'error_detaching', 'reserved', 'deleted')), 'attach_time':
DateTime(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'attached_host': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'connection_info': Dict(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'connector': Dict(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'created_at': DateTime(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'deleted': Boolean(default=False,nullable=True), 'deleted_at':
DateTime(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'detach_time': DateTime(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'id':
UUID(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False),
'instance_uuid': UUID(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'mountpoint': String(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'updated_at': DateTime(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True),
'volume': Object(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False),
'volume_id': UUID(default=<class
'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}
```

`finish_attach`(*instance_uuid*, *host_name*, *mount_point*, *attach_mode='rw'*)

`property id`

`property instance_uuid`

`model`

alias of `VolumeAttachment`

`property mountpoint`

`obj_extra_fields = ['project_id', 'volume_host']`

`obj_load_attr`(*attrname*)

Load an additional attribute from the real object.

This should load self.$attrname and cache any data that might be useful for future load operations.

property project_id

save()

Save the changed fields back to the store.

This is optional for subclasses, but is presented here in the base class for consistency among those that do.

property updated_at

property volume

property volume_host

property volume_id

class VolumeAttachmentList(**args*, ***kwargs*)

Bases: *ObjectListBase*, *CinderObject*

VERSION = '1.1'

fields = {'objects': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}

classmethod get_all(*context*, *search_opts=None*, *marker=None*, *limit=None*, *offset=None*, *sort_keys=None*, *sort_direction=None*)

classmethod get_all_by_host(*context*, *host*, *search_opts=None*)

classmethod get_all_by_instance_uuid(*context*, *instance_uuid*, *search_opts=None*)

classmethod get_all_by_project(*context*, *project_id*, *search_opts=None*, *marker=None*, *limit=None*, *offset=None*, *sort_keys=None*, *sort_direction=None*)

classmethod get_all_by_volume_id(*context*, *volume_id*)

property objects

cinder.objects.volume_type module

class VolumeType(*context=None*, ***kwargs*)

Bases: *CinderPersistentObject*, *CinderObject*, *CinderObjectDictCompat*, *CinderComparableObject*

OPTIONAL_FIELDS = ('extra_specs', 'projects', 'qos_specs')

VERSION = '1.3'

create()

property created_at

property deleted

property deleted_at

property description

destroy()

property extra_specs

fields = {'created_at': DateTime(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'deleted': Boolean(default=False,nullable=True), 'deleted_at': DateTime(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'description': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'extra_specs': Dict(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'id': UUID(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False), 'is_public': Boolean(default=True,nullable=True), 'name': String(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'projects': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'qos_specs': Object(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'qos_specs_id': UUID(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True), 'updated_at': DateTime(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=True)}

classmethod get_by_name_or_id(*context*, *identity*)

property id

is_multiattach()

property is_public

is_replicated()

model

alias of VolumeType

property name

obj_load_attr(*attrname*)

Load an additional attribute from the real object.

This should load self.$attrname and cache any data that might be useful for future load operations.

property projects

property qos_specs

property qos_specs_id

save()

Save the changed fields back to the store.

This is optional for subclasses, but is presented here in the base class for consistency among those that do.

property updated_at

class VolumeTypeList(**args*, ***kwargs*)

Bases: ObjectListBase, CinderObject

VERSION = '1.1'

fields = {'objects': List(default=<class 'oslo_versionedobjects.fields.UnspecifiedDefault'>,nullable=False)}

classmethod get_all(*context*, *inactive=0*, *filters=None*, *marker=None*, *limit=None*, *sort_keys=None*, *sort_dirs=None*, *offset=None*)

classmethod get_all_by_group(*context*, *group_id*)

classmethod get_all_types_for_qos(*context*, *qos_id*)

property objects

Module contents

register_all()

cinder.policies package

Submodules

cinder.policies.attachments module

list_rules()

cinder.policies.backup_actions module

list_rules()

cinder.policies.backups module

list_rules()

cinder.policies.base module

class CinderDeprecatedRule(*name: str*, *check_str: str*, ***, *deprecated_reason: str | None = 'Default policies now support the three Keystone default roles, namely \'admin\', \'member\', and \'reader\' to implement three Cinder "personas". See "Policy Personas and Permissions" in the "Cinder Service Configuration" documentation (Xena release) for details.'*, *deprecated_since: str | None = 'X'*)

Bases: `DeprecatedRule`

A DeprecatedRule subclass with pre-defined fields.

`list_rules()`

cinder.policies.capabilities module

`list_rules()`

cinder.policies.clusters module

`list_rules()`

cinder.policies.default_types module

`list_rules()`

cinder.policies.group_actions module

`list_rules()`

cinder.policies.group_snapshot_actions module

`list_rules()`

cinder.policies.group_snapshots module

`list_rules()`

cinder.policies.group_types module

`list_rules()`

cinder.policies.groups module

`list_rules()`

cinder.policies.hosts module

`list_rules()`

cinder.policies.limits module

`list_rules()`

cinder.policies.manageable_snapshots module

`list_rules()`

cinder.policies.manageable_volumes module

`list_rules()`

cinder.policies.messages module

`list_rules()`

cinder.policies.qos_specs module

`list_rules()`

cinder.policies.quota_class module

`list_rules()`

cinder.policies.quotas module

`list_rules()`

cinder.policies.scheduler_stats module

`list_rules()`

cinder.policies.services module

`list_rules()`

cinder.policies.snapshot_actions module

`list_rules()`

cinder.policies.snapshot_metadata module

`list_rules()`

cinder.policies.snapshots module

`list_rules()`

cinder.policies.type_extra_specs module

`list_rules()`

cinder.policies.volume_access module

`list_rules()`

cinder.policies.volume_actions module

`list_rules()`

cinder.policies.volume_metadata module

`list_rules()`

cinder.policies.volume_transfer module

`list_rules()`

cinder.policies.volume_type module

`list_rules()`

cinder.policies.volumes module

`list_rules()`

cinder.policies.workers module

`list_rules()`

Module contents

`list_rules()`

cinder.privsep package

Subpackages

cinder.privsep.targets package

Submodules

cinder.privsep.targets.nvmet module

NVMet Python Interface using privsep

This file adds the privsep support to the nvmet package so it can be easily consumed by Cinder nvmet target.

It also:

- Adds some methods to the Root class to be able to get a specific subsystem or port directly without having to go through all the existing ones.
- Presents the CFSNotFound exception as a NotFound exception which is easier to consume.

class Namespace(**args: Any*, ***kwargs: Any*)

Bases: `Namespace`

delete()

classmethod setup(*subsys*, *n*, *err_func=None*)

class Port(**args: Any*, ***kwargs: Any*)

Bases: `Port`

add_subsystem(*nqn*)

delete()

remove_subsystem(*nqn*)

classmethod setup(*root*, *n*, *err_func=None*)

class Root(**args: Any*, ***kwargs: Any*)

Bases: `Root`

property ports

class Subsystem(**args: Any*, ***kwargs: Any*)

Bases: `Subsystem`

delete()

property namespaces

classmethod setup(*t*, *err_func=None*)

deserialize(*data*)

Deserialize an instance, specially nvmet instances.

Reverse operation of the serialize method. Converts an nvmet instance serialized in a tuple into an actual nvmet instance.

deserialize_params(*args*, *kwargs*)

Deserialize function arguments using deserialize method.

privsep_setup(*cls_name*, **args*, ***kwargs*)

Wrapper for _privsep_setup that accepts err_func argument.

serialize(*instance*)

Serialize parameters, specially nvmet instances.

The idea is to be able to pass an nvmet instance to privsep methods, since they are sometimes required as parameters (ie: port.setup) and also to pass the instance where do_privsep_call has to call a specific method.

Instances are passed as a tuple, with the name of the class as the first element, and in the second element the kwargs necessary to instantiate the instance of that class.

To differentiate nvmet instances from tuples there is a tuple value that can be passed in the first element of the tuple to differentiate them.

All other instances as passed unaltered.

cinder.privsep.targets.scst module

Helpers for scst related routines.

cinder.privsep.targets.tgt module

Helpers for iscsi related routines.

Module contents

Submodules

cinder.privsep.cgroup module

Helpers for cgroup related routines.

cinder.privsep.format_inspector module

Helpers for the image format_inspector.

cinder.privsep.fs module

Helpers for filesystem related routines.

cinder.privsep.lvm module

Helpers for lvm related routines

cinder.privsep.path module

Helpers for path related routines.

Module contents

Setup privsep decorator.

cinder.scheduler package

Subpackages

cinder.scheduler.evaluator package

Submodules

cinder.scheduler.evaluator.evaluator module

class EvalAddOp(*toks*)

Bases: `object`

eval()

class EvalBoolAndOp(*toks*)

Bases: `object`

eval()

class **EvalBoolOrOp**(*toks*)

Bases: `object`

eval()

class **EvalCommaSeperator**(*toks*)

Bases: `object`

eval()

class **EvalComparisonOp**(*toks*)

Bases: `object`

eval()

operations = {'!=': <built-in function ne>, '<': <built-in function lt>, '<=': <built-in function le>, '<>': <built-in function ne>, '==': <built-in function eq>, '>': <built-in function gt>, '>=': <built-in function ge>}

class **EvalConstant**(*toks*)

Bases: `object`

eval()

class **EvalFunction**(*toks*)

Bases: `object`

eval()

functions: dict[str, Callable] = {'abs': <built-in function abs>, 'max': <built-in function max>, 'min': <built-in function min>}

class **EvalMultOp**(*toks*)

Bases: `object`

eval()

class **EvalNegateOp**(*toks*)

Bases: `object`

eval()

class **EvalPowerOp**(*toks*)

Bases: `object`

eval()

class **EvalSignOp**(*toks*)

Bases: `object`

eval()

operations = {'+': 1, '-': -1}

class EvalTernaryOp(*toks*)

Bases: `object`

eval()

evaluate(*expression*, ***kwargs*)

Evaluates an expression.

Provides the facility to evaluate mathematical expressions, and to substitute variables from dictionaries into those expressions.

Supports both integer and floating point values, and automatic promotion where necessary.

Module contents

cinder.scheduler.filters package

Submodules

cinder.scheduler.filters.affinity_filter module

class AffinityFilter

Bases: *BaseBackendFilter*

class DifferentBackendFilter

Bases: *AffinityFilter*

Schedule volume on a different back-end from a set of volumes.

backend_passes(*backend_state*, *filter_properties*)

Return True if the HostState passes the filter, otherwise False.

Override this in a subclass.

class SameBackendFilter

Bases: *AffinityFilter*

Schedule volume on the same back-end as another volume.

backend_passes(*backend_state*, *filter_properties*)

Return True if the HostState passes the filter, otherwise False.

Override this in a subclass.

cinder.scheduler.filters.availability_zone_filter module

class AvailabilityZoneFilter

Bases: *BaseBackendFilter*

Filters Backends by availability zone.

backend_passes(*backend_state*, *filter_properties*)

Return True if the HostState passes the filter, otherwise False.

Override this in a subclass.

run_filter_once_per_request = True

cinder.scheduler.filters.capabilities_filter module

class CapabilitiesFilter

Bases: *BaseBackendFilter*

BackendFilter to work with resource (instance & volume) type records.

backend_passes(*backend_state*, *filter_properties*)

Return a list of backends that can create resource_type.

cinder.scheduler.filters.capacity_filter module

class CapacityFilter

Bases: *BaseBackendFilter*

Capacity filters based on volume backends capacity utilization.

backend_passes(*backend_state*, *filter_properties*)

Return True if host has sufficient capacity.

cinder.scheduler.filters.driver_filter module

class DriverFilter

Bases: *BaseBackendFilter*

DriverFilter filters backend based on a filter function and metrics.

DriverFilter filters based on volume backends provided filter function and metrics.

backend_passes(*backend_state*, *filter_properties*)

Determines if a backend has a passing filter_function or not.

cinder.scheduler.filters.extra_specs_ops module

match(*value*, *req*)

cinder.scheduler.filters.ignore_attempted_hosts_filter module

class IgnoreAttemptedHostsFilter

Bases: *BaseBackendFilter*

Filter out previously attempted hosts

A host passes this filter if it has not already been attempted for scheduling. The scheduler needs to add previously attempted hosts to the retry key of filter_properties in order for this to work correctly. For example:

```
{
 'retry': {
           'backends': ['backend1', 'backend2'],
           'num_attempts': 3,
          }
}
```

backend_passes(*backend_state*, *filter_properties*)

Skip nodes that have already been attempted.

cinder.scheduler.filters.instance_locality_filter module

class InstanceLocalityFilter

Bases: *BaseBackendFilter*

Schedule volume on the same host as a given instance.

This filter enables selection of a storage back-end located on the host where the instances hypervisor is running. This provides data locality: the instance and the volume are located on the same physical machine.

In order to work:

- The Extended Server Attributes extension needs to be active in Nova (this is by default), so that the OS-EXT-SRV-ATTR:host property is returned when requesting instance info.
- Either an account with privileged rights for Nova must be configured in Cinder configuration (configure a keystone authentication plugin in the [nova] section), or the user making the call needs to have sufficient rights (see extended_server_attributes in Nova policy).

backend_passes(*backend_state*, *filter_properties*)

Return True if the HostState passes the filter, otherwise False.

Override this in a subclass.

cinder.scheduler.filters.json_filter module

class JsonFilter

Bases: *BaseBackendFilter*

Backend filter for simple JSON-based grammar for selecting backends.

If you want to choose one of your backend, make a query hint, for example:

cinder create hint query=[=, $backend_id, rbd:vol@ceph#cloud]

backend_passes(*backend_state*, *filter_properties*)

Return a list of backends that can fulfill query requirements.

commands = {'<': <function JsonFilter._less_than>, '<=': <function JsonFilter._less_than_equal>, '=': <function JsonFilter._equals>, '>': <function JsonFilter._greater_than>, '>=': <function JsonFilter._greater_than_equal>, 'and': <function JsonFilter._and>, 'in': <function JsonFilter._in>, 'not': <function JsonFilter._not>, 'or': <function JsonFilter._or>}

Module contents

Scheduler host filters

class BackendFilterHandler(*namespace*)

Bases: *BaseFilterHandler*

class BaseBackendFilter

Bases: *BaseFilter*

Base class for host filters.

backend_passes(*host_state*, *filter_properties*)

Return True if the HostState passes the filter, otherwise False.

Override this in a subclass.

BaseHostFilter

alias of *BaseBackendFilter*

HostFilterHandler

alias of *BackendFilterHandler*

cinder.scheduler.flows package

Submodules

cinder.scheduler.flows.create_volume module

class ExtractSchedulerSpecTask(***kwargs*)

Bases: *CinderTask*

Extracts a spec object from a partial and/or incomplete request spec.

Reversion strategy: N/A

default_provides = {'request_spec'}

execute(*context: RequestContext*, *request_spec: dict | None*, *volume: Volume*, *snapshot_id: str | None*, *image_id: str | None*, *backup_id: str | None*) → dict[str, Any]

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** positional arguments that atom requires to execute.
- **kwargs** any keyword arguments that atom requires to execute.

class ScheduleCreateVolumeTask(*driver_api*, ***kwargs*)

Bases: *CinderTask*

Activates a scheduler driver and handles any subsequent failures.

Notification strategy: on failure the scheduler rpc notifier will be activated and a notification will be emitted indicating what errored, the reason, and the request (and misc. other data) that caused the error to be triggered.

Reversion strategy: N/A

FAILURE_TOPIC = 'scheduler.create_volume'

execute(*context: RequestContext*, *request_spec: dict*, *filter_properties: dict*, *volume: Volume*) → None

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** positional arguments that atom requires to execute.
- **kwargs** any keyword arguments that atom requires to execute.

get_flow(*context: RequestContext*, *driver_api*, *request_spec: dict | None = None*, *filter_properties: dict | None = None*, *volume: Volume | None = None*, *snapshot_id: str | None = None*, *image_id: str | None = None*, *backup_id: str | None = None*) → Engine

Constructs and returns the scheduler entrypoint flow.

This flow will do the following:

1. Inject keys & values for dependent tasks.
2. Extract a scheduler specification from the provided inputs.
3. Use provided scheduler driver to select host and pass volume creation request further.

Module contents

cinder.scheduler.weights package

Submodules

cinder.scheduler.weights.capacity module

class AllocatedCapacityWeigher

Bases: *BaseHostWeigher*

Allocated Capacity Weigher weighs hosts by their allocated capacity.

The default behavior is to place new volume to the host allocated the least space. This weigher is intended to simulate the behavior of SimpleScheduler. If you prefer to place volumes to host allocated the most space, you can set the `allocated_capacity_weight_multiplier` option to a positive number and the weighing has the opposite effect of the default.

weight_multiplier() → float

Override the weight multiplier.

class CapacityWeigher

Bases: *BaseHostWeigher*

Capacity Weigher weighs hosts by their virtual or actual free capacity.

For thin provisioning, weigh hosts by their virtual free capacity calculated by the total capacity multiplied by the max over subscription ratio and subtracting the provisioned capacity; Otherwise, weigh hosts by their actual free capacity, taking into account the reserved space.

The default is to spread volumes across all hosts evenly. If you prefer stacking, you can set the `capacity_weight_multiplier` option to a negative number and the weighing has the opposite effect of the default.

weigh_objects(*weighed_obj_list*, *weight_properties*)

Override the weigh objects.

This override calls the parent to do the weigh objects and then replaces any infinite weights with a value that is a multiple of the delta between the min and max values.

NOTE(jecarey): the infinite weight value is only used when the smallest value is being favored (negative multiplier). When the largest weight value is being used a weight of -1 is used instead. See _weigh_object method.

weight_multiplier() → float

Override the weight multiplier.

cinder.scheduler.weights.chance module

class ChanceWeigher

Bases: *BaseHostWeigher*

Chance Weigher assigns random weights to hosts.

Used to spread volumes randomly across a list of equally suitable hosts.

cinder.scheduler.weights.goodness module

class GoodnessWeigher

Bases: *BaseHostWeigher*

Goodness Weigher. Assign weights based on a hosts goodness function.

Goodness rating is the following:

```
  0 -- host is a poor choice
  .
  .
 50 -- host is a good choice
  .
  .
100 -- host is a perfect choice
```

cinder.scheduler.weights.stochastic module

Stochastic weight handler

This weight handler differs from the default weight handler by giving every pool a chance to be chosen where the probability is proportional to each pools weight.

class StochasticHostWeightHandler(*namespace*)

Bases: *BaseWeightHandler*

get_weighed_objects(*weigher_classes*, *obj_list*, *weighing_properties*)

Return a sorted (descending), normalized list of WeighedObjects.

cinder.scheduler.weights.volume_number module

class VolumeNumberWeigher

Bases: *BaseHostWeigher*

Weigher that weighs hosts by volume number in backends.

The default is to spread volumes across all hosts evenly. If you prefer stacking, you can set the `volume_number_multiplier` option to a positive number and the weighing has the opposite effect of the default.

weight_multiplier() → float

Override the weight multiplier.

Module contents

Scheduler host weights

class BaseHostWeigher

Bases: *BaseWeigher*

Base class for host weights.

class OrderedHostWeightHandler(*namespace*)

Bases: *BaseWeightHandler*

object_class

alias of *WeighedHost*

class WeighedHost(*obj*, *weight: float*)

Bases: *WeighedObject*

to_dict()

Submodules

cinder.scheduler.base_filter module

Filter support

class BaseFilter

Bases: `object`

Base class for all filter classes.

filter_all(*filter_obj_list*, *filter_properties*)

Yield objects that pass the filter.

Can be overridden in a subclass, if you need to base filtering decisions on all objects. Otherwise, one can just override _filter_one() to filter a single object.

run_filter_for_index(*index*)

Return True if the filter needs to be run for n-th instances.

Only need to override this if a filter needs anything other than first only or all behaviour.

run_filter_once_per_request = False

class **BaseFilterHandler**(*modifier_class_type*, *modifier_namespace*)

Bases: `BaseHandler`

Base class to handle loading filter classes.

This class should be subclassed where one needs to use filters.

get_filtered_objects(*filter_classes*, *objs: Iterable*, *filter_properties: dict*, *index: int = 0*) → list

Get objects after filter

Parameters

- **filter_classes** filters that will be used to filter the objects
- **objs** objects that will be filtered
- **filter_properties** client filter properties
- **index** This value needs to be increased in the caller function of get_filtered_objects when handling each resource.

cinder.scheduler.base_handler module

A common base for handling extension classes.

Used by BaseFilterHandler and BaseWeightHandler

class **BaseHandler**(*modifier_class_type*, *modifier_namespace*)

Bases: `object`

Base class to handle loading filter and weight classes.

get_all_classes() → list

cinder.scheduler.base_weight module

Pluggable Weighing support

class **BaseWeigher**

Bases: `object`

Base class for pluggable weighers.

The attributes maxval and minval can be specified to set up the maximum and minimum values for the weighed objects. These values will then be taken into account in the normalization step, instead of taking the values from the calculated weights.

maxval: float | None = None

minval: float | None = None

weigh_objects(*weighed_obj_list: list[WeighedObject], weight_properties: dict*) → list[float]

Weigh multiple objects.

Override in a subclass if you need access to all objects in order to calculate weights. Do not modify the weight of an object here, just return a list of weights.

weight_multiplier() → float

How weighted this weigher should be.

Override this method in a subclass, so that the returned value is read from a configuration option to permit operators specify a multiplier for the weigher.

class BaseWeightHandler(*modifier_class_type, modifier_namespace*)

Bases: `BaseHandler`

get_weighed_objects(*weigher_classes: list, obj_list: list[WeighedObject], weighing_properties: dict*) → list[*WeighedObject*]

Return a sorted (descending), normalized list of WeighedObjects.

object_class

alias of `WeighedObject`

class WeighedObject(*obj, weight: float*)

Bases: `object`

Object with weight information.

normalize(*weight_list: list[float], minval: float | None = None, maxval: float | None = None*) → Iterable[float]

Normalize the values in a list between 0 and 1.0.

The normalization is made regarding the lower and upper values present in weight_list. If the minval and/or maxval parameters are set, these values will be used instead of the minimum and maximum from the list.

If all the values are equal, they are normalized to 0.

cinder.scheduler.driver module

Scheduler base class that all Schedulers should inherit from

class Scheduler

Bases: `object`

The base class that all Scheduler classes should inherit from.

backend_passes_filters(*context, backend, request_spec, filter_properties*)

Check if the specified backend passes the filters.

find_retype_backend(*context, request_spec, filter_properties=None, migration_policy='never'*)

Find a backend that can accept the volume with its new type.

find_retype_host(*context*, *request_spec*, *filter_properties=None*, *migration_policy='never'*)

Find a backend that can accept the volume with its new type.

get_backup_host(*volume*, *availability_zone*, *driver=None*)

Must override schedule method for scheduler to work.

get_pools(*context*, *filters*)

Must override schedule method for scheduler to work.

host_passes_filters(*context*, *backend*, *request_spec*, *filter_properties*)

Check if the specified backend passes the filters.

is_first_receive()

Returns True if Scheduler receives the capabilities at startup.

This is to handle the problem of too long sleep time during scheduler service startup process.

is_ready()

Returns True if Scheduler is ready to accept requests.

This is to handle scheduler service startup when it has no volume hosts stats and will fail all the requests.

notify_service_capabilities(*service_name*, *backend*, *capabilities*, *timestamp*)

Notify capability update from a service node.

reset()

Reset volume RPC API object to load new version pins.

schedule(*context*, *topic*, *method*, **_args*, ***_kwargs*)

Must override schedule method for scheduler to work.

schedule_create_group(*context*, *group*, *group_spec*, *request_spec_list*, *group_filter_properties*, *filter_properties_list*)

Must override schedule method for scheduler to work.

schedule_create_volume(*context*, *request_spec*, *filter_properties*)

Must override schedule method for scheduler to work.

update_service_capabilities(*service_name*, *host*, *capabilities*, *cluster_name*, *timestamp*)

Process a capability update from a service node.

generic_group_update_db(*context*, *group*, *host*, *cluster_name*)

Set the host and the scheduled_at field of a group.

Returns

A Group with the updated fields set properly.

volume_update_db(*context*, *volume_id*, *host*, *cluster_name*, *availability_zone=None*, *volume=None*)

Set the host, cluster_name, and set the scheduled_at field of a volume.

Returns

A Volume with the updated fields set properly.

cinder.scheduler.filter_scheduler module

The FilterScheduler is for creating volumes.

You can customize this scheduler by specifying your own volume Filters and Weighing Functions.

class FilterScheduler(**args*, ***kwargs*)

Bases: *Scheduler*

Scheduler that can be used for filtering and weighing.

backend_passes_filters(*context: RequestContext*, *backend: str*, *request_spec: dict*, *filter_properties: dict*)

Check if the specified backend passes the filters.

find_retype_backend(*context: RequestContext*, *request_spec: dict*, *filter_properties: dict | None = None*, *migration_policy: str = 'never'*) → *BackendState*

Find a backend that can accept the volume with its new type.

get_backup_host(*volume: Volume*, *availability_zone: str | None*, *driver=None*)

Must override schedule method for scheduler to work.

get_pools(*context: RequestContext*, *filters: dict*)

Must override schedule method for scheduler to work.

populate_filter_properties(*request_spec: dict*, *filter_properties: dict*) → None

Stuff things into filter_properties.

Can be overridden in a subclass to add more data.

schedule_create_group(*context: RequestContext*, *group*, *group_spec*, *request_spec_list*, *group_filter_properties*, *filter_properties_list*) → None

Must override schedule method for scheduler to work.

schedule_create_volume(*context: RequestContext*, *request_spec: dict*, *filter_properties: dict*) → None

Must override schedule method for scheduler to work.

cinder.scheduler.host_manager module

Manage backends in the current zone.

class BackendState(*host: str*, *cluster_name: str | None*, *capabilities: ReadOnlyDict | None | dict = None*, *service=None*)

Bases: object

Mutable and immutable information tracked for a volume backend.

property backend_id: str

consume_from_volume(*volume: Volume*, *update_time: bool = True*) → None

Incrementally update host state from a volume.

static get_storage_protocol_variants(*storage_protocol*)

update_backend(*capability: dict*) → None

update_capabilities(*capabilities: ReadOnlyDict | None | dict = None*, *service: dict | None = None*) → None

update_from_volume_capability(*capability: dict[str, Any]*, *service=None*) → None

Update information about a host from its volume_node info.

capability is the status info reported by volume backend, a typical capability looks like this:

```
{
 capability = {
     'volume_backend_name': 'Local iSCSI', #
     'vendor_name': 'OpenStack',           #  backend level
     'driver_version': '1.0',              #  mandatory/fixed
     'storage_protocol': 'iSCSI',          #  stats&capabilities

     'active_volumes': 10,                 #
     'IOPS_provisioned': 30000,            #  optional custom
     'fancy_capability_1': 'eat',          #  stats & capabilities
     'fancy_capability_2': 'drink',        #

     'pools': [
         {'pool_name': '1st pool',         #
          'total_capacity_gb': 500,        #  mandatory stats for
          'free_capacity_gb': 230,         #  pools
          'allocated_capacity_gb': 270,    #
          'QoS_support': 'False',          #
          'reserved_percentage': 0,        #

          'dying_disks': 100,              #
          'super_hero_1': 'spider-man',    #  optional custom
          'super_hero_2': 'flash',         #  stats & capabilities
          'super_hero_3': 'neoncat'        #
         },
         {'pool_name': '2nd pool',
          'total_capacity_gb': 1024,
          'free_capacity_gb': 1024,
          'allocated_capacity_gb': 0,
          'QoS_support': 'False',
          'reserved_percentage': 0,

          'dying_disks': 200,
          'super_hero_1': 'superman',
          'super_hero_2': ' ',
          'super_hero_2': 'Hulk'
         }
     ]
 }
}
```

update_pools(*capability: dict | None*, *service*) → None

Update storage pools information from backend reported info.

class HostManager

Bases: `object`

Base HostManager class.

ALLOWED_SERVICE_NAMES = ('volume', 'backup')

REQUIRED_KEYS = frozenset({'allocated_capacity_gb', 'free_capacity_gb', 'max_over_subscription_ratio', 'pool_name', 'provisioned_capacity_gb', 'reserved_percentage', 'thick_provisioning_support', 'thin_provisioning_support', 'total_capacity_gb'})

backend_state_cls

alias of `BackendState`

first_receive_capabilities() → bool

get_all_backend_states(*context: RequestContext*) → Iterable

Returns a dict of all the backends the HostManager knows about.

Each of the consumable resources in BackendState are populated with capabilities scheduler received from RPC.

For example:
{192.168.1.100: BackendState(), }

get_az(*volume: Volume*, *availability_zone: str | None*) → str | None

get_backup_host(*volume: Volume*, *availability_zone: str | None*, *driver=None*) → str

get_filtered_backends(*backends*, *filter_properties*, *filter_class_names=None*)

Filter backends and return only ones passing all filters.

get_pools(*context: RequestContext*, *filters: dict | None = None*) → list[dict]

Returns a dict of all pools on all hosts HostManager knows about.

get_usage_and_notify(*capa_new: dict*, *updated_pools: Iterable[dict]*, *host: str*, *timestamp*) → None

get_weighed_backends(*backends*, *weight_properties*, *weigher_class_names=None*) → list

Weigh the backends.

has_all_capabilities() → bool

notify_service_capabilities(*service_name*, *backend*, *capabilities*, *timestamp*)

Notify the ceilometer with updated volume stats

revert_volume_consumed_capacity(*pool_name: str*, *size: int*) → None

update_service_capabilities(*service_name: str*, *host: str*, *capabilities: dict*, *cluster_name: str | None*, *timestamp*) → None

Update the per-service capabilities based on this notification.

class PoolState(*host: str*, *cluster_name: str | None*, *capabilities: ReadOnlyDict | None | dict*, *pool_name: str*)

Bases: `BackendState`

update_from_volume_capability(*capability: dict[str, Any]*, *service=None*) → None

Update information about a pool from its volume_node info.

update_pools(*capability*)

Update storage pools information from backend reported info.

class ReadOnlyDict(*source:* ReadOnlyDict | *None* | *dict = None*)

Bases: `Mapping`

A read-only dict.

cinder.scheduler.manager module

Scheduler Service

class SchedulerManager(*scheduler_driver=None*, *service_name=None*, **args*, ***kwargs*)

Bases: `CleanableManager`, `Manager`

Chooses a host to create volumes.

RPC_API_VERSION = '3.12'

create_backup(*context*, *backup*)

create_group(*context*, *group*, *group_spec=None*, *group_filter_properties=None*, *request_spec_list=None*, *filter_properties_list=None*)

create_snapshot(*ctxt*, *volume*, *snapshot*, *backend*, *request_spec=None*, *filter_properties=None*)

Create snapshot for a volume.

The main purpose of this method is to check if target backend (of volume and snapshot) has sufficient capacity to host to-be-created snapshot.

create_volume(*context*, *volume*, *snapshot_id=None*, *image_id=None*, *request_spec=None*, *filter_properties=None*, *backup_id=None*)

extend_volume(*context*, *volume*, *new_size*, *reservations*, *request_spec=None*, *filter_properties=None*)

get_pools(*context*, *filters=None*)

Get active pools from schedulers cache.

NOTE(dulek): Theres no self._wait_for_scheduler() because get_pools is an RPC call (is blocking for the c-api). Also this is admin-only API extension so it wont hurt the user much to retry the request manually.

init_host_with_rpc()

A hook for service to do jobs after RPC is ready.

Like init_host(), this method is a hook where services get a chance to execute tasks that *need* RPC. Child classes should override this method.

manage_existing(*context*, *volume*, *request_spec*, *filter_properties=None*)

Ensure that the host exists and can accept the volume.

manage_existing_snapshot(*context*, *volume*, *snapshot*, *ref*, *request_spec*, *filter_properties=None*)

Ensure that the host exists and can accept the snapshot.

migrate_volume(*context: RequestContext*, *volume: Volume*, *backend: str*, *force_copy: bool*, *request_spec*, *filter_properties*) → None

Ensure that the backend exists and can accept the volume.

migrate_volume_to_host(*context*, *volume*, *host*, *force_host_copy*, *request_spec*, *filter_properties=None*)

notify_service_capabilities(*context*, *service_name*, *capabilities*, *host=None*, *backend=None*, *timestamp=None*)

Process a capability update from a service node.

request_service_capabilities(*context: RequestContext*) → None

reset()

Method executed when SIGHUP is caught by the process.

Were utilizing it to reset RPC API version pins to avoid restart of the service when rolling upgrade is completed.

retype(*context*, *volume*, *request_spec*, *filter_properties=None*)

Schedule the modification of a volumes type.

Parameters

- **context** the request context
- **volume** the volume object to retype
- **request_spec** parameters for this retype request
- **filter_properties** parameters to filter by

target = <Target version=3.12>

update_service_capabilities(*context*, *service_name=None*, *host=None*, *capabilities=None*, *cluster_name=None*, *timestamp=None*, ***kwargs*)

Process a capability update from a service node.

property upgrading_cloud

validate_host_capacity(*context*, *backend*, *request_spec*, *filter_properties*)

work_cleanup(*context*, *cleanup_request*)

Process request from API to do cleanup on services.

Here we retrieve from the DB which services we want to clean up based on the request from the user.

Then send individual cleanup requests to each of the services that are up, and we finally return a tuple with services that we have sent a cleanup request and those that were not up and we couldnt send it.

append_operation_type(*name=None*)

cinder.scheduler.rpcapi module

Client side of the scheduler manager RPC API.

class SchedulerAPI

Bases: RPCAPI

Client side of the scheduler RPC API.

API version history:

```
1.0 - Initial version.
1.1 - Add create_volume() method
1.2 - Add request_spec, filter_properties arguments to
      create_volume()
1.3 - Add migrate_volume_to_host() method
1.4 - Add retype method
1.5 - Add manage_existing method
1.6 - Add create_consistencygroup method
1.7 - Add get_active_pools method
1.8 - Add sending object over RPC in create_consistencygroup method
1.9 - Adds support for sending objects over RPC in create_volume()
1.10 - Adds support for sending objects over RPC in retype()
1.11 - Adds support for sending objects over RPC in
       migrate_volume_to_host()

... Mitaka supports messaging 1.11. Any changes to existing methods in
1.x after this point should be done so that they can handle version cap
set to 1.11.

2.0 - Remove 1.x compatibility
2.1 - Adds support for sending objects over RPC in manage_existing()
2.2 - Sends request_spec as object in create_volume()
2.3 - Add create_group method

... Newton supports messaging 2.3. Any changes to existing methods in
2.x after this point should be done so that they can handle version cap
set to 2.3.

3.0 - Remove 2.x compatibility
3.1 - Adds notify_service_capabilities()
3.2 - Adds extend_volume()
3.3 - Add cluster support to migrate_volume, and to
      update_service_capabilities and send the timestamp from the
      capabilities.
3.4 - Adds work_cleanup and do_cleanup methods.
3.5 - Make notify_service_capabilities support A/A
3.6 - Removed create_consistencygroup method
3.7 - Adds set_log_levels and get_log_levels
3.8 - Addds ``valid_host_capacity`` method
3.9 - Adds create_snapshot method
3.10 - Adds backup_id to create_volume method.
```

(continues on next page)

(continued from previous page)

```
3.11 - Adds manage_existing_snapshot method.
3.12 - Adds create_backup method.
```

BINARY = 'cinder-scheduler'

RPC_API_VERSION = '3.12'

RPC_DEFAULT_VERSION = '3.0'

TOPIC = 'cinder-scheduler'

create_backup(*ctxt*, *backup*)

create_group(*ctxt*, *group*, *group_spec=None*, *request_spec_list=None*, *group_filter_properties=None*, *filter_properties_list=None*)

create_snapshot(*ctxt*, *volume*, *snapshot*, *backend*, *request_spec=None*, *filter_properties=None*)

create_volume(*ctxt*, *volume*, *snapshot_id=None*, *image_id=None*, *request_spec=None*, *filter_properties=None*, *backup_id=None*)

do_cleanup(*ctxt*, *cleanup_request*)

Perform this schedulers resource cleanup as per cleanup_request.

extend_volume(*ctxt*, *volume*, *new_size*, *reservations*, *request_spec*, *filter_properties=None*)

get_log_levels(*context*, *service*, *log_request*)

get_pools(*ctxt*, *filters=None*)

manage_existing(*ctxt*, *volume*, *request_spec=None*, *filter_properties=None*)

manage_existing_snapshot(*ctxt*, *volume*, *snapshot*, *ref*, *request_spec=None*, *filter_properties=None*)

migrate_volume(*ctxt*, *volume*, *backend*, *force_copy=False*, *request_spec=None*, *filter_properties=None*)

notify_service_capabilities(*ctxt*, *service_name*, *backend*, *capabilities*, *timestamp=None*)

static prepare_timestamp(*timestamp*)

retype(*ctxt*, *volume*, *request_spec=None*, *filter_properties=None*)

set_log_levels(*context*, *service*, *log_request*)

update_service_capabilities(*ctxt*, *service_name*, *host*, *capabilities*, *cluster_name*, *timestamp=None*)

validate_host_capacity(*ctxt*, *backend*, *request_spec*, *filter_properties=None*)

work_cleanup(*ctxt*, *cleanup_request*)

Generate individual service cleanup requests from user request.

cinder.scheduler.scheduler_options module

SchedulerOptions monitors a local .json file for changes and loads it if needed. This file is converted to a data structure and passed into the filtering and weighing functions which can use it for dynamic configuration.

class SchedulerOptions

Bases: `object`

SchedulerOptions monitors a local .json file for changes.

The file is reloaded if needed and converted to a data structure and passed into the filtering and weighing functions which can use it for dynamic configuration.

get_configuration(*filename=None*) → dict

Check the json file for changes and load it if needed.

Module contents

cinder.transfer package

Submodules

cinder.transfer.api module

Handles all requests relating to transferring ownership of volumes.

class API

Bases: `Base`

API for interacting volume transfers.

accept(*context*, *transfer_id*, *auth_key*)

Accept a volume that has been offered for transfer.

create(*context*, *volume_id*, *display_name*, *no_snapshots=False*, *allow_encrypted=False*)

Creates an entry in the transfers table.

delete(*context*, *transfer_id*)

Make the RPC call to delete a volume transfer.

get(*context*, *transfer_id*)

get_all(*context*, *marker=None*, *limit=None*, *sort_keys=None*, *sort_dirs=None*, *filters=None*, *offset=None*)

Module contents

cinder.volume package

Subpackages

cinder.volume.flows package

Subpackages

cinder.volume.flows.api package

Submodules

cinder.volume.flows.api.create_volume module

class EntryCreateTask

Bases: *CinderTask*

Creates an entry for the given volume creation in the database.

Reversion strategy: remove the volume_id created from the database.

default_provides = {'volume', 'volume_id', 'volume_properties'}

execute(*context: RequestContext*, *optional_args: dict*, ***kwargs*) → dict[str, Any]

Creates a database entry for the given inputs and returns details.

Accesses the database and creates a new entry for the to be created volume using the given volume properties which are extracted from the input kwargs (and associated requirements this task needs). These requirements should be previously satisfied and validated by a precursor task.

revert(*context: RequestContext*, *result: dict | Failure*, *optional_args: dict*, ***kwargs*) → None

Revert this atom.

This method should undo any side-effects caused by previous execution of the atom using the result of the *execute()* method and information on the failure which triggered reversion of the flow the atom is contained in (if applicable).

Parameters

- **args** positional arguments that the atom required to execute.
- **kwargs** any keyword arguments that the atom required to execute; the special key `'result'` will contain the *execute()* result (if any) and the `**kwargs` key `'flow_failures'` will contain any failure information.

class ExtractVolumeRequestTask(*image_service: GlanceImageService*, *availability_zones*, ***kwargs*)

Bases: *CinderTask*

Processes an api request values into a validated set of values.

This tasks responsibility is to take in a set of inputs that will form a potential volume request and validates those values against a set of conditions and/or translates those values into a valid set and then returns the validated/translated values for use by other tasks.

Reversion strategy: N/A

default_provides = {'availability_zones', 'backup_id', 'cgsnapshot_id', 'consistencygroup_id', 'encryption_key_id', 'group_id', 'multiattach', 'qos_specs', 'refresh_az', 'size', 'snapshot_id', 'source_volid', 'volume_type', 'volume_type_id'}

execute(*context: RequestContext*, *size: int*, *snapshot: dict | None*, *image_id: str | None*, *source_volume: dict | None*, *availability_zone: str | None*, *volume_type*, *metadata*, *key_manager*, *consistencygroup*, *cgsnapshot*, *group*, *group_snapshot*, *backup: dict | None*) → dict[str, Any]

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** positional arguments that atom requires to execute.
- **kwargs** any keyword arguments that atom requires to execute.

class QuotaCommitTask

Bases: *CinderTask*

Commits the reservation.

Reversion strategy: N/A (the rollback will be handled by the task that did the initial reservation (see: QuotaReserveTask).

Warning Warning: if the process that is running this reserve and commit process fails (or is killed before the quota is rolled back or committed it does appear like the quota will never be rolled back). This makes software upgrades hard (inflight operations will need to be stopped or allowed to complete before the upgrade can occur). *In the future* when taskflow has persistence built-in this should be easier to correct via an automated or manual process.

execute(*context: RequestContext*, *reservations*, *volume_properties*, *optional_args: dict*) → dict

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** positional arguments that atom requires to execute.
- **kwargs** any keyword arguments that atom requires to execute.

revert(*context: RequestContext*, *result: dict | Failure*, ***kwargs*) → None

Revert this atom.

This method should undo any side-effects caused by previous execution of the atom using the result of the *execute()* method and information on the failure which triggered reversion of the flow the atom is contained in (if applicable).

Parameters

- **args** positional arguments that the atom required to execute.
- **kwargs** any keyword arguments that the atom required to execute; the special key 'result' will contain the *execute()* result (if any) and the **kwargs key 'flow_failures' will contain any failure information.

class QuotaReserveTask

Bases: *CinderTask*

Reserves a single volume with the given size & the given volume type.

Reversion strategy: rollback the quota reservation.

Warning Warning: if the process that is running this reserve and commit process fails (or is killed before the quota is rolled back or committed it does appear like the quota will never be rolled back). This makes software upgrades hard (inflight operations will need to be stopped or allowed to complete before the upgrade can occur). *In the future* when taskflow has persistence built-in this should be easier to correct via an automated or manual process.

default_provides = {'reservations'}

execute(*context: RequestContext*, *size: int*, *volume_type_id*, *group_snapshot: Snapshot | None*, *optional_args: dict*) → dict | None

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via *args and **kwargs) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** positional arguments that atom requires to execute.
- **kwargs** any keyword arguments that atom requires to execute.

revert(*context: RequestContext*, *result: dict | Failure*, *optional_args: dict*, ***kwargs*) → None

Revert this atom.

This method should undo any side-effects caused by previous execution of the atom using the result of the *execute()* method and information on the failure which triggered reversion of the flow the atom is contained in (if applicable).

Parameters

- **args** positional arguments that the atom required to execute.
- **kwargs** any keyword arguments that the atom required to execute; the special key 'result' will contain the *execute()* result (if any) and the **kwargs key 'flow_failures' will contain any failure information.

class VolumeCastTask(*scheduler_rpcapi*, *volume_rpcapi*, *db*)

Bases: `CinderTask`

Performs a volume create cast to the scheduler or to the volume manager.

This will signal a transition of the api workflow to another child and/or related workflow on another component.

Reversion strategy: rollback source volume status and error out newly created volume.

execute(*context: RequestContext*, ***kwargs*) → None

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** positional arguments that atom requires to execute.
- **kwargs** any keyword arguments that atom requires to execute.

revert(*context: RequestContext*, *result: dict | Failure*, *flow_failures*, *volume: Volume*, ***kwargs*) → None

Revert this atom.

This method should undo any side-effects caused by previous execution of the atom using the result of the `execute()` method and information on the failure which triggered reversion of the flow the atom is contained in (if applicable).

Parameters

- **args** positional arguments that the atom required to execute.
- **kwargs** any keyword arguments that the atom required to execute; the special key `'result'` will contain the `execute()` result (if any) and the `**kwargs` key `'flow_failures'` will contain any failure information.

get_flow(*db_api*, *image_service_api*, *availability_zones*, *create_what*, *scheduler_rpcapi=None*, *volume_rpcapi=None*)

Constructs and returns the api entrypoint flow.

This flow will do the following:

1. Inject keys & values for dependent tasks.
2. Extracts and validates the input keys & values.
3. Reserves the quota (reverts quota on any failures).
4. Creates the database entry.
5. Commits the quota.

6. Casts to volume manager or scheduler for further processing.

cinder.volume.flows.api.manage_existing module

class EntryCreateTask(*db*)

Bases: `CinderTask`

Creates an entry for the given volume creation in the database.

Reversion strategy: remove the volume_id created from the database.

default_provides = {'volume', 'volume_properties'}

execute(*context*, ***kwargs*)

Creates a database entry for the given inputs and returns details.

Accesses the database and creates a new entry for the to be created volume using the given volume properties which are extracted from the input kwargs.

revert(*context*, *result*, *optional_args=None*, ***kwargs*)

Revert this atom.

This method should undo any side-effects caused by previous execution of the atom using the result of the `execute()` method and information on the failure which triggered reversion of the flow the atom is contained in (if applicable).

Parameters

- **args** positional arguments that the atom required to execute.
- **kwargs** any keyword arguments that the atom required to execute; the special key `'result'` will contain the `execute()` result (if any) and the `**kwargs` key `'flow_failures'` will contain any failure information.

class ManageCastTask(*scheduler_rpcapi*, *db*)

Bases: `CinderTask`

Performs a volume manage cast to the scheduler and the volume manager.

This which will signal a transition of the api workflow to another child and/or related workflow.

execute(*context*, *volume*, ***kwargs*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** positional arguments that atom requires to execute.
- **kwargs** any keyword arguments that atom requires to execute.

revert(*context*, *result*, *flow_failures*, *volume*, ***kwargs*)

Revert this atom.

This method should undo any side-effects caused by previous execution of the atom using the result of the `execute()` method and information on the failure which triggered reversion of the flow the atom is contained in (if applicable).

Parameters

- **args** positional arguments that the atom required to execute.
- **kwargs** any keyword arguments that the atom required to execute; the special key `'result'` will contain the `execute()` result (if any) and the `**kwargs` key `'flow_failures'` will contain any failure information.

get_flow(*scheduler_rpcapi*, *db_api*, *create_what*)

Constructs and returns the api entrypoint flow.

This flow will do the following:

1. Inject keys & values for dependent tasks.
2. Extracts and validates the input keys & values.
3. Creates the database entry.
4. Casts to volume manager and scheduler for further processing.

Module contents

cinder.volume.flows.manager package

Submodules

cinder.volume.flows.manager.create_volume module

class CreateVolumeFromSpecTask(*manager*, *db*, *driver*, *image_volume_cache=None*)

Bases: `CinderTask`

Creates a volume from a provided specification.

Reversion strategy: N/A

default_provides = 'volume_spec'

execute(*context: RequestContext*, *volume: Volume*, *volume_spec*) → dict

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** positional arguments that atom requires to execute.
- **kwargs** any keyword arguments that atom requires to execute.

property message

class CreateVolumeOnFinishTask(*db*, *event_suffix*)

Bases: *NotifyVolumeActionTask*

On successful volume creation this will perform final volume actions.

When a volume is created successfully it is expected that MQ notifications and database updates will occur to signal to others that the volume is now ready for usage. This task does those notifications and updates in a reliable manner (not re-raising exceptions if said actions can not be triggered).

Reversion strategy: N/A

execute(*context*, *volume*, *volume_spec*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** positional arguments that atom requires to execute.
- **kwargs** any keyword arguments that atom requires to execute.

class ExtractVolumeRefTask(*db*, *host*, *set_error=True*)

Bases: *CinderTask*

Extracts volume reference for given volume id.

default_provides = 'refreshed'

execute(*context*, *volume*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** positional arguments that atom requires to execute.

- **kwargs** any keyword arguments that atom requires to execute.

revert(*context*, *volume*, *result*, ***kwargs*)

Revert this atom.

This method should undo any side-effects caused by previous execution of the atom using the result of the *execute()* method and information on the failure which triggered reversion of the flow the atom is contained in (if applicable).

Parameters

- **args** positional arguments that the atom required to execute.
- **kwargs** any keyword arguments that the atom required to execute; the special key 'result' will contain the *execute()* result (if any) and the **kwargs key 'flow_failures' will contain any failure information.

class **ExtractVolumeSpecTask**(*db*)

Bases: *CinderTask*

Extracts a spec of a volume to be created into a common structure.

This task extracts and organizes the input requirements into a common and easier to analyze structure for later tasks to use. It will also attach the underlying database volume reference which can be used by other tasks to reference for further details about the volume to be.

Reversion strategy: N/A

default_provides = 'volume_spec'

execute(*context*, *volume*, *request_spec*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via *args and **kwargs) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** positional arguments that atom requires to execute.
- **kwargs** any keyword arguments that atom requires to execute.

revert(*context*, *result*, ***kwargs*)

Revert this atom.

This method should undo any side-effects caused by previous execution of the atom using the result of the *execute()* method and information on the failure which triggered reversion of the flow the atom is contained in (if applicable).

Parameters

- **args** positional arguments that the atom required to execute.

- **kwargs** any keyword arguments that the atom required to execute; the special key `'result'` will contain the *execute()* result (if any) and the `**kwargs` key `'flow_failures'` will contain any failure information.

class NotifyVolumeActionTask(*db*, *event_suffix*)

Bases: *CinderTask*

Performs a notification about the given volume when called.

Reversion strategy: N/A

execute(*context*, *volume*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** positional arguments that atom requires to execute.
- **kwargs** any keyword arguments that atom requires to execute.

class OnFailureRescheduleTask(*reschedule_context*, *db*, *manager*, *scheduler_rpcapi*, *do_reschedule*)

Bases: *CinderTask*

Triggers a rescheduling request to be sent when reverting occurs.

If rescheduling doesnt occur this task errors out the volume.

Reversion strategy: Triggers the rescheduling mechanism whereby a cast gets sent to the scheduler rpc api to allow for an attempt X of Y for scheduling this volume elsewhere.

execute(***kwargs*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** positional arguments that atom requires to execute.
- **kwargs** any keyword arguments that atom requires to execute.

`revert`(*context*, *result*, *flow_failures*, *volume*, ***kwargs*)

Revert this atom.

This method should undo any side-effects caused by previous execution of the atom using the result of the `execute()` method and information on the failure which triggered reversion of the flow the atom is contained in (if applicable).

Parameters

- **args** positional arguments that the atom required to execute.
- **kwargs** any keyword arguments that the atom required to execute; the special key `'result'` will contain the `execute()` result (if any) and the `**kwargs` key `'flow_failures'` will contain any failure information.

`get_flow`(*context*, *manager*, *db*, *driver*, *scheduler_rpcapi*, *host*, *volume*, *allow_reschedule*, *reschedule_context*, *request_spec*, *filter_properties*, *image_volume_cache=None*)

Constructs and returns the manager entrypoint flow.

This flow will do the following:

1. Determines if rescheduling is enabled (ahead of time).
2. Inject keys & values for dependent tasks.
3. Selects 1 of 2 activated only on *failure* tasks (one to update the db status & notify or one to update the db status & notify & *reschedule*).
4. Extracts a volume specification from the provided inputs.
5. Notifies that the volume has started to be created.
6. Creates a volume from the extracted volume specification.
7. Attaches an on-success *only* task that notifies that the volume creation has ended and performs further database status updates.

cinder.volume.flows.manager.manage_existing module

class `ManageExistingTask`(*db*, *driver*)

Bases: `CinderTask`

Brings an existing volume under Cinder management.

`default_provides = {'volume'}`

`execute`(*context*, *volume*, *manage_existing_ref*, *size*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** positional arguments that atom requires to execute.
- **kwargs** any keyword arguments that atom requires to execute.

class PrepareForQuotaReservationTask(*db*, *driver*)

Bases: *CinderTask*

Gets the volume size from the driver.

default_provides = {'size', 'volume_properties', 'volume_spec', 'volume_type_id'}

execute(*context*, *volume*, *manage_existing_ref*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** positional arguments that atom requires to execute.
- **kwargs** any keyword arguments that atom requires to execute.

revert(*context*, *result*, *flow_failures*, *volume*, ***kwargs*)

Revert this atom.

This method should undo any side-effects caused by previous execution of the atom using the result of the *execute()* method and information on the failure which triggered reversion of the flow the atom is contained in (if applicable).

Parameters

- **args** positional arguments that the atom required to execute.
- **kwargs** any keyword arguments that the atom required to execute; the special key `'result'` will contain the *execute()* result (if any) and the `**kwargs` key `'flow_failures'` will contain any failure information.

get_flow(*context*, *db*, *driver*, *host*, *volume*, *ref*)

Constructs and returns the manager entrypoint flow.

cinder.volume.flows.manager.manage_existing_snapshot module

class CreateSnapshotOnFinishTask(*db*, *event_suffix*, *host*)

Bases: *NotifySnapshotActionTask*

Perform final snapshot actions.

When a snapshot is created successfully it is expected that MQ notifications and database updates will occur to signal to others that the snapshot is now ready for usage. This task does those notifications and updates in a reliable manner (not re-raising exceptions if said actions can not be triggered).

Reversion strategy: N/A

execute(*context*, *snapshot*, *new_status*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** positional arguments that atom requires to execute.
- **kwargs** any keyword arguments that atom requires to execute.

class **ExtractSnapshotRefTask**(*db*)

Bases: *CinderTask*

Extracts snapshot reference for given snapshot id.

default_provides = 'snapshot_ref'

execute(*context*, *snapshot_id*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** positional arguments that atom requires to execute.
- **kwargs** any keyword arguments that atom requires to execute.

revert(*context*, *snapshot_id*, *result*, ***kwargs*)

Revert this atom.

This method should undo any side-effects caused by previous execution of the atom using the result of the *execute()* method and information on the failure which triggered reversion of the flow the atom is contained in (if applicable).

Parameters

- **args** positional arguments that the atom required to execute.

- **kwargs** any keyword arguments that the atom required to execute; the special key `'result'` will contain the *execute()* result (if any) and the `**kwargs` key `'flow_failures'` will contain any failure information.

class ManageExistingTask(*db*, *driver*)

Bases: *CinderTask*

Brings an existing snapshot under Cinder management.

default_provides = {'new_status', 'snapshot'}

execute(*context*, *snapshot_ref*, *manage_existing_ref*, *size*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** positional arguments that atom requires to execute.
- **kwargs** any keyword arguments that atom requires to execute.

class NotifySnapshotActionTask(*db*, *event_suffix*, *host*)

Bases: *CinderTask*

Performs a notification about the given snapshot when called.

Reversion strategy: N/A

execute(*context*, *snapshot_ref*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** positional arguments that atom requires to execute.
- **kwargs** any keyword arguments that atom requires to execute.

class PrepareForQuotaReservationTask(*db*, *driver*)

Bases: *CinderTask*

Gets the snapshot size from the driver.

default_provides = {'size', 'snapshot_properties'}

execute(*context*, *snapshot_ref*, *manage_existing_ref*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** positional arguments that atom requires to execute.
- **kwargs** any keyword arguments that atom requires to execute.

class QuotaCommitTask

Bases: *CinderTask*

Commits the reservation.

Reversion strategy: N/A (the rollback will be handled by the task that did the initial reservation (see: QuotaReserveTask).

Warning Warning: if the process that is running this reserve and commit process fails (or is killed before the quota is rolled back or committed it does appear like the quota will never be rolled back). This makes software upgrades hard (inflight operations will need to be stopped or allowed to complete before the upgrade can occur). *In the future* when taskflow has persistence built-in this should be easier to correct via an automated or manual process.

execute(*context*, *reservations*, *snapshot_properties*, *optional_args*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** positional arguments that atom requires to execute.
- **kwargs** any keyword arguments that atom requires to execute.

revert(*context*, *result*, ***kwargs*)

Revert this atom.

This method should undo any side-effects caused by previous execution of the atom using the result of the *execute()* method and information on the failure which triggered reversion of the flow the atom is contained in (if applicable).

Parameters

- **args** positional arguments that the atom required to execute.
- **kwargs** any keyword arguments that the atom required to execute; the special key `'result'` will contain the *execute()* result (if any) and the `**kwargs` key `'flow_failures'` will contain any failure information.

class QuotaReserveTask

Bases: *CinderTask*

Reserves a single snapshot with the given size.

Reversion strategy: rollback the quota reservation.

Warning Warning: if the process that is running this reserve and commit process fails (or is killed before the quota is rolled back or committed it does appear like the quota will never be rolled back). This makes software upgrades hard (inflight operations will need to be stopped or allowed to complete before the upgrade can occur). *In the future* when taskflow has persistence built-in this should be easier to correct via an automated or manual process.

default_provides = {'reservations'}

execute(*context*, *size*, *snapshot_ref*, *optional_args*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** positional arguments that atom requires to execute.
- **kwargs** any keyword arguments that atom requires to execute.

revert(*context*, *result*, *optional_args*, ***kwargs*)

Revert this atom.

This method should undo any side-effects caused by previous execution of the atom using the result of the *execute()* method and information on the failure which triggered reversion of the flow the atom is contained in (if applicable).

Parameters

- **args** positional arguments that the atom required to execute.
- **kwargs** any keyword arguments that the atom required to execute; the special key `'result'` will contain the *execute()* result (if any) and the `**kwargs` key `'flow_failures'` will contain any failure information.

get_flow(*context*, *db*, *driver*, *host*, *snapshot_id*, *ref*)
Constructs and returns the manager entry point flow.

Module contents

Submodules

cinder.volume.flows.common module

error_out(*resource*, *reason=None*, *status='error'*)
Sets status to error for any persistent OVO.

make_pretty_name(*method: Callable*) → str
Makes a pretty name for a function/method.

restore_source_status(*context*, *db*, *volume_spec*)

Module contents

cinder.volume.targets package

Submodules

cinder.volume.targets.cxt module

class CxtAdm(**args*, ***kwargs*)
Bases: *ISCSITarget*

Chiscsi target configuration for block storage devices.

This includes things like create targets, attach, detach etc.

TARGET_FMT = '\n target:\n TargetName=%s\n TargetDevice=%s\n PortalGroup=1@%s\n '

TARGET_FMT_WITH_CHAP = '\n target:\n TargetName=%s\n TargetDevice=%s\n PortalGroup=1@%s\n AuthMethod=CHAP\n Auth_CHAP_Policy=Oneway\n Auth_CHAP_Initiator=%s\n '

create_iscsi_target(*name*, *tid*, *lun*, *path*, *chap_auth=None*, ***kwargs*)

cxt_subdir = 'cxt'

remove_iscsi_target(*tid*, *lun*, *vol_id*, *vol_name*, ***kwargs*)

cinder.volume.targets.driver module

class Target(**args*, ***kwargs*)
Bases: object

Target object for block storage devices.

Base class for target object, where target is data transport mechanism (target) specific calls. This includes things like create targets, attach, detach etc.

Base class here does nothing more than set an executor and db as well as force implementation of required methods.

SECONDARY_IP_SUPPORT = True

SHARED_TARGET_SUPPORT = False

static **are_same_connector**(*A*, *B*)

Whether 2 connectors belong to the same host or not.

This is used for multi attach volumes, to be able to know when there are no more attachments on a given host.

This is the generic implementation, but specific targets may overwrite it. For example iSCSI would check the the initiator key instead, and NVMe-oF would check the nqn key.

abstract **create_export**(*context*, *volume*, *volume_path*)

Exports a Target/Volume.

Can optionally return a Dict of changes to the volume object to be persisted.

abstract **ensure_export**(*context*, *volume*, *volume_path*)

Synchronously recreates an export for a volume.

extend_target(*volume*)

Reinitializes a target after the volume has been extended.

Most drivers dont need to do anything, but in other cases this may cause IO disruption.

abstract **initialize_connection**(*volume*, *connector*)

Allow connection to connector and return connection info.

abstract **remove_export**(*context*, *volume*)

Removes an export for a Target/Volume.

storage_protocol = None

abstract **terminate_connection**(*volume*, *connector*, ***kwargs*)

Disallow connection from connector.

cinder.volume.targets.fake module

class **FakeTarget**(**args*, ***kwargs*)

Bases: *ISCSITarget*

VERSION = '0.1'

create_iscsi_target(*name*, *tid*, *lun*, *path*, *chap_auth*, ***kwargs*)

remove_iscsi_target(*tid*, *lun*, *vol_id*, *vol_name*, ***kwargs*)

cinder.volume.targets.iscsi module

class **ISCSITarget**(**args*, ***kwargs*)

Bases: *Target*

Target object for block storage devices.

Base class for target object, where target is data transport mechanism (target) specific calls. This includes things like create targets, attach, detach etc.

static are_same_connector(*A*, *B*)

Whether 2 connectors belong to the same host or not.

This is used for multi attach volumes, to be able to know when there are no more attachments on a given host.

This is the generic implementation, but specific targets may overwrite it. For example iSCSI would check the the initiator key instead, and NVMe-oF would check the nqn key.

create_export(*context*, *volume*, *volume_path*)

Creates an export for a logical volume.

abstract create_iscsi_target(*name*, *tid*, *lun*, *path*, *chap_auth*, ***kwargs*)

ensure_export(*context*, *volume*, *volume_path*)

Recreates an export for a logical volume.

extend_target(*volume*)

Reinitializes a target after the LV has been extended.

Note: This will cause IO disruption in most cases.

initialize_connection(*volume*, *connector*)

Initializes the connection and returns connection info.

The iscsi driver returns a driver_volume_type of iscsi. The format of the driver data is defined in _get_iscsi_properties. Example return value:

```
{
    'driver_volume_type': 'iscsi'
    'data': {
        'target_discovered': True,
        'target_iqn': 'iqn.2010-10.org.openstack:volume-00000001',
        'target_portal': '127.0.0.0.1:3260',
        'volume_id': '9a0d35d0-175a-11e4-8c21-0800200c9a66',
        'discard': False,
    }
}
```

remove_export(*context*, *volume*)

Removes an export for a Target/Volume.

abstract remove_iscsi_target(*tid*, *lun*, *vol_id*, *vol_name*, ***kwargs*)

show_target(*iscsi_target*, *iqn*, ***kwargs*)

terminate_connection(*volume*, *connector*, ***kwargs*)

Disallow connection from connector.

validate_connector(*connector*)

class SanISCSITarget(**args*, ***kwargs*)

Bases: *ISCSITarget*

iSCSI target for san devices.

San devices are slightly different, they dont need to implement all of the same things that we need to implement locally fro LVM and local block devices when we create and manage our own targets.

abstract create_export(*context*, *volume*, *volume_path*)

Creates an export for a logical volume.

create_iscsi_target(*name*, *tid*, *lun*, *path*, *chap_auth*, ***kwargs*)

abstract ensure_export(*context*, *volume*, *volume_path*)

Recreates an export for a logical volume.

abstract remove_export(*context*, *volume*)

Removes an export for a Target/Volume.

remove_iscsi_target(*tid*, *lun*, *vol_id*, *vol_name*, ***kwargs*)

abstract terminate_connection(*volume*, *connector*, ***kwargs*)

Disallow connection from connector.

cinder.volume.targets.lio module

class LioAdm(**args*, ***kwargs*)

Bases: *ISCSITarget*

iSCSI target administration for LIO using python-rtslib.

create_iscsi_target(*name*, *tid*, *lun*, *path*, *chap_auth=None*, ***kwargs*)

ensure_export(*context*, *volume*, *volume_path*)

Recreate exports for logical volumes.

initialize_connection(*volume*, *connector*)

Initializes the connection and returns connection info.

The iscsi driver returns a driver_volume_type of iscsi. The format of the driver data is defined in _get_iscsi_properties. Example return value:

```
{
    'driver_volume_type': 'iscsi'
    'data': {
        'target_discovered': True,
        'target_iqn': 'iqn.2010-10.org.openstack:volume-00000001',
        'target_portal': '127.0.0.0.1:3260',
        'volume_id': '9a0d35d0-175a-11e4-8c21-0800200c9a66',
        'discard': False,
    }
}
```

remove_iscsi_target(*tid*, *lun*, *vol_id*, *vol_name*, ***kwargs*)

terminate_connection(*volume*, *connector*, ***kwargs*)

Disallow connection from connector.

cinder.volume.targets.nvmeof module

class NVMeOF(**args*, ***kwargs*)

Bases: `Target`

Target object for block storage devices with RDMA transport.

static are_same_connector(*A*, *B*)

Whether 2 connectors belong to the same host or not.

This is used for multi attach volumes, to be able to know when there are no more attachments on a given host.

This is the generic implementation, but specific targets may overwrite it. For example iSCSI would check the the initiator key instead, and NVMe-oF would check the nqn key.

create_export(*context*, *volume*, *volume_path*)

Creates export data for a logical volume.

create_nvmeof_target(*volume_id*, *subsystem_name*, *target_ips*, *target_port*, *transport_type*, *nvmet_port_id*, *ns_id*, *volume_path*)

Targets that dont override create_export must implement this.

delete_nvmeof_target(*target_name*)

Targets that dont override remove_export must implement this.

ensure_export(*context*, *volume*, *volume_path*)

Synchronously recreates an export for a volume.

get_nvmeof_location(*nqn*, *target_ips*, *target_port*, *nvme_transport_type*, *nvmet_ns_id*)

Serializes driver data into single line string.

initialize_connection(*volume*, *connector*)

Returns the connection info.

In NVMeOF driver, :driver_volume_type: is set to nvmeof, :data: is the driver data that has the value of _get_connection_properties_from_vol.

Example return value:

```
{
    "driver_volume_type": "nvmeof",
    "data":
    {
        "target_portal": "1.1.1.1",
        "target_port": 4420,
        "nqn": "nqn.volume-0001",
        "transport_type": "rdma",
        "ns_id": 10
    }
}
```

protocol = 'nvmeof'

remove_export(*context*, *volume*)

Removes an export for a Target/Volume.

target_protocol_map = {'nvmet_rdma': 'rdma', 'nvmet_tcp': 'tcp'}

terminate_connection(*volume*, *connector*, ***kwargs*)
Disallow connection from connector.

validate_connector(*connector*)

exception UnsupportedNVMETProtocol(*message: str | tuple | None = None*, ***kwargs*)
Bases: *Invalid*

message = "An invalid 'target_protocol' value was provided: %(protocol)s"

cinder.volume.targets.nvmet module

class NVMET(**args*, ***kwargs*)
Bases: *NVMeOF*

SHARED_TARGET_SUPPORT = True

create_export(*context*, *volume*, *volume_path*)
Create an export & map if not shared.

initialize_connection(*volume*, *connector*)
Create an export & map if shared.

remove_export(*context*, *volume*)
Remove the mapping for non shared.

terminate_connection(*volume*, *connector*, ***kwargs*)
Remove the mapping for shared.

exception NVMETTargetAddError(*message: str | tuple | None = None*, ***kwargs*)
Bases: *CinderException*

message = 'Failed to add subsystem: %(subsystem)s'

exception NVMETTargetDeleteError(*message: str | tuple | None = None*, ***kwargs*)
Bases: *CinderException*

message = 'Failed to delete subsystem: %(subsystem)s'

cinder.volume.targets.scst module

class SCSTAdm(**args*, ***kwargs*)
Bases: *ISCSITarget*

create_export(*context*, *volume*, *volume_path*)
Creates an export for a logical volume.

create_iscsi_target(*name*, *vol_id*, *tid*, *lun*, *path*, *chap_auth=None*)

ensure_export(*context*, *volume*, *volume_path*)
Recreates an export for a logical volume.

remove_export(*context*, *volume*)
Removes an export for a Target/Volume.

remove_iscsi_target(*tid*, *lun*, *vol_id*, *vol_name*, ***kwargs*)

scst_execute(**args*)

show_target(*tid*, *iqn*)

terminate_connection(*volume*, *connector*, ***kwargs*)

Disallow connection from connector.

validate_connector(*connector*)

cinder.volume.targets.spdknvmf module

class **SpdkNvmf**(**args*, ***kwargs*)

Bases: *NVMeOF*

SECONDARY_IP_SUPPORT = False

create_nvmeof_target(*volume_id*, *subsystem_name*, *target_ips*, *target_port*, *transport_type*, *nvmet_port_id*, *ns_id*, *volume_path*)

Targets that dont override create_export must implement this.

delete_nvmeof_target(*target_name*)

Targets that dont override remove_export must implement this.

cinder.volume.targets.tgt module

class **TgtAdm**(**args*, ***kwargs*)

Bases: *ISCSITarget*

Target object for block storage devices.

Base class for target object, where target is data transport mechanism (target) specific calls. This includes things like create targets, attach, detach etc.

VOLUME_CONF = '\n<target %(name)s>\n backing-store %(path)s\n driver %(driver)s\n %(chap_auth)s\n %(target_flags)s\n write-cache %(write_cache)s\n</target>\n'

create_iscsi_target(*name*, *tid*, *lun*, *path*, *chap_auth=None*, ***kwargs*)

remove_iscsi_target(*tid*, *lun*, *vol_id*, *vol_name*, ***kwargs*)

Module contents

Submodules

cinder.volume.api module

Handles all requests relating to volumes.

class **API**(*image_service=None*)

Bases: *Base*

API for interacting with the volume manager.

AVAILABLE_MIGRATION_STATUS = (None, 'deleting', 'error', 'success')

accept_transfer(*context: RequestContext*, *volume: Volume*, *new_user: str*, *new_project: str*, *no_snapshots: bool = False*) → dict

attach(*context: RequestContext*, *volume: Volume*, *instance_uuid: str*, *host_name: str*, *mountpoint: str*, *mode: str*) → *VolumeAttachment*

attachment_create(*ctxt: RequestContext*, *volume_ref: Volume*, *instance_uuid: str*, *connector: dict | None = None*, *attach_mode: str | None = 'null'*) → *VolumeAttachment*

Create an attachment record for the specified volume.

attachment_delete(*ctxt: RequestContext*, *attachment*) → *VolumeAttachmentList*

attachment_deletion_allowed(*ctxt: RequestContext*, *attachment_or_attachment_id*, *volume=None*)

Check if deleting an attachment is allowed (Bug #2004555)

Allowed is based on the REST API policy, the status of the attachment, where it is used, and who is making the request.

Deleting an attachment on the Cinder side while leaving the volume connected to the nova host results in leftover devices that can lead to data leaks/corruption.

OS-Brick may have code to detect it, but in some cases it is detected after it has already been exposed, so its better to prevent users from being able to intentionally triggering the issue.

attachment_update(*ctxt: RequestContext*, *attachment_ref: VolumeAttachment*, *connector*) → *VolumeAttachment*

Update an existing attachment record.

begin_detaching(*context: RequestContext*, *volume: Volume*) → None

calculate_resource_count(*context: RequestContext*, *resource_type: str*, *filters: dict | None*) → int

check_volume_filters(*filters: dict*, *strict: bool = False*) → None

Sets the user filter value to accepted format

copy_volume_to_image(*context: RequestContext*, *volume: Volume*, *metadata: dict[str, str]*, *force: bool*) → dict[str, str | None]

Create a new image from the specified volume.

create(*context: RequestContext*, *size: str | int*, *name: str | None*, *description: str | None*, *snapshot: Snapshot | None = None*, *image_id: str | None = None*, *volume_type: VolumeType | None = None*, *metadata: dict | None = None*, *availability_zone: str | None = None*, *source_volume: Volume | None = None*, *scheduler_hints=None*, *source_replica=None*, *consistencygroup: ConsistencyGroup | None = None*, *cgsnapshot: CGSnapshot | None = None*, *source_cg=None*, *group: Group | None = None*, *group_snapshot=None*, *source_group=None*, *backup: Backup | None = None*)

create_snapshot(*context: RequestContext*, *volume: Volume*, *name: str*, *description: str*, *metadata: dict[str, Any] | None = None*, *cgsnapshot_id: str | None = None*, *group_snapshot_id: str | None = None*, *allow_in_use: bool = False*) → *Snapshot*

create_snapshot_force(*context: RequestContext*, *volume: Volume*, *name: str*, *description: str*, *metadata: dict[str, Any] | None = None*) → *Snapshot*

create_snapshot_in_db(*context: RequestContext*, *volume: Volume*, *name: str | None*, *description: str | None*, *force: bool*, *metadata: dict | None*, *cgsnapshot_id: str | None*, *commit_quota: bool = True*, *group_snapshot_id: str | None = None*, *allow_in_use: bool = False*) → *Snapshot*

create_snapshots_in_db(*context: RequestContext*, *volume_list: list*, *name: str*, *description: str*, *cgsnapshot_id: str*, *group_snapshot_id: str | None = None*) → list

create_volume_metadata(*context: RequestContext*, *volume: Volume*, *metadata: dict[str, Any]*) → dict

Creates volume metadata.

delete(*context: RequestContext*, *volume: Volume*, *force: bool = False*, *unmanage_only: bool = False*, *cascade: bool = False*) → None

delete_snapshot(*context: RequestContext*, *snapshot: Snapshot*, *force: bool = False*, *unmanage_only: bool = False*) → None

delete_snapshot_metadata(*context: RequestContext*, *snapshot: Snapshot*, *key: str*) → None

Delete the given metadata item from a snapshot.

delete_volume_metadata(*context: RequestContext*, *volume: Volume*, *key: str*, *meta_type=METADATA_TYPES.user*) → None

Delete the given metadata item from a volume.

detach(*context: RequestContext*, *volume: Volume*, *attachment_id: str*) → None

extend(*context: RequestContext*, *volume: Volume*, *new_size: int*) → None

extend_attached_volume(*context: RequestContext*, *volume: Volume*, *new_size: int*) → None

extend_volume_completion(*context: RequestContext*, *volume: Volume*, *error: bool*)

failover(*ctxt: RequestContext*, *host: str*, *cluster_name: str*, *secondary_id: str | None = None*) → None

freeze_host(*ctxt: RequestContext*, *host: str*, *cluster_name: str*) → None

get(*context: RequestContext*, *volume_id: str*, *viewable_admin_meta: bool = False*) → *Volume*

get_all(*context: RequestContext*, *marker: str | None = None*, *limit: int | None = None*, *sort_keys: Iterable[str] | None = None*, *sort_dirs: Iterable[str] | None = None*, *filters: dict | None = None*, *viewable_admin_meta: bool = False*, *offset: int | None = None*) → *VolumeList*

get_all_snapshots(*context: RequestContext*, *search_opts: dict | None = None*, *marker: str | None = None*, *limit: int | None = None*, *sort_keys: list[str] | None = None*, *sort_dirs: list[str] | None = None*, *offset: int | None = None*) → *SnapshotList*

get_list_volumes_image_metadata(*context: RequestContext*, *volume_id_list: list[str]*) → DefaultDict[str, str]

get_manageable_snapshots(*context: RequestContext*, *host: str*, *cluster_name: str | None*, *marker: str | None = None*, *limit: int | None = None*, *offset: int | None = None*, *sort_keys: list[str] | None = None*, *sort_dirs: list[str] | None = None*) → list[dict]

get_manageable_volumes(*context: RequestContext*, *host: str*, *cluster_name*, *marker: str | None = None*, *limit: int | None = None*, *offset: int | None = None*, *sort_keys: list[str] | None = None*, *sort_dirs: list[str] | None = None*)

get_snapshot(*context: RequestContext*, *snapshot_id: str*) → *Snapshot*

get_snapshot_metadata(*context: RequestContext*, *snapshot: Snapshot*) → dict

Get all metadata associated with a snapshot.

get_volume(*context: RequestContext*, *volume_id: str*) → *Volume*

get_volume_image_metadata(*context: RequestContext*, *volume: Volume*) → dict[str, str]

get_volume_metadata(*context: RequestContext*, *volume: Volume*) → dict

Get all metadata associated with a volume.

get_volume_summary(*context: RequestContext*, *filters: dict | None = None*) → *VolumeList*

get_volumes_image_metadata(*context: RequestContext*) → defaultdict

initialize_connection(*context: RequestContext*, *volume: Volume*, *connector: dict*) → dict

static is_service_request(*ctxt: RequestContext*) → bool

Check if a request is coming from a service

A request is coming from a service if it has a service token and the service user has one of the roles configured in the *service_token_roles* configuration option in the *[keystone_authtoken]* section (defaults to *service*).

list_availability_zones(*enable_cache: bool = False*, *refresh_cache: bool = False*) → tuple

Describe the known availability zones

Parameters

- **enable_cache** Enable az cache
- **refresh_cache** Refresh cache immediately

Returns

tuple of dicts, each with a name and available key

manage_existing(*context: RequestContext, host: str, cluster_name: str | None, ref: dict, name: str | None = None, description: str | None = None, volume_type: VolumeType | None = None, metadata: dict | None = None, availability_zone: str | None = None, bootable: bool | None = False*) → *Volume*

manage_existing_snapshot(*context: RequestContext, ref: dict, volume: Volume, name: str | None = None, description: str | None = None, metadata: dict | None = None*) → *Snapshot*

migrate_volume(*context: RequestContext, volume: Volume, host: str, cluster_name: str, force_copy: bool, lock_volume: bool*) → None

Migrate the volume to the specified host or cluster.

migrate_volume_completion(*context: RequestContext, volume: Volume, new_volume: Volume, error: bool*) → str

reimage(*context, volume, image_id, reimage_reserved=False*)

reserve_volume(*context: RequestContext, volume: Volume*) → None

retype(*context: RequestContext, volume: Volume, new_type: str | VolumeType, migration_policy: str | None = None*) → None

Attempt to modify the type associated with an existing volume.

revert_to_snapshot(*context: RequestContext, volume: Volume, snapshot: Snapshot*) → None

revert a volume to a snapshot

roll_detaching(*context: RequestContext, volume: Volume*) → None

terminate_connection(*context: RequestContext, volume: Volume, connector: dict, force: bool = False*) → None

thaw_host(*ctxt: RequestContext, host: str, cluster_name: str*) → str | None

unreserve_volume(*context: RequestContext, volume: Volume*) → None

update(*context: RequestContext, volume: Volume, fields: dict*) → None

update_readonly_flag(*context: RequestContext, volume: Volume, flag*) → None

update_snapshot(*context: RequestContext, snapshot: Snapshot, fields: dict[str, Any]*) → None

update_snapshot_metadata(*context: RequestContext, snapshot: Snapshot, metadata: dict[str, Any], delete: bool = False*) → dict

Updates or creates snapshot metadata.

If delete is True, metadata items that are not specified in the *metadata* argument will be deleted.

update_volume_admin_metadata(*context: RequestContext, volume: Volume, metadata: dict[str, Any], delete: bool | None = False, add: bool | None = True, update: bool | None = True*) → dict

Updates or creates volume administration metadata.

If delete is True, metadata items that are not specified in the *metadata* argument will be deleted.

update_volume_metadata(*context: RequestContext*, *volume: Volume*, *metadata: dict[str, Any]*, *delete: bool = False*, *meta_type=METADATA_TYPES.user*) → dict

Updates volume metadata.

If delete is True, metadata items that are not specified in the *metadata* argument will be deleted.

class HostAPI

Bases: `Base`

Sub-set of the Volume Manager API for managing host operations.

set_host_enabled(*context*, *host*, *enabled*)

Sets the specified hosts ability to accept new volumes.

cinder.volume.configuration module

Configuration support for all drivers.

This module allows support for setting configurations either from default or from a particular FLAGS group, to be able to set multiple configurations for a given set of values.

For instance, two lvm configurations can be set by naming them in groups as

[lvm1] volume_group=lvm-group-1

[lvm2] volume_group=lvm-group-2

And the configuration group name will be passed in so that all calls to configuration.volume_group within that instance will be mapped to the proper named group.

This class also ensures the implementations configuration is grafted into the option group. This is due to the way cfg works. All cfg options must be defined and registered in the group in which they are used.

class BackendGroupConfiguration(*volume_opts*, *config_group=None*)

Bases: `object`

append_config_values(*volume_opts*)

get(*key*, *default=None*)

safe_get(*value*)

set_default(*opt_name*, *default*)

class Configuration(*volume_opts*, *config_group=None*)

Bases: `object`

append_config_values(*volume_opts*)

safe_get(*value*)

class DefaultGroupConfiguration

Bases: object

Get config options from only DEFAULT.

append_config_values(*volume_opts*)

safe_get(*value*)

cinder.volume.driver module

Drivers for volumes.

class BaseVD(*execute=<function execute>*, **args*, ***kwargs*)

Bases: object

Executes commands relating to Volumes.

Base Driver for Cinder Volume Control Path, This includes supported/required implementation for API calls. Also provides *generic* implementation of core features like cloning, copy_image_to_volume etc, this way drivers that inherit from this base class and dont offer their own impl can fall back on a general solution here.

Key thing to keep in mind with this driver is that its intended that these drivers ONLY implement Control Path details (create, delete, extend), while transport or data path related implementation should be a *member object* that we call a connector. The point here is that for example dont allow the LVM driver to implement iSCSI methods, instead call whatever connector it has configured via conf file (iSCSI{LIO, TGT, ET}, FC, etc).

In the base class and for example the LVM driver we do this via a has-a relationship and just provide an interface to the specific connector methods. How you do this in your own driver is of course up to you.

REPLICATION_FEATURE_CHECKERS = {'a/a': 'failover_completed', 'v2.1': 'failover_host'}

SUPPORTED = True

SUPPORTS_ACTIVE_ACTIVE = False

VERSION = 'N/A'

accept_transfer(*context*, *volume*, *new_user*, *new_project*)

after_volume_copy(*context*, *src_vol*, *dest_vol*, *remote=None*)

Driver-specific actions executed after copying a volume.

Refer to cinder.interface.volume_driver.VolumeDriverCore.after_volume_copy for additional information.

backup_use_temp_snapshot()

Get the configured setting for backup from snapshot.

If an inheriting driver does not support this operation, the driver should override this method to return false and log a warning letting the administrator know they have configured something that cannot be done.

before_volume_copy(*context*, *src_vol*, *dest_vol*, *remote=None*)
Driver-specific actions executed before copying a volume.

Refer to *cinder.interface.volume_driver.VolumeDriverCore.before_volume_copy* for additional information.

abstract check_for_setup_error()

classmethod clean_snapshot_file_locks(*snapshot_id*)
Clean up driver specific snapshot locks.

This method will be called when a snapshot has been removed from cinder or when we detect that the snapshot doesnt exist.

There are 3 types of locks in Cinder:

- Process locks: Dont need cleanup
- Node locks: Must use cinder.utils.synchronized_remove
- Global locks: Must use cinder.coordination.synchronized_remove

When using method cinder.utils.synchronized_remove we must pass the exact lock name, whereas method cinder.coordination.synchronized_remove accepts a glob.

Refer to clean_volume_file_locks, api_clean_volume_file_locks, and clean_snapshot_file_locks in cinder.utils for examples.

classmethod clean_volume_file_locks(*volume_id*)
Clean up driver specific volume locks.

This method will be called when a volume has been removed from Cinder or when we detect that the volume doesnt exist.

There are 3 types of locks in Cinder:

- Process locks: Dont need cleanup
- Node locks: Must use cinder.utils.synchronized_remove
- Global locks: Must use cinder.coordination.synchronized_remove

When using method cinder.utils.synchronized_remove we must pass the exact lock name, whereas method cinder.coordination.synchronized_remove accepts a glob.

Refer to clean_volume_file_locks, api_clean_volume_file_locks, and clean_snapshot_file_locks in cinder.utils for examples.

clear_download(*context*, *volume*)
Clean up after an interrupted image copy.

clone_image(*context*, *volume*, *image_location*, *image_meta*, *image_service*)
Create a volume efficiently from an existing image.

Refer to *cinder.interface.volume_driver.VolumeDriverCore.clone_image* for additional information.

copy_image_to_encrypted_volume(*context*, *volume*, *image_service*, *image_id*, *disable_sparse=False*)
Fetch image from image_service and write to encrypted volume.

This attaches the encryptor layer when connecting to the volume.

`copy_image_to_volume`(*context*, *volume*, *image_service*, *image_id*, *disable_sparse=False*)

Fetch image from image_service and write to unencrypted volume.

This does not attach an encryptor layer when connecting to the volume.

`copy_volume_to_image`(*context*, *volume*, *image_service*, *image_meta*)

Copy the volume to the specified image.

`create_cloned_volume`(*volume*, *src_vref*)

Creates a clone of the specified volume.

If volume_type extra specs includes replication: <is> True the driver needs to create a volume replica (secondary) and setup replication between the newly created volume and the secondary volume.

`abstract create_export`(*context*, *volume*, *connector*)

Exports the volume.

Can optionally return a Dictionary of changes to the volume object to be persisted.

`create_export_snapshot`(*context*, *snapshot*, *connector*)

Exports the snapshot.

Can optionally return a Dictionary of changes to the snapshot object to be persisted.

`create_group`(*context*, *group*)

Creates a group.

Parameters

- `context` the context of the caller.
- `group` the Group object of the group to be created.

Returns

model_update

model_update will be in this format: {status: xxx, }.

If the status in model_update is error, the manager will throw an exception and it will be caught in the try-except block in the manager. If the driver throws an exception, the manager will also catch it in the try-except block. The group status in the db will be changed to error.

For a successful operation, the driver can either build the model_update and return it or return None. The group status will be set to available.

`create_group_from_src`(*context*, *group*, *volumes*, *group_snapshot=None*, *snapshots=None*, *source_group=None*, *source_vols=None*)

Creates a group from source.

Parameters

- `context` the context of the caller.
- `group` the Group object to be created.
- `volumes` a list of Volume objects in the group.
- `group_snapshot` the GroupSnapshot object as source.
- `snapshots` a list of Snapshot objects in group_snapshot.

- `source_group` the Group object as source.
- `source_vols` a list of Volume objects in the source_group.

Returns
model_update, volumes_model_update

The source can be group_snapshot or a source_group.

param volumes is a list of objects retrieved from the db. It cannot be assigned to volumes_model_update. volumes_model_update is a list of dictionaries. It has to be built by the driver. An entry will be in this format: {id: xxx, status: xxx, }. model_update will be in this format: {status: xxx, }.

To be consistent with other volume operations, the manager will assume the operation is successful if no exception is thrown by the driver. For a successful operation, the driver can either build the model_update and volumes_model_update and return them or return None, None.

`create_group_snapshot`(*context*, *group_snapshot*, *snapshots*)

Creates a group_snapshot.

Parameters

- `context` the context of the caller.
- `group_snapshot` the GroupSnapshot object to be created.
- `snapshots` a list of Snapshot objects in the group_snapshot.

Returns
model_update, snapshots_model_update

param snapshots is a list of Snapshot objects. It cannot be assigned to snapshots_model_update. snapshots_model_update is a list of dictionaries. It has to be built by the driver. An entry will be in this format: {id: xxx, status: xxx, }. model_update will be in this format: {status: xxx, }.

The driver should populate snapshots_model_update and model_update and return them.

The manager will check snapshots_model_update and update db accordingly for each snapshot. If the driver successfully deleted some snapshots but failed to delete others, it should set statuses of the snapshots accordingly so that the manager can update db correctly.

If the status in any entry of snapshots_model_update is error, the status in model_update will be set to the same if it is not already error.

If the status in model_update is error, the manager will raise an exception and the status of group_snapshot will be set to error in the db. If snapshots_model_update is not returned by the driver, the manager will set the status of every snapshot to error in the except block.

If the driver raises an exception during the operation, it will be caught by the try-except block in the manager and the statuses of group_snapshot and all snapshots will be set to error.

For a successful operation, the driver can either build the model_update and snapshots_model_update and return them or return None, None. The statuses of group_snapshot and all snapshots will be set to available at the end of the manager function.

abstract create_volume(*volume*)

Creates a volume.

Can optionally return a Dictionary of changes to the volume object to be persisted.

If volume_type extra specs includes capabilities:replication <is> True the driver needs to create a volume replica (secondary), and setup replication between the newly created volume and the secondary volume. Returned dictionary should include:

```
volume['replication_status'] = 'copying'
volume['replication_extended_status'] = <driver specific value>
volume['driver_data'] = <driver specific value>
```

create_volume_from_backup(*volume*, *backup*)

Creates a volume from a backup.

Can optionally return a Dictionary of changes to the volume object to be persisted.

Parameters

- **volume** the volume object to be created.
- **backup** the backup object as source.

Returns

volume_model_update

delete_group(*context*, *group*, *volumes*)

Deletes a group.

Parameters

- **context** the context of the caller.
- **group** the Group object of the group to be deleted.
- **volumes** a list of Volume objects in the group.

Returns

model_update, volumes_model_update

param volumes is a list of objects retrieved from the db. It cannot be assigned to volumes_model_update. volumes_model_update is a list of dictionaries. It has to be built by the driver. An entry will be in this format: {id: xxx, status: xxx, }. model_update will be in this format: {status: xxx, }.

The driver should populate volumes_model_update and model_update and return them.

The manager will check volumes_model_update and update db accordingly for each volume. If the driver successfully deleted some volumes but failed to delete others, it should set statuses of the volumes accordingly so that the manager can update db correctly.

If the status in any entry of volumes_model_update is error_deleting or error, the status in model_update will be set to the same if it is not already error_deleting or error.

If the status in model_update is error_deleting or error, the manager will raise an exception and the status of the group will be set to error in the db. If volumes_model_update is not returned by the driver, the manager will set the status of every volume in the group to error in the except block.

If the driver raises an exception during the operation, it will be caught by the try-except block in the manager. The statuses of the group and all volumes in it will be set to error.

For a successful operation, the driver can either build the model_update and volumes_model_update and return them or return None, None. The statuses of the group and all volumes will be set to deleted after the manager deletes them from db.

`delete_group_snapshot`(*context*, *group_snapshot*, *snapshots*)

Deletes a group_snapshot.

Parameters

- **`context`** the context of the caller.
- **`group_snapshot`** the GroupSnapshot object to be deleted.
- **`snapshots`** a list of Snapshot objects in the group_snapshot.

Returns

model_update, snapshots_model_update

param snapshots is a list of objects. It cannot be assigned to snapshots_model_update. snapshots_model_update is a list of of dictionaries. It has to be built by the driver. An entry will be in this format: {id: xxx, status: xxx, }. model_update will be in this format: {status: xxx, }.

The driver should populate snapshots_model_update and model_update and return them.

The manager will check snapshots_model_update and update db accordingly for each snapshot. If the driver successfully deleted some snapshots but failed to delete others, it should set statuses of the snapshots accordingly so that the manager can update db correctly.

If the status in any entry of snapshots_model_update is error_deleting or error, the status in model_update will be set to the same if it is not already error_deleting or error.

If the status in model_update is error_deleting or error, the manager will raise an exception and the status of group_snapshot will be set to error in the db. If snapshots_model_update is not returned by the driver, the manager will set the status of every snapshot to error in the except block.

If the driver raises an exception during the operation, it will be caught by the try-except block in the manager and the statuses of group_snapshot and all snapshots will be set to error.

For a successful operation, the driver can either build the model_update and snapshots_model_update and return them or return None, None. The statuses of group_snapshot and all snapshots will be set to deleted after the manager deletes them from db.

`abstract delete_volume`(*volume*)

Deletes a volume.

If volume_type extra specs includes replication: <is> True then the driver needs to delete the volume replica too.

It is imperative that this operation ensures that the data from the deleted volume cannot leak into new volumes when they are created, as new volumes are likely to belong to a different tenant/project.

If the driver uses custom file locks they should be cleaned on success using cinder.utils.synchronized_remove `disable_replication`(*context*, *group*, *volumes*)

Disables replication for a group and volumes in the group.

Parameters

- `group` group object
- `volumes` list of volume objects in the group

Returns
model_update - dict of group updates

Returns
volume_model_updates - list of dicts of volume updates

`do_setup`(*context*)

Any initialization the volume driver does while starting.

`enable_replication`(*context*, *group*, *volumes*)

Enables replication for a group and volumes in the group.

Parameters

- `group` group object
- `volumes` list of volume objects in the group

Returns
model_update - dict of group updates

Returns
volume_model_updates - list of dicts of volume updates

`abstract ensure_export`(*context*, *volume*)

Synchronously recreates an export for a volume.

`extend_volume`(*volume*, *new_size*)

`failover`(*context*, *volumes*, *secondary_id=None*, *groups=None*)

Like failover but for a host that is clustered.

Most of the time this will be the exact same behavior as failover_host, so if its not overwritten, it is assumed to be the case.

`failover_completed`(*context*, *active_backend_id=None*)

This method is called after failover for clustered backends.

`failover_host`(*context*, *volumes*, *secondary_id=None*, *groups=None*)

Failover a backend to a secondary replication target.

Instructs a replication capable/configured backend to failover to one of its secondary replication targets. host=None is an acceptable input, and leaves it to the driver to failover to the only configured target, or to choose a target on its own. All of the hosts volumes will be passed on to the driver in order for it to determine the replicated volumes on the host, if needed.

Response is a tuple, including the new target backend_id AND a lit of dictionaries with volume_id and updates. Key things to consider (attaching failed-over volumes): - provider_location - provider_auth - provider_id - replication_status

Parameters

- `context` security context
- `volumes` list of volume objects, in case the driver needs to take action on them in some way
- `secondary_id` Specifies rep target backend to fail over to
- `groups` replication groups

Returns
ID of the backend that was failed-over to, model update for volumes, and model update for groups

`failover_replication`(*context*, *group*, *volumes*, *secondary_backend_id=None*)

Fails over replication for a group and volumes in the group.

Parameters

- `group` group object
- `volumes` list of volume objects in the group
- `secondary_backend_id` backend_id of the secondary site

Returns
model_update - dict of group updates

Returns
volume_model_updates - list of dicts of volume updates

`freeze_backend`(*context*)

Notify the backend that its frozen.

We use set to prohibit the creation of any new resources on the backend, or any modifications to existing items on a backend. We set/enforce this by not allowing scheduling of new volumes to the specified backend, and checking at the api for modifications to resources and failing.

In most cases the driver may not need to do anything, but this provides a handle if they need it.

Parameters
`context` security context

Response
True|False

`get_backup_device`(*context*, *backup*)

Get a backup device from an existing volume.

The function returns a volume or snapshot to backup service, and then backup service attaches the device and does backup.

`get_default_filter_function`()

Get the default filter_function string.

Each driver could overwrite the method to return a well-known default string if it is available.

Returns
None

get_default_goodness_function()

Get the default goodness_function string.

Each driver could overwrite the method to return a well-known default string if it is available.

Returns
None

static get_driver_options()

Return the oslo_config options specific to the driver.

get_filter_function()

Get filter_function string.

Returns either the string from the driver instance or global section in cinder.conf. If nothing is specified in cinder.conf, then try to find the default filter_function. When None is returned the scheduler will always pass the driver instance.

Returns
a filter_function string or None

get_goodness_function()

Get good_function string.

Returns either the string from the driver instance or global section in cinder.conf. If nothing is specified in cinder.conf, then try to find the default goodness_function. When None is returned the scheduler will give the lowest score to the driver instance.

Returns
a goodness_function string or None

get_pool(*volume*)

Return pool name where volume reside on.

Parameters
volume The volume hosted by the driver.

Returns
name of the pool where given volume is in.

get_replication_error_status(*context*, *groups*)

Returns error info for replicated groups and its volumes.

Returns
group_model_updates - list of dicts of group updates

if error happens. For example, a dict of a group can be as follows:

```
{'group_id': xxxx,
 'replication_status': fields.ReplicationStatus.ERROR}
```

Returns
volume_model_updates - list of dicts of volume updates

if error happens. For example, a dict of a volume can be as follows:

```
{'volume_id': xxxx,
 'replication_status': fields.ReplicationStatus.ERROR}
```

get_version()

Get the current version of this driver.

get_volume_stats(*refresh=False*)

Get volume stats.

If refresh is True, run update the stats first.

init_capabilities()

Obtain backend volume stats and capabilities list.

This stores a dictionary which is consisted of two parts. First part includes static backend capabilities which are obtained by get_volume_stats(). Second part is properties, which includes parameters correspond to extra specs. This properties part is consisted of cinder standard capabilities and vendor unique properties.

Using this capabilities list, operator can manage/configure backend using key/value from capabilities without specific knowledge of backend.

abstract initialize_connection(*volume*, *connector*)

Allow connection to connector and return connection info.

..note::

Whether or not a volume is cacheable for volume local cache on the hypervisor is normally configured in the volume-type extra-specs. Support may be disabled at the driver level, however, by returning cacheable: False in the conn_info. This will override any setting in the volume-type extra-specs.

Parameters

- **volume** The volume to be attached
- **connector** Dictionary containing information about what is being connected to.

Returns conn_info

A dictionary of connection information.

initialize_connection_snapshot(*snapshot*, *connector*, ***kwargs*)

Allow connection to connector and return connection info.

Parameters

- **snapshot** The snapshot to be attached
- **connector** Dictionary containing information about what is being connected to.

Returns conn_info

A dictionary of connection information. This can optionally include a initiator_updates field.

The initiator_updates field must be a dictionary containing a set_values and/or remove_values field. The set_values field must be a dictionary of key-value pairs to be set/updated in the db.

The remove_values field must be a list of keys, previously set with set_values, that will be deleted from the db.

property initialized

manage_existing(*volume*, *existing_ref*)

Manage exiting stub.

This is for drivers that dont implement manage_existing().

migrate_volume(*context*, *volume*, *host*)

Migrate the volume to the specified host.

This is a stub for drivers that dont implement an enhanced version of this operation.

Refer to `cinder.interface.volume_driver.VolumeDriverCore.migrate_volume` for additional information.

abstract remove_export(*context*, *volume*)

Removes an export for a volume.

remove_export_snapshot(*context*, *snapshot*)

Removes an export for a snapshot.

retype(*context*, *volume*, *new_type*, *diff*, *host*)

Change the type of a volume.

Refer to `cinder.interface.volume_driver.VolumeDriverCore.retype` for additional information.

secure_file_operations_enabled()

Determine if driver is running in Secure File Operations mode.

The Cinder Volume driver needs to query if this driver is running in a secure file operations mode. By default, it is False: any driver that does support secure file operations should override this method.

set_initialized()

set_throttle()

snapshot_revert_use_temp_snapshot()

property supported

classmethod supports_replication_feature(*feature*)

Check if driver class supports replication features.

Feature is a string that must be one of:

- v2.1
- a/a

abstract terminate_connection(*volume*, *connector*, ***kwargs*)

Disallow connection from connector.

Parameters

- **volume** The volume to be disconnected.

- **connector** A dictionary describing the connection with details about the initiator. Can be None.

terminate_connection_snapshot(*snapshot*, *connector*, ***kwargs*)

Disallow connection from connector.

thaw_backend(*context*)

Notify the backend that its unfrozen/thawed.

Returns the backend to a normal state after a freeze operation.

In most cases the driver may not need to do anything, but this provides a handle if they need it.

Parameters
context security context

Response
True|False

unmanage(*volume*)

Unmanage stub.

This is for drivers that dont implement unmanage().

update_group(*context*, *group*, *add_volumes=None*, *remove_volumes=None*)

Updates a group.

Parameters

- **context** the context of the caller.
- **group** the Group object of the group to be updated.
- **add_volumes** a list of Volume objects to be added.
- **remove_volumes** a list of Volume objects to be removed.

Returns
model_update, add_volumes_update, remove_volumes_update

model_update is a dictionary that the driver wants the manager to update upon a successful return. If None is returned, the manager will set the status to available.

add_volumes_update and remove_volumes_update are lists of dictionaries that the driver wants the manager to update upon a successful return. Note that each entry requires a {id: xxx} so that the correct volume entry can be updated. If None is returned, the volume will remain its original status. Also note that you cannot directly assign add_volumes to add_volumes_update as add_volumes is a list of volume objects and cannot be used for db update directly. Same with remove_volumes.

If the driver throws an exception, the status of the group as well as those of the volumes to be added/removed will be set to error.

update_migrated_volume(*ctxt*, *volume*, *new_volume*, *original_volume_status*)

Return model update for migrated volume.

Refer to `cinder.interface.volume_driver.VolumeDriverCore.update_migrated_volume` for additional information.

update_provider_info(*volumes*, *snapshots*)

Get provider info updates from driver.

Parameters

- **volumes** List of Cinder volumes to check for updates
- **snapshots** List of Cinder snapshots to check for updates

Returns

tuple (volume_updates, snapshot_updates)

where volume updates {id: uuid, provider_id: <provider-id>} and snapshot updates {id: uuid, provider_id: <provider-id>}

validate_connector(*connector*)

Fail if connector doesnt contain all the data needed by driver.

static validate_connector_has_setting(*connector*, *setting*)

class CloneableImageVD

Bases: `object`

abstract clone_image(*context*, *volume*, *image_location*, *image_meta*, *image_service*)

Create a volume efficiently from an existing image.

Refer to `cinder.interface.volume_driver.VolumeDriverCore.clone_image` for additional information.

class FibreChannelDriver(**args*, ***kwargs*)

Bases: `VolumeDriver`

Executes commands relating to Fibre Channel volumes.

initialize_connection(*volume*, *connector*)

Initializes the connection and returns connection info.

The driver returns a driver_volume_type of fibre_channel. The target_wwn can be a single entry or a list of wwns that correspond to the list of remote wwn(s) that will export the volume. Example return values:

```
{
    'driver_volume_type': 'fibre_channel',
    'data': {
        'target_discovered': True,
        'target_lun': 1,
        'target_wwn': '1234567890123',
        'discard': False,
    }
}
```

or

```
{
    'driver_volume_type': 'fibre_channel',
    'data': {
```

(continues on next page)

(continued from previous page)

```
            'target_discovered': True,
            'target_lun': 1,
            'target_wwn': ['1234567890123', '0987654321321'],
            'discard': False,
            'addressing_mode': os_brick.constants.SCSI_ADDRESSING_SAM2,
        }
}
```

validate_connector(*connector*)

Fail if connector doesnt contain all the data needed by driver.

Do a check on the connector and ensure that it has wwnns, wwpns.

static validate_connector_has_setting(*connector*, *setting*)

Test for non-empty setting in connector.

class ISCSIDriver(**args*, ***kwargs*)

Bases: *VolumeDriver*

Executes commands relating to ISCSI volumes.

We make use of model provider properties as follows:

provider_location
if present, contains the iSCSI target information in the same format as an ietadm discovery i.e. <ip>:<port>,<portal> <target IQN>

provider_auth
if present, contains a space-separated triple: <auth method> <auth username> <auth password>. *CHAP* is the only auth_method in use at the moment.

initialize_connection(*volume*, *connector*)

Initializes the connection and returns connection info.

The iscsi driver returns a driver_volume_type of iscsi. The format of the driver data is defined in _get_iscsi_properties. Example return value:

```
{
    'driver_volume_type': 'iscsi',
    'data': {
        'target_discovered': True,
        'target_iqn': 'iqn.2010-10.org.openstack:volume-00000001',
        'target_portal': '127.0.0.0.1:3260',
        'volume_id': 1,
        'discard': False,
    }
}
```

If the backend driver supports multiple connections for multipath and for single path with failover, target_portals, target_iqns, target_luns are also populated. In this example also LUN values greater than 255 use flat addressing mode:

```
{
    'driver_volume_type': 'iscsi',
    'data': {
        'target_discovered': False,
        'target_iqn': 'iqn.2010-10.org.openstack:volume1',
        'target_iqns': ['iqn.2010-10.org.openstack:volume1',
                        'iqn.2010-10.org.openstack:volume1-2'],
        'target_portal': '10.0.0.1:3260',
        'target_portals': ['10.0.0.1:3260', '10.0.1.1:3260'],
        'target_lun': 1,
        'target_luns': [1, 1],
        'volume_id': 1,
        'discard': False,
        'addressing_mode': os_brick.constants.SCSI_ADDRESSING_SAM2,
    }
}
```

terminate_connection(*volume*, *connector*, ***kwargs*)

Disallow connection from connector

Parameters

- **volume** The volume to be disconnected.
- **connector** A dictionary describing the connection with details about the initiator. Can be None.

validate_connector(*connector*)

Fail if connector doesnt contain all the data needed by driver.

class ISERDriver(**args*, ***kwargs*)

Bases: *ISCSIDriver*

Executes commands relating to ISER volumes.

We make use of model provider properties as follows:

provider_location
if present, contains the iSER target information in the same format as an ietadm discovery i.e. <ip>:<port>,<portal> <target IQN>

provider_auth
if present, contains a space-separated triple: <auth method> <auth username> <auth password>. *CHAP* is the only auth_method in use at the moment.

initialize_connection(*volume*, *connector*)

Initializes the connection and returns connection info.

The iser driver returns a driver_volume_type of iser. The format of the driver data is defined in _get_iser_properties. Example return value:

```
{
    'driver_volume_type': 'iser',
    'data': {
        'target_discovered': True,
```

(continues on next page)

(continued from previous page)

```
        'target_iqn':
        'iqn.2010-10.org.iser.openstack:volume-00000001',
        'target_portal': '127.0.0.0.1:3260',
        'volume_id': 1,
    }
}
```

class ManageableSnapshotsVD

Bases: `object`

get_manageable_snapshots(*cinder_snapshots*, *marker*, *limit*, *offset*, *sort_keys*, *sort_dirs*)

List snapshots on the backend available for management by Cinder.

Returns a list of dictionaries, each specifying a snapshot in the host, with the following keys: - reference (dictionary): The reference for a snapshot, which can be passed to manage_existing_snapshot. - size (int): The size of the snapshot according to the storage backend, rounded up to the nearest GB. - safe_to_manage (boolean): Whether or not this snapshot is safe to manage according to the storage backend. For example, is the snapshot in use or invalid for any reason. - reason_not_safe (string): If safe_to_manage is False, the reason why. - cinder_id (string): If already managed, provide the Cinder ID. - extra_info (string): Any extra information to return to the user - source_reference (string): Similar to reference, but for the snapshots source volume.

Parameters

- **cinder_snapshots** A list of snapshots in this host that Cinder currently manages, used to determine if a snapshot is manageable or not.
- **marker** The last item of the previous page; we return the next results after this value (after sorting)
- **limit** Maximum number of items to return
- **offset** Number of items to skip after marker
- **sort_keys** List of keys to sort results by (valid keys are identifier and size)
- **sort_dirs** List of directions to sort by, corresponding to sort_keys (valid directions are asc and desc)

manage_existing_snapshot(*snapshot*, *existing_ref*)

Brings an existing backend storage object under Cinder management.

existing_ref is passed straight through from the API requests manage_existing_ref value, and it is up to the driver how this should be interpreted. It should be sufficient to identify a storage object that the driver should somehow associate with the newly-created cinder snapshot structure.

There are two ways to do this:

1. Rename the backend storage object so that it matches the snapshot[name] which is how drivers traditionally map between a cinder snapshot and the associated backend storage object.
2. Place some metadata on the snapshot, or somewhere in the backend, that allows other driver requests (e.g. delete) to locate the backend storage object when required.

If the existing_ref doesnt make sense, or doesnt refer to an existing backend storage object, raise a ManageExistingInvalidReference exception.

Parameters

- `snapshot` Cinder volume snapshot to manage
- `existing_ref` Driver-specific information used to identify a volume snapshot

`manage_existing_snapshot_get_size`(*snapshot*, *existing_ref*)

Return size of snapshot to be managed by manage_existing.

When calculating the size, round up to the next GB.

Parameters

- `snapshot` Cinder volume snapshot to manage
- `existing_ref` Driver-specific information used to identify a volume snapshot

Returns size
Volume snapshot size in GiB (integer)

`unmanage_snapshot`(*snapshot*)

Removes the specified snapshot from Cinder management.

Does not delete the underlying backend storage object.

For most drivers, this will not need to do anything. However, some drivers might use this call as an opportunity to clean up any Cinder-specific configuration that they have associated with the backend storage object.

Parameters
`snapshot` Cinder volume snapshot to unmanage

`class ManageableVD`

Bases: `object`

`get_manageable_volumes`(*cinder_volumes*, *marker*, *limit*, *offset*, *sort_keys*, *sort_dirs*)

List volumes on the backend available for management by Cinder.

Returns a list of dictionaries, each specifying a volume in the host, with the following keys: - reference (dictionary): The reference for a volume, which can be passed to manage_existing. - size (int): The size of the volume according to the storage backend, rounded up to the nearest GB. - safe_to_manage (boolean): Whether or not this volume is safe to manage according to the storage backend. For example, is the volume in use or invalid for any reason. - reason_not_safe (string): If safe_to_manage is False, the reason why. - cinder_id (string): If already managed, provide the Cinder ID. - extra_info (string): Any extra information to return to the user

Parameters

- `cinder_volumes` A list of volumes in this host that Cinder currently manages, used to determine if a volume is manageable or not.
- `marker` The last item of the previous page; we return the next results after this value (after sorting)

- `limit` Maximum number of items to return
- `offset` Number of items to skip after marker
- `sort_keys` List of keys to sort results by (valid keys are identifier and size)
- `sort_dirs` List of directions to sort by, corresponding to sort_keys (valid directions are asc and desc)

`abstract manage_existing`(*volume*, *existing_ref*)

Brings an existing backend storage object under Cinder management.

existing_ref is passed straight through from the API requests manage_existing_ref value, and it is up to the driver how this should be interpreted. It should be sufficient to identify a storage object that the driver should somehow associate with the newly-created cinder volume structure.

There are two ways to do this:

1. Rename the backend storage object so that it matches the, volume[name] which is how drivers traditionally map between a cinder volume and the associated backend storage object.
2. Place some metadata on the volume, or somewhere in the backend, that allows other driver requests (e.g. delete, clone, attach, detach) to locate the backend storage object when required.

If the existing_ref doesnt make sense, or doesnt refer to an existing backend storage object, raise a ManageExistingInvalidReference exception.

The volume may have a volume_type, and the driver can inspect that and compare against the properties of the referenced backend storage object. If they are incompatible, raise a ManageExistingVolumeTypeMismatch, specifying a reason for the failure.

Parameters

- `volume` Cinder volume to manage
- `existing_ref` Driver-specific information used to identify a volume

`abstract manage_existing_get_size`(*volume*, *existing_ref*)

Return size of volume to be managed by manage_existing.

When calculating the size, round up to the next GB.

Parameters

- `volume` Cinder volume to manage
- `existing_ref` Driver-specific information used to identify a volume

Returns size
Volume size in GiB (integer)

`abstract unmanage`(*volume*)

Removes the specified volume from Cinder management.

Does not delete the underlying backend storage object.

For most drivers, this will not need to do anything. However, some drivers might use this call as an opportunity to clean up any Cinder-specific configuration that they have associated with the backend storage object.

Parameters
`volume` Cinder volume to unmanage

`class MigrateVD`

Bases: `object`

`abstract migrate_volume`(*context*, *volume*, *host*)

Migrate the volume to the specified host.

Refer to *cinder.interface.volume_driver.VolumeDriverCore.migrate_volume* for additional information.

`class ProxyVD`

Bases: `object`

Proxy Volume Driver to mark proxy drivers

If a driver uses a proxy class (e.g. by using __setattr__ and __getattr__) without directly inheriting from base volume driver this class can help marking them and retrieve the actual used driver object.

`class VolumeDriver`(*execute=<function execute>*, **args*, ***kwargs*)

Bases: *ManageableVD*, *CloneableImageVD*, *ManageableSnapshotsVD*, *MigrateVD*, *BaseVD*

`accept_transfer`(*context*, *volume*, *new_user*, *new_project*)

`check_for_setup_error`()

`clear_download`(*context*, *volume*)

Clean up after an interrupted image copy.

`clone_image`(*context*, *volume*, *image_location*, *image_meta*, *image_service*)

Create a volume efficiently from an existing image.

Refer to *cinder.interface.volume_driver.VolumeDriverCore.clone_image* for additional information.

`create_cgsnapshot`(*context*, *cgsnapshot*, *snapshots*)

Creates a cgsnapshot.

Parameters

- `context` the context of the caller.
- `cgsnapshot` the dictionary of the cgsnapshot to be created.
- `snapshots` a list of snapshot dictionaries in the cgsnapshot.

Returns
model_update, snapshots_model_update

param snapshots is retrieved directly from the db. It is a list of cinder.db.sqlalchemy.models.Snapshot to be precise. It cannot be assigned to snapshots_model_update. snapshots_model_update is a list of dictionaries. It has to be built by the driver. An entry will be in this format: {id: xxx, status: xxx, }. model_update will be in this format: {status: xxx, }.

The driver should populate snapshots_model_update and model_update and return them.

The manager will check snapshots_model_update and update db accordingly for each snapshot. If the driver successfully deleted some snapshots but failed to delete others, it should set statuses of the snapshots accordingly so that the manager can update db correctly.

If the status in any entry of snapshots_model_update is error, the status in model_update will be set to the same if it is not already error.

If the status in model_update is error, the manager will raise an exception and the status of cgsnapshot will be set to error in the db. If snapshots_model_update is not returned by the driver, the manager will set the status of every snapshot to error in the except block.

If the driver raises an exception during the operation, it will be caught by the try-except block in the manager and the statuses of cgsnapshot and all snapshots will be set to error.

For a successful operation, the driver can either build the model_update and snapshots_model_update and return them or return None, None. The statuses of cgsnapshot and all snapshots will be set to available at the end of the manager function.

`create_consistencygroup`(*context*, *group*)

Creates a consistencygroup.

Parameters

- `context` the context of the caller.
- `group` the dictionary of the consistency group to be created.

Returns

model_update

model_update will be in this format: {status: xxx, }.

If the status in model_update is error, the manager will throw an exception and it will be caught in the try-except block in the manager. If the driver throws an exception, the manager will also catch it in the try-except block. The group status in the db will be changed to error.

For a successful operation, the driver can either build the model_update and return it or return None. The group status will be set to available.

`create_consistencygroup_from_src`(*context*, *group*, *volumes*, *cgsnapshot=None*, *snapshots=None*, *source_cg=None*, *source_vols=None*)

Creates a consistencygroup from source.

Parameters

- `context` the context of the caller.
- `group` the dictionary of the consistency group to be created.
- `volumes` a list of volume dictionaries in the group.
- `cgsnapshot` the dictionary of the cgsnapshot as source.
- `snapshots` a list of snapshot dictionaries in the cgsnapshot.
- `source_cg` the dictionary of a consistency group as source.
- `source_vols` a list of volume dictionaries in the source_cg.

Returns

model_update, volumes_model_update

The source can be cgsnapshot or a source cg.

param volumes is retrieved directly from the db. It is a list of cinder.db.sqlalchemy.models.Volume to be precise. It cannot be assigned to volumes_model_update. volumes_model_update is a list of dictionaries. It has to be built by the driver. An entry will be in this format: {id: xxx, status: xxx, }. model_update will be in this format: {status: xxx, }.

To be consistent with other volume operations, the manager will assume the operation is successful if no exception is thrown by the driver. For a successful operation, the driver can either build the model_update and volumes_model_update and return them or return None, None.

create_export(*context*, *volume*, *connector*)

Exports the volume.

Can optionally return a Dictionary of changes to the volume object to be persisted.

create_export_snapshot(*context*, *snapshot*, *connector*)

Exports the snapshot.

Can optionally return a Dictionary of changes to the snapshot object to be persisted.

create_snapshot(*snapshot*)

Creates a snapshot.

create_volume(*volume*)

Creates a volume.

Can optionally return a Dictionary of changes to the volume object to be persisted.

If volume_type extra specs includes capabilities:replication <is> True the driver needs to create a volume replica (secondary), and setup replication between the newly created volume and the secondary volume. Returned dictionary should include:

```
volume['replication_status'] = 'copying'
volume['replication_extended_status'] = <driver specific value>
volume['driver_data'] = <driver specific value>
```

create_volume_from_snapshot(*volume*, *snapshot*)

Creates a volume from a snapshot.

If volume_type extra specs includes replication: <is> True the driver needs to create a volume replica (secondary), and setup replication between the newly created volume and the secondary volume.

delete_cgsnapshot(*context*, *cgsnapshot*, *snapshots*)

Deletes a cgsnapshot.

Parameters

- **context** the context of the caller.
- **cgsnapshot** the dictionary of the cgsnapshot to be deleted.
- **snapshots** a list of snapshot dictionaries in the cgsnapshot.

Returns

model_update, snapshots_model_update

param snapshots is retrieved directly from the db. It is a list of cinder.db.sqlalchemy.models.Snapshot to be precise. It cannot be assigned to snapshots_model_update. snapshots_model_update is a list of dictionaries. It has to be built by the driver. An entry will be in this format: {id: xxx, status: xxx, }. model_update will be in this format: {status: xxx, }.

The driver should populate snapshots_model_update and model_update and return them.

The manager will check snapshots_model_update and update db accordingly for each snapshot. If the driver successfully deleted some snapshots but failed to delete others, it should set statuses of the snapshots accordingly so that the manager can update db correctly.

If the status in any entry of snapshots_model_update is error_deleting or error, the status in model_update will be set to the same if it is not already error_deleting or error.

If the status in model_update is error_deleting or error, the manager will raise an exception and the status of cgsnapshot will be set to error in the db. If snapshots_model_update is not returned by the driver, the manager will set the status of every snapshot to error in the except block.

If the driver raises an exception during the operation, it will be caught by the try-except block in the manager and the statuses of cgsnapshot and all snapshots will be set to error.

For a successful operation, the driver can either build the model_update and snapshots_model_update and return them or return None, None. The statuses of cgsnapshot and all snapshots will be set to deleted after the manager deletes them from db.

`delete_consistencygroup`(*context*, *group*, *volumes*)

Deletes a consistency group.

Parameters

- **context** the context of the caller.
- **group** the dictionary of the consistency group to be deleted.
- **volumes** a list of volume dictionaries in the group.

Returns

model_update, volumes_model_update

param volumes is retrieved directly from the db. It is a list of cinder.db.sqlalchemy.models.Volume to be precise. It cannot be assigned to volumes_model_update. volumes_model_update is a list of dictionaries. It has to be built by the driver. An entry will be in this format: {id: xxx, status: xxx, }. model_update will be in this format: {status: xxx, }.

The driver should populate volumes_model_update and model_update and return them.

The manager will check volumes_model_update and update db accordingly for each volume. If the driver successfully deleted some volumes but failed to delete others, it should set statuses of the volumes accordingly so that the manager can update db correctly.

If the status in any entry of volumes_model_update is error_deleting or error, the status in model_update will be set to the same if it is not already error_deleting or error.

If the status in model_update is error_deleting or error, the manager will raise an exception and the status of the group will be set to error in the db. If volumes_model_update is not returned by the driver, the manager will set the status of every volume in the group to error in the except block.

If the driver raises an exception during the operation, it will be caught by the try-except block in the manager. The statuses of the group and all volumes in it will be set to error.

For a successful operation, the driver can either build the model_update and volumes_model_update and return them or return None, None. The statuses of the group and all volumes will be set to deleted after the manager deletes them from db.

`delete_snapshot`(*snapshot*)

Deletes a snapshot.

If the driver uses custom file locks they should be cleaned on success using cinder.utils.synchronized_remove

`delete_volume`(*volume*)

Deletes a volume.

If volume_type extra specs includes replication: <is> True then the driver needs to delete the volume replica too.

It is imperative that this operation ensures that the data from the deleted volume cannot leak into new volumes when they are created, as new volumes are likely to belong to a different tenant/project.

If the driver uses custom file locks they should be cleaned on success using cinder.utils.synchronized_remove

`ensure_export`(*context*, *volume*)

Synchronously recreates an export for a volume.

`extend_volume`(*volume*, *new_size*)

`get_manageable_snapshots`(*cinder_snapshots*, *marker*, *limit*, *offset*, *sort_keys*, *sort_dirs*)

List snapshots on the backend available for management by Cinder.

Returns a list of dictionaries, each specifying a snapshot in the host, with the following keys: - reference (dictionary): The reference for a snapshot, which can be passed to manage_existing_snapshot. - size (int): The size of the snapshot according to the storage backend, rounded up to the nearest GB. - safe_to_manage (boolean): Whether or not this snapshot is safe to manage according to the storage backend. For example, is the snapshot in use or invalid for any reason. - reason_not_safe (string): If safe_to_manage is False, the reason why. - cinder_id (string): If already managed, provide the Cinder ID. - extra_info (string): Any extra information to return to the user - source_reference (string): Similar to reference, but for the snapshots source volume.

Parameters

- **cinder_snapshots** A list of snapshots in this host that Cinder currently manages, used to determine if a snapshot is manageable or not.
- **marker** The last item of the previous page; we return the next results after this value (after sorting)
- **limit** Maximum number of items to return
- **offset** Number of items to skip after marker
- **sort_keys** List of keys to sort results by (valid keys are identifier and size)

- **sort_dirs** List of directions to sort by, corresponding to sort_keys (valid directions are asc and desc)

`get_manageable_volumes`(*cinder_volumes*, *marker*, *limit*, *offset*, *sort_keys*, *sort_dirs*)

List volumes on the backend available for management by Cinder.

Returns a list of dictionaries, each specifying a volume in the host, with the following keys: - reference (dictionary): The reference for a volume, which can be passed to manage_existing. - size (int): The size of the volume according to the storage backend, rounded up to the nearest GB. - safe_to_manage (boolean): Whether or not this volume is safe to manage according to the storage backend. For example, is the volume in use or invalid for any reason. - reason_not_safe (string): If safe_to_manage is False, the reason why. - cinder_id (string): If already managed, provide the Cinder ID. - extra_info (string): Any extra information to return to the user

Parameters

- **cinder_volumes** A list of volumes in this host that Cinder currently manages, used to determine if a volume is manageable or not.
- **marker** The last item of the previous page; we return the next results after this value (after sorting)
- **limit** Maximum number of items to return
- **offset** Number of items to skip after marker
- **sort_keys** List of keys to sort results by (valid keys are identifier and size)
- **sort_dirs** List of directions to sort by, corresponding to sort_keys (valid directions are asc and desc)

`get_pool`(*volume*)

Return pool name where volume reside on.

Parameters
volume The volume hosted by the driver.

Returns
name of the pool where given volume is in.

`initialize_connection`(*volume*, *connector*, ***kwargs*)

Allow connection to connector and return connection info.

..note::
Whether or not a volume is cacheable for volume local cache on the hypervisor is normally configured in the volume-type extra-specs. Support may be disabled at the driver level, however, by returning cacheable: False in the conn_info. This will override any setting in the volume-type extra-specs.

Parameters

- **volume** The volume to be attached
- **connector** Dictionary containing information about what is being connected to.

Returns conn_info
A dictionary of connection information.

initialize_connection_snapshot(*snapshot*, *connector*, ***kwargs*)

Allow connection from connector for a snapshot.

local_path(*volume*)

manage_existing(*volume*, *existing_ref*)

Brings an existing backend storage object under Cinder management.

existing_ref is passed straight through from the API requests manage_existing_ref value, and it is up to the driver how this should be interpreted. It should be sufficient to identify a storage object that the driver should somehow associate with the newly-created cinder volume structure.

There are two ways to do this:

1. Rename the backend storage object so that it matches the, volume[name] which is how drivers traditionally map between a cinder volume and the associated backend storage object.
2. Place some metadata on the volume, or somewhere in the backend, that allows other driver requests (e.g. delete, clone, attach, detach) to locate the backend storage object when required.

If the existing_ref doesnt make sense, or doesnt refer to an existing backend storage object, raise a ManageExistingInvalidReference exception.

The volume may have a volume_type, and the driver can inspect that and compare against the properties of the referenced backend storage object. If they are incompatible, raise a ManageExistingVolumeTypeMismatch, specifying a reason for the failure.

Parameters

- **volume** Cinder volume to manage
- **existing_ref** Driver-specific information used to identify a volume

manage_existing_get_size(*volume*, *existing_ref*)

Return size of volume to be managed by manage_existing.

When calculating the size, round up to the next GB.

Parameters

- **volume** Cinder volume to manage
- **existing_ref** Driver-specific information used to identify a volume

Returns size

Volume size in GiB (integer)

manage_existing_snapshot(*snapshot*, *existing_ref*)

Brings an existing backend storage object under Cinder management.

existing_ref is passed straight through from the API requests manage_existing_ref value, and it is up to the driver how this should be interpreted. It should be sufficient to identify a storage object that the driver should somehow associate with the newly-created cinder snapshot structure.

There are two ways to do this:

1. Rename the backend storage object so that it matches the snapshot[name] which is how drivers traditionally map between a cinder snapshot and the associated backend storage object.
2. Place some metadata on the snapshot, or somewhere in the backend, that allows other driver requests (e.g. delete) to locate the backend storage object when required.

If the existing_ref doesnt make sense, or doesnt refer to an existing backend storage object, raise a ManageExistingInvalidReference exception.

Parameters

- **snapshot** Cinder volume snapshot to manage
- **existing_ref** Driver-specific information used to identify a volume snapshot

manage_existing_snapshot_get_size(*snapshot*, *existing_ref*)

Return size of snapshot to be managed by manage_existing.

When calculating the size, round up to the next GB.

Parameters

- **snapshot** Cinder volume snapshot to manage
- **existing_ref** Driver-specific information used to identify a volume snapshot

Returns size
Volume snapshot size in GiB (integer)

migrate_volume(*context*, *volume*, *host*)

Migrate the volume to the specified host.

Refer to *cinder.interface.volume_driver.VolumeDriverCore.migrate_volume* for additional information.

remove_export(*context*, *volume*)

Removes an export for a volume.

remove_export_snapshot(*context*, *snapshot*)

Removes an export for a snapshot.

retype(*context*, *volume*, *new_type*, *diff*, *host*)

Change the type of a volume.

Refer to *cinder.interface.volume_driver.VolumeDriverCore.retype* for additional information.

revert_to_snapshot(*context*, *volume*, *snapshot*)

Revert volume to snapshot.

Note: the revert process should not change the volumes current size, that means if the driver shrank the volume during the process, it should extend the volume internally.

terminate_connection(*volume*, *connector*, ***kwargs*)

Disallow connection from connector

Parameters

- **volume** The volume to be disconnected.
- **connector** A dictionary describing the connection with details about the initiator. Can be None.

terminate_connection_snapshot(*snapshot*, *connector*, ***kwargs*)

Disallow connection from connector for a snapshot.

unmanage(*volume*)

Removes the specified volume from Cinder management.

Does not delete the underlying backend storage object.

For most drivers, this will not need to do anything. However, some drivers might use this call as an opportunity to clean up any Cinder-specific configuration that they have associated with the backend storage object.

Parameters

volume Cinder volume to unmanage

unmanage_snapshot(*snapshot*)

Unmanage the specified snapshot from Cinder management.

update_consistencygroup(*context*, *group*, *add_volumes=None*, *remove_volumes=None*)

Updates a consistency group.

Parameters

- **context** the context of the caller.
- **group** the dictionary of the consistency group to be updated.
- **add_volumes** a list of volume dictionaries to be added.
- **remove_volumes** a list of volume dictionaries to be removed.

Returns

model_update, add_volumes_update, remove_volumes_update

model_update is a dictionary that the driver wants the manager to update upon a successful return. If None is returned, the manager will set the status to available.

add_volumes_update and remove_volumes_update are lists of dictionaries that the driver wants the manager to update upon a successful return. Note that each entry requires a {id: xxx} so that the correct volume entry can be updated. If None is returned, the volume will remain its original status. Also note that you cannot directly assign add_volumes to add_volumes_update as add_volumes is a list of cinder.db.sqlalchemy.models.Volume objects and cannot be used for db update directly. Same with remove_volumes.

If the driver throws an exception, the status of the group as well as those of the volumes to be added/removed will be set to error.

cinder.volume.driver_utils module

class VolumeDriverUtils(*namespace*, *db*)

Bases: `object`

get_driver_initiator_data(*initiator*, *ctxt=None*)

insert_driver_initiator_data(*initiator*, *key*, *value*, *ctxt=None*)

Update the initiator data at key with value.

If the key has already been set to something return False, otherwise if saved successfully return True.

cinder.volume.group_types module

Built-in group type properties.

add_group_type_access(*context*, *group_type_id*, *project_id*)

Add access to group type for project_id.

create(*context*, *name*, *group_specs=None*, *is_public=True*, *projects=None*, *description=None*)

Creates group types.

destroy(*context*, *id*)

Marks group types as deleted.

get_all_group_types(*context*, *inactive=0*, *filters=None*, *marker=None*, *limit=None*, *sort_keys=None*, *sort_dirs=None*, *offset=None*, *list_result=False*)

Get all non-deleted group_types.

Pass true as argument if you want deleted group types returned also.

get_default_cgsnapshot_type()

Get the default group type for migrating cgsnapshots.

Get the default group type for migrating consistencygroups to groups and cgsnapshots to group_snapshots.

get_default_group_type()

Get the default group type.

get_group_type(*ctxt*, *id*, *expected_fields=None*)

Retrieves single group type by id.

get_group_type_by_name(*context*, *name*)

Retrieves single group type by name.

get_group_type_specs(*group_type_id*, *key=False*)

is_default_cgsnapshot_type(*group_type_id*)

is_public_group_type(*context*, *group_type_id*)

Return is_public boolean value of group type

remove_group_type_access(*context*, *group_type_id*, *project_id*)

Remove access to group type for project_id.

update(*context*, *id*, *name*, *description*, *is_public=None*)

Update group type by id.

cinder.volume.manager module

Volume manager manages creating, attaching, detaching, and persistent storage.

Persistent storage volumes keep their state independent of instances. You can attach to an instance, terminate the instance, spawn a new instance (even one from a different image) and re-attach the volume with the same data intact.

Related Flags

volume_manager
: The module name of a class derived from `manager.Manager` (default: `cinder.volume.manager.Manager`).

volume_driver
: Used by `Manager`. Defaults to `cinder.volume.drivers.lvm.LVMVolumeDriver`.

volume_group
: Name of the group that will contain exported volumes (default: *cinder-volumes*)

num_shell_tries
: Number of times to attempt to run commands (default: 3)

class VolumeManager(*volume_driver=None*, *service_name: str | None = None*, **args*, ***kwargs*)

Bases: *CleanableManager*, *SchedulerDependentManager*

Manages attachable block storage devices.

FAILBACK_SENTINEL = 'default'

RPC_API_VERSION = '3.19'

accept_transfer(*context*, *volume_id*, *new_user*, *new_project*, *no_snapshots=False*) → dict

attach_volume(*context*, *volume_id*, *instance_uuid*, *host_name*, *mountpoint*, *mode*, *volume=None*) → *VolumeAttachment*

Updates db to show volume is attached.

attachment_delete(*context: RequestContext*, *attachment_id: str*, *vref: Volume*) → None

Delete/Detach the specified attachment.

Notifies the backend device that were detaching the specified attachment instance.

param: attachment_id: Attachment id to remove param: vref: Volume object associated with the attachment

attachment_update(*context: RequestContext*, *vref: Volume*, *connector: dict*, *attachment_id: str*) → dict[str, Any]

Update/Finalize an attachment.

This call updates a valid attachment record to associate with a volume and provide the caller with the proper connection info. Note that this call requires an *attachment_ref*. Its expected that prior to this call that the volume and an attachment UUID has been reserved.

param: vref: Volume object to create attachment for param: connector: Connector object to use for attachment creation param: attachment_ref: ID of the attachment record to update

copy_volume_to_image(*context: RequestContext*, *volume_id: str*, *image_meta: dict*) → None
: Uploads the specified volume to Glance.

  image_meta is a dictionary containing the following keys: id, container_format, disk_format

create_group(*context: RequestContext*, *group*) → *Group*
: Creates the group.

create_group_from_src(*context: RequestContext*, *group: Group*, *group_snapshot: GroupSnapshot | None = None*, *source_group=None*) → *Group*
: Creates the group from source.

  The source can be a group snapshot or a source group.

create_group_snapshot(*context: RequestContext*, *group_snapshot: GroupSnapshot*) → *GroupSnapshot*
: Creates the group_snapshot.

create_snapshot(*context*, *snapshot*) → UUIDField
: Creates and exports the snapshot.

create_volume(*context*, *volume*, *request_spec=None*, *filter_properties=None*, *allow_reschedule=True*) → UUIDField
: Creates the volume.

delete_group(*context: RequestContext*, *group: Group*) → None
: Deletes group and the volumes in the group.

delete_group_snapshot(*context: RequestContext*, *group_snapshot: GroupSnapshot*) → None
: Deletes group_snapshot.

delete_snapshot(*context: RequestContext*, *snapshot: Snapshot*, *unmanage_only: bool = False*) → bool | None
: Deletes and unexports snapshot.

delete_volume(*context: RequestContext*, *volume: Volume*, *unmanage_only=False*, *cascade=False*) → bool | None
: Deletes and unexports volume.

  1. Delete a volume(normal case) Delete a volume and update quotas.
  2. Delete a migration volume If deleting the volume in a migration, we want to skip quotas but we need database updates for the volume.
  3. Delete a temp volume for backup If deleting the temp volume for backup, we want to skip quotas but we need database updates for the volume.

detach_volume(*context*, *volume_id*, *attachment_id=None*, *volume=None*) → None
: Updates db to show volume is detached.

disable_replication(*ctxt: RequestContext*, *group: Group*) → None
: Disable replication.

driver_delete_snapshot(*snapshot*)

driver_delete_volume(*volume*)

enable_replication(*ctxt: RequestContext*, *group: Group*) → None

Enable replication.

extend_volume(*context: RequestContext*, *volume: Volume*, *new_size: int*, *reservations*) → None

extend_volume_completion(*context: RequestContext*, *volume: Volume*, *new_size: int*, *reservations: list[str]*, *error: bool*) → None

failover(*context: RequestContext*, *secondary_backend_id=None*) → None

Failover a backend to a secondary replication target.

Instructs a replication capable/configured backend to failover to one of its secondary replication targets. host=None is an acceetable input, and leaves it to the driver to failover to the only configured target, or to choose a target on its own. All of the hosts volumes will be passed on to the driver in order for it to determine the replicated volumes on the host, if needed.

Parameters

- **context** security context
- **secondary_backend_id** Specifies backend_id to fail over to

failover_completed(*context: RequestContext*, *updates*) → None

Finalize failover of this backend.

When a service is clustered and replicated the failover has 2 stages, one that does the failover of the volumes and another that finalizes the failover of the services themselves.

This method takes care of the last part and is called from the service doing the failover of the volumes after finished processing the volumes.

failover_host(*context: RequestContext*, *secondary_backend_id=None*) → None

Failover a backend to a secondary replication target.

Instructs a replication capable/configured backend to failover to one of its secondary replication targets. host=None is an acceetable input, and leaves it to the driver to failover to the only configured target, or to choose a target on its own. All of the hosts volumes will be passed on to the driver in order for it to determine the replicated volumes on the host, if needed.

Parameters

- **context** security context
- **secondary_backend_id** Specifies backend_id to fail over to

failover_replication(*ctxt: RequestContext*, *group: Group*, *allow_attached_volume: bool = False*, *secondary_backend_id=None*) → None

Failover replication.

finish_failover(*context: RequestContext*, *service*, *updates*) → None

Completion of the failover locally or via RPC.

freeze_host(*context: RequestContext*) → bool

Freeze management plane on this backend.

Basically puts the control/management plane into a Read Only state. We should handle this in the scheduler, however this is provided to let the driver know in case it needs/wants to do something specific on the backend.

Parameters
`context` security context

get_backup_device(*ctxt: RequestContext*, *backup: Backup*, *want_objects: bool = False*, *async_call: bool = False*)

get_capabilities(*context: RequestContext*, *discover: bool*)

Get capabilities of backend storage.

get_manageable_snapshots(*ctxt: RequestContext*, *marker*, *limit: int | None*, *offset: int | None*, *sort_keys*, *sort_dirs*, *want_objects=False*)

get_manageable_volumes(*ctxt: RequestContext*, *marker*, *limit: int | None*, *offset: int | None*, *sort_keys*, *sort_dirs*, *want_objects=False*) → list

init_host(*added_to_cluster=None*, ***kwargs*) → None

Perform any required initialization.

init_host_with_rpc() → None

A hook for service to do jobs after RPC is ready.

Like init_host(), this method is a hook where services get a chance to execute tasks that *need* RPC. Child classes should override this method.

initialize_connection(*context*, *volume: Volume*, *connector: dict*) → dict

Prepare volume for connection from host represented by connector.

This method calls the driver initialize_connection and returns it to the caller. The connector parameter is a dictionary with information about the host that will connect to the volume in the following format:

```
{
    "ip": "<ip>",
    "initiator": "<initiator>"
}
```

ip:
the ip address of the connecting machine

initiator:
the iscsi initiator name of the connecting machine. This can be None if the connecting machine does not support iscsi connections.

driver is responsible for doing any necessary security setup and returning a connection_info dictionary in the following format:

```
{
    "driver_volume_type": "<driver_volume_type>",
    "data": "<data>"
}
```

driver_volume_type:
a string to identify the type of volume. This can be used by the calling code to determine the strategy for connecting to the volume. This could be iscsi, rbd, etc.

data:
this is the data that the calling code will use to connect to the volume. Keep in mind that this will be serialized to json in various places, so it should not contain any non-json data types.

initialize_connection_snapshot(*ctxt*, *snapshot_id: UUIDField*, *connector: dict*) → dict

is_working() → bool

Return if Manager is ready to accept requests.

This is to inform Service class that in case of volume driver initialization failure the manager is actually down and not ready to accept any requests.

list_replication_targets(*ctxt:* RequestContext, *group:* Group) → dict[str, list]

Provide a means to obtain replication targets for a group.

This method is used to find the replication_device config info. backend_id is a required key in replication_device.

Response Example for admin:

```
{
    "replication_targets": [
        {
            "backend_id": "vendor-id-1",
            "unique_key": "val1"
        },
        {
            "backend_id": "vendor-id-2",
            "unique_key": "val2"
        }
    ]
}
```

Response example for non-admin:

```
{
    "replication_targets": [
        {
            "backend_id": "vendor-id-1"
        },
        {
            "backend_id": "vendor-id-2"
        }
    ]
}
```

manage_existing(*ctxt:* RequestContext, *volume:* Volume, *ref=None*) → UUIDField

manage_existing_snapshot(*ctxt: RequestContext*, *snapshot: Snapshot*, *ref=None*) → UUIDField

migrate_volume(*ctxt: RequestContext*, *volume*, *host*, *force_host_copy: bool = False*, *new_type_id=None*) → None

Migrate the volume to the specified host (called on source host).

migrate_volume_completion(*ctxt: RequestContext*, *volume*, *new_volume*, *error=False*) → UUIDField

publish_service_capabilities(*context: RequestContext*) → None

Collect driver status and then publish.

reimage(*context*, *volume*, *image_meta*)

Reimage a volume with specific image.

remove_export(*context*, *volume_id: UUIDField*) → None

Removes an export for a volume.

remove_export_snapshot(*ctxt*, *snapshot_id: UUIDField*) → None

Removes an export for a snapshot.

retype(*context: RequestContext*, *volume: Volume*, *new_type_id: str*, *host*, *migration_policy: str = 'never'*, *reservations=None*, *old_reservations=None*) → None

revert_to_snapshot(*context*, *volume*, *snapshot*) → None

Revert a volume to a snapshot.

The process of reverting to snapshot consists of several steps: 1. create a snapshot for backup (in case of data loss) 2.1. use drivers specific logic to revert volume 2.2. try the generic way to revert volume if drivers method is missing 3. delete the backup snapshot

secure_file_operations_enabled(*ctxt: RequestContext*, *volume: Volume | None*) → bool

target = <Target version=3.19>

terminate_connection(*context*, *volume_id: UUIDField*, *connector: dict*, *force=False*) → None

Cleanup connection from host represented by connector.

The format of connector is the same as for initialize_connection.

terminate_connection_snapshot(*ctxt*, *snapshot_id: UUIDField*, *connector: dict*, *force=False*) → None

thaw_host(*context: RequestContext*) → bool

UnFreeze management plane on this backend.

Basically puts the control/management plane back into a normal state. We should handle this in the scheduler, however this is provided to let the driver know in case it needs/wants to do something specific on the backend.

Parameters

context security context

update_group(*context: RequestContext*, *group*, *add_volumes: str | None = None*, *remove_volumes: str | None = None*) → None

Updates group.

Update group by adding volumes to the group, or removing volumes from the group.

update_migrated_volume(*ctxt: RequestContext*, *volume: Volume*, *new_volume: Volume*, *volume_status*) → None

Finalize migration process on backend device.

clean_snapshot_locks(*func*)

clean_volume_locks(*func*)

cinder.volume.qos_specs module

The QoS Specs Implementation

associate_qos_with_type(*context*, *specs_id*, *type_id*)

Associate qos_specs with volume type.

Associate target qos specs with specific volume type.

Parameters

- **specs_id** qos specs ID to associate with
- **type_id** volume type ID to associate with

Raises

- *VolumeTypeNotFound* if volume type doesnt exist
- *QoSSpecsNotFound* if qos specs doesnt exist
- *InvalidVolumeType* if volume type is already associated with qos specs other than given one.
- *QoSSpecsAssociateFailed* if there was general DB error

create(*context*, *name*, *specs=None*)

Creates qos_specs.

Parameters

specs Dictionary that contains specifications for QoS

Expected format of the input parameter:

```
{
   'consumer': 'front-end',
   'total_iops_sec': 1000,
   'total_bytes_sec': 1024000
}
```

delete(*context*, *qos_specs_id*, *force=False*)

Marks qos specs as deleted.

force parameter is a flag to determine whether should destroy should continue when there were entities associated with the qos specs. force=True indicates caller would like to mark qos specs as deleted even if there was entities associate with target qos specs. Trying to delete a qos specs still associated with entities will cause QoSSpecsInUse exception if force=False (default).

delete_keys(*context*, *qos_specs_id*, *keys*)

Marks specified key of target qos specs as deleted.

disassociate_all(*context*, *specs_id*)

Disassociate qos_specs from all entities.

disassociate_qos_specs(*context*, *specs_id*, *type_id*)

Disassociate qos_specs from volume type.

get_all_specs(*context*, *filters=None*, *marker=None*, *limit=None*, *offset=None*, *sort_keys=None*, *sort_dirs=None*)

Get all non-deleted qos specs.

get_associations(*context*, *qos_specs_id*)

Get all associations of given qos specs.

get_qos_specs(*ctxt*, *spec_id*)

Retrieves single qos specs by id.

update(*context*, *qos_specs_id*, *specs*)

Update qos specs.

Parameters

specs

dictionary that contains key/value pairs for updating
existing specs.

e.g. {consumer: front-end,
total_iops_sec: 500, total_bytes_sec: 512000,}

cinder.volume.rpcapi module

class VolumeAPI

Bases: RPCAPI

Client side of the volume rpc API.

API version history:

```
1.0 - Initial version.
1.1 - Adds clone volume option to create_volume.
1.2 - Add publish_service_capabilities() method.
1.3 - Pass all image metadata (not just ID) in copy_volume_to_image.
1.4 - Add request_spec, filter_properties and
      allow_reschedule arguments to create_volume().
1.5 - Add accept_transfer.
1.6 - Add extend_volume.
1.7 - Adds host_name parameter to attach_volume()
      to allow attaching to host rather than instance.
1.8 - Add migrate_volume, rename_volume.
1.9 - Add new_user and new_project to accept_transfer.
```

(continues on next page)

(continued from previous page)

```
1.10 - Add migrate_volume_completion, remove rename_volume.
1.11 - Adds mode parameter to attach_volume()
       to support volume read-only attaching.
1.12 - Adds retype.
1.13 - Adds create_export.
1.14 - Adds reservation parameter to extend_volume().
1.15 - Adds manage_existing and unmanage_only flag to delete_volume.
1.16 - Removes create_export.
1.17 - Add replica option to create_volume, promote_replica and
       sync_replica.
1.18 - Adds create_consistencygroup, delete_consistencygroup,
       create_cgsnapshot, and delete_cgsnapshot. Also adds
       the consistencygroup_id parameter in create_volume.
1.19 - Adds update_migrated_volume
1.20 - Adds support for sending objects over RPC in create_snapshot()
       and delete_snapshot()
1.21 - Adds update_consistencygroup.
1.22 - Adds create_consistencygroup_from_src.
1.23 - Adds attachment_id to detach_volume.
1.24 - Removed duplicated parameters: snapshot_id, image_id,
       source_volid, source_replicaid, consistencygroup_id and
       cgsnapshot_id from create_volume. All off them are already
       passed either in request_spec or available in the DB.
1.25 - Add source_cg to create_consistencygroup_from_src.
1.26 - Adds support for sending objects over RPC in
       create_consistencygroup(), create_consistencygroup_from_src(),
       update_consistencygroup() and delete_consistencygroup().
1.27 - Adds support for replication V2
1.28 - Adds manage_existing_snapshot
1.29 - Adds get_capabilities.
1.30 - Adds remove_export
1.31 - Updated: create_consistencygroup_from_src(), create_cgsnapshot()
       and delete_cgsnapshot() to cast method only with necessary
       args. Forwarding CGSnapshot object instead of CGSnapshot_id.
1.32 - Adds support for sending objects over RPC in create_volume().
1.33 - Adds support for sending objects over RPC in delete_volume().
1.34 - Adds support for sending objects over RPC in retype().
1.35 - Adds support for sending objects over RPC in extend_volume().
1.36 - Adds support for sending objects over RPC in migrate_volume(),
       migrate_volume_completion(), and update_migrated_volume().
1.37 - Adds old_reservations parameter to retype to support quota
       checks in the API.
1.38 - Scaling backup service, add get_backup_device() and
       secure_file_operations_enabled()
1.39 - Update replication methods to reflect new backend rep strategy
1.40 - Add cascade option to delete_volume().

... Mitaka supports messaging version 1.40. Any changes to existing
methods in 1.x after that point should be done so that they can handle
```

(continues on next page)

(continued from previous page)

```
the version_cap being set to 1.40.

2.0  - Remove 1.x compatibility
2.1  - Add get_manageable_volumes() and get_manageable_snapshots().
2.2  - Adds support for sending objects over RPC in manage_existing().
2.3  - Adds support for sending objects over RPC in
       initialize_connection().
2.4  - Sends request_spec as object in create_volume().
2.5  - Adds create_group, delete_group, and update_group
2.6  - Adds create_group_snapshot, delete_group_snapshot, and
       create_group_from_src().

... Newton supports messaging version 2.6. Any changes to existing
methods in 2.x after that point should be done so that they can handle
the version_cap being set to 2.6.

3.0  - Drop 2.x compatibility
3.1  - Remove promote_replica and reenable_replication. This is
       non-backward compatible, but the user-facing API was removed
       back in Mitaka when introducing cheesecake replication.
3.2  - Adds support for sending objects over RPC in
       get_backup_device().
3.3  - Adds support for sending objects over RPC in attach_volume().
3.4  - Adds support for sending objects over RPC in detach_volume().
3.5  - Adds support for cluster in retype and migrate_volume
3.6  - Switch to use oslo.messaging topics to indicate backends instead
       of @backend suffixes in server names.
3.7  - Adds do_cleanup method to do volume cleanups from other nodes
       that we were doing in init_host.
3.8  - Make failover_host cluster aware and add failover_completed.
3.9  - Adds new attach/detach methods
3.10 - Returning objects instead of raw dictionaries in
       get_manageable_volumes & get_manageable_snapshots
3.11 - Removes create_consistencygroup, delete_consistencygroup,
       create_cgsnapshot, delete_cgsnapshot, update_consistencygroup,
       and create_consistencygroup_from_src.
3.12 - Adds set_log_levels and get_log_levels
3.13 - Add initialize_connection_snapshot,
       terminate_connection_snapshot, and remove_export_snapshot.
3.14 - Adds enable_replication, disable_replication,
       failover_replication, and list_replication_targets.
3.15 - Add revert_to_snapshot method
3.16 - Add no_snapshots to accept_transfer method
3.17 - Make get_backup_device a cast (async)
3.18 - Add reimage method
3.19 - Add extend_volume_completion method
```

BINARY = 'cinder-volume'

RPC_API_VERSION = '3.19'

`RPC_DEFAULT_VERSION = '3.0'`

`TOPIC = 'cinder-volume'`

accept_transfer(*ctxt*, *volume*, *new_user*, *new_project*, *no_snapshots=False*)

attach_volume(*ctxt*, *volume*, *instance_uuid*, *host_name*, *mountpoint*, *mode*)

attachment_delete(*ctxt*, *attachment_id*, *vref*)

attachment_update(*ctxt*, *vref*, *connector*, *attachment_id*)

copy_volume_to_image(*ctxt*, *volume*, *image_meta*)

create_group(*ctxt: RequestContext*, *group: Group*) → None

create_group_from_src(*ctxt*, *group*, *group_snapshot=None*, *source_group=None*)

create_group_snapshot(*ctxt*, *group_snapshot*)

create_snapshot(*ctxt: RequestContext*, *volume: Volume*, *snapshot: Snapshot*) → None

create_volume(*ctxt: RequestContext*, *volume: Volume*, *request_spec: dict | None*, *filter_properties: dict | None*, *allow_reschedule: bool = True*) → None

delete_group(*ctxt*, *group*)

delete_group_snapshot(*ctxt*, *group_snapshot*)

delete_snapshot(*ctxt*, *snapshot*, *unmanage_only=False*)

delete_volume(*ctxt: RequestContext*, *volume: Volume*, *unmanage_only: bool = False*, *cascade: bool = False*) → None

detach_volume(*ctxt*, *volume*, *attachment_id*)

disable_replication(*ctxt*, *group*)

do_cleanup(*ctxt*, *cleanup_request*)

Perform this service/cluster resource cleanup as requested.

enable_replication(*ctxt*, *group*)

extend_volume(*ctxt*, *volume*, *new_size*, *reservations*)

extend_volume_completion(*ctxt*, *volume*, *new_size*, *reservations*, *error*)

failover(*ctxt*, *service*, *secondary_backend_id=None*)

Failover host to the specified backend_id (secondary).

failover_completed(*ctxt*, *service*, *updates*)

Complete failover on all services of the cluster.

failover_replication(*ctxt*, *group*, *allow_attached_volume=False*, *secondary_backend_id=None*)

freeze_host(*ctxt*, *service*)

Set backend host to frozen.

get_backup_device(*ctxt*, *backup*, *volume*)

get_capabilities(*ctxt*, *backend_id*, *discover*)

get_log_levels(*context*, *service*, *log_request*)

get_manageable_snapshots(*ctxt*, *service*, *marker*, *limit*, *offset*, *sort_keys*, *sort_dirs*)

get_manageable_volumes(*ctxt*, *service*, *marker*, *limit*, *offset*, *sort_keys*, *sort_dirs*)

initialize_connection(*ctxt*, *volume*, *connector*)

initialize_connection_snapshot(*ctxt*, *snapshot*, *connector*)

list_replication_targets(*ctxt*, *group*)

manage_existing(*ctxt*, *volume*, *ref*)

manage_existing_snapshot(*ctxt*, *snapshot*, *ref*, *backend*)

migrate_volume(*ctxt*, *volume*, *dest_backend*, *force_host_copy*)

migrate_volume_completion(*ctxt*, *volume*, *new_volume*, *error*)

publish_service_capabilities(*ctxt*)

reimage(*ctxt*, *volume*, *image_meta*)

remove_export(*ctxt*, *volume*, *sync=False*)

remove_export_snapshot(*ctxt*, *snapshot*, *sync=False*)

retype(*ctxt*, *volume*, *new_type_id*, *dest_backend*, *migration_policy='never'*, *reservations=None*, *old_reservations=None*)

revert_to_snapshot(*ctxt*, *volume*, *snapshot*)

secure_file_operations_enabled(*ctxt*, *volume*)

set_log_levels(*context*, *service*, *log_request*)

terminate_connection(*ctxt*, *volume*, *connector*, *force=False*)

terminate_connection_snapshot(*ctxt*, *snapshot*, *connector*, *force=False*)

thaw_host(*ctxt*, *service*)

Clear the frozen setting on a backend host.

update_group(*ctxt*, *group*, *add_volumes=None*, *remove_volumes=None*)

update_migrated_volume(*ctxt*, *volume*, *new_volume*, *original_volume_status*)

cinder.volume.throttling module

Volume copy throttling helpers.

class BlkioCgroup(*bps_limit*, *cgroup_name*)

Bases: *Throttle*

Throttle disk I/O bandwidth using blkio cgroups.

subcommand(*srcpath*, *dstpath*)

Sub-command that reads from srcpath and writes to dstpath.

Throttle disk I/O bandwidth used by a sub-command, such as dd, that reads from srcpath and writes to dstpath. The sub-command must be executed with the generated prefix command.

class Throttle(*prefix=None*)

Bases: `object`

Base class for throttling disk I/O bandwidth

DEFAULT = None

static get_default()

static set_default(*throttle*)

subcommand(*srcpath*, *dstpath*)

Sub-command that reads from srcpath and writes to dstpath.

Throttle disk I/O bandwidth used by a sub-command, such as dd, that reads from srcpath and writes to dstpath. The sub-command must be executed with the generated prefix command.

cinder.volume.volume_migration module

class VolumeMigration(*id*, *user_id*, *encryption_key_id*)

Bases: `object`

Lightweight Volume Migration object.

Will be used by KeyMigrator instead of regular Volume object to avoid extra memory usage.

static from_volume(*volume*, *context*)

save()

class VolumeMigrationList

Bases: `list`

append(*volumes*, *context*)

Append object to the end of the list.

cinder.volume.volume_types module

Built-in volume type properties.

add_volume_type_access(*context: RequestContext*, *volume_type_id: str | None*, *project_id: str*) → None

Add access to volume type for project_id.

create(*context: RequestContext, name: str, extra_specs: dict[str, Any] | None = None, is_public: bool = True, projects: list[str] | None = None, description: str | None = None*)

Creates volume types.

destroy(*context: RequestContext, id: str*) → dict[str, Any]

Marks volume types as deleted.

There must exist at least one volume type (i.e. the default type) in the deployment. This method achieves that by ensuring: 1) the default_volume_type is set and is a valid one 2) the type requested to delete isnt the default type

Raises

VolumeTypeDefaultDeletionError when the type requested to delete is the default type

get_all_types(*context: RequestContext, inactive: int = 0, filters: dict | None = None, marker: dict[str, Any] | None = None, limit: int | None = None, sort_keys: list[str] | None = None, sort_dirs: list[str] | None = None, offset: int | None = None, list_result: bool = False*) → dict[str, Any] | list

Get all non-deleted volume_types.

Pass true as argument if you want deleted volume types returned also.

get_by_name_or_id(*context: RequestContext, identity: str*) → dict[str, Any]

Retrieves volume type by id or name

get_default_volume_type(*contxt: RequestContext | None = None*) → dict[str, Any]

Get the default volume type.

Raises

VolumeTypeDefaultMisconfiguredError when the configured default is not found

get_volume_type(*ctxt: RequestContext | None, id: str | None, expected_fields: Iterable[str] | None = None*) → dict[str, Any]

Retrieves single volume type by id.

get_volume_type_by_name(*context: RequestContext, name: str | None*) → dict[str, Any]

Retrieves single volume type by name.

get_volume_type_encryption(*context: RequestContext, volume_type_id: str | None*) → dict | None

get_volume_type_extra_specs(*volume_type_id: str*) → dict

get_volume_type_qos_specs(*volume_type_id: str*) → dict[str, Any]

Get all qos specs for given volume type.

is_encrypted(*context: RequestContext, volume_type_id: str | None*) → bool

is_public_volume_type(*context: RequestContext, volume_type_id: str*) → bool

Return is_public boolean value of volume type

notify_about_volume_type_access_usage(*context: RequestContext, volume_type_id: str, project_id: str, event_suffix: str, host: str | None = None*) → None

Notify about successful usage type-access-(add/remove) command.

Parameters

- **context** security context
- **volume_type_id** volume type uuid
- **project_id** tenant uuid
- **event_suffix** name of called operation access-(add/remove)
- **host** hostname

provision_filter_on_size(*context: RequestContext*, *volume_type: dict[str, Any] | None*, *size: str | int*) → None

This function filters volume provisioning requests on size limits.

If a volume type has provisioning size min/max set, this filter will ensure that the volume size requested is within the size limits specified in the volume type.

remove_volume_type_access(*context: RequestContext*, *volume_type_id: str | None*, *project_id: str*) → None

Remove access to volume type for project_id.

update(*context: RequestContext*, *id: str | None*, *name: str | None*, *description: str | None*, *is_public: bool | None = None*) → None

Update volume type by id.

volume_types_diff(*context: RequestContext*, *vol_type_id1: str*, *vol_type_id2: str*) → tuple[dict[str, Any], bool]

Returns a diff of two volume types and whether they are equal.

Returns a tuple of (diff, equal), where equal is a boolean indicating whether there is any difference, and diff is a dictionary with the following format:

```
{
    'extra_specs': {'key1': (value_in_1st_vol_type,
                             value_in_2nd_vol_type),
                    'key2': (value_in_1st_vol_type,
                             value_in_2nd_vol_type),
                    {...}}
    'qos_specs': {'key1': (value_in_1st_vol_type,
                           value_in_2nd_vol_type),
                  'key2': (value_in_1st_vol_type,
                           value_in_2nd_vol_type),
                  {...}}
    'encryption': {'cipher': (value_in_1st_vol_type,
                              value_in_2nd_vol_type),
                  {'key_size': (value_in_1st_vol_type,
                                value_in_2nd_vol_type),
                   {...}}
}
```

volume_types_encryption_changed(*context: RequestContext*, *vol_type_id1: str | None*, *vol_type_id2: str | None*) → bool

Return whether encryptions of two volume types are same.

cinder.volume.volume_utils module

Volume-related Utilities and helpers.

class TraceWrapperMetaclass(*classname*, *bases*, *classDict*)

Bases: `type`

Metaclass that wraps all methods of a class with trace_method.

This metaclass will cause every function inside of the class to be decorated with the trace_method decorator.

To use the metaclass you define a class like so: class MyClass(object, metaclass=utils.TraceWrapperMetaclass):

class TraceWrapperWithABCMetaclass(*name*, *bases*, *namespace*, ***kwargs*)

Bases: `ABCMeta`, `TraceWrapperMetaclass`

Metaclass that wraps all methods of a class with trace.

append_host(*host: str | None*, *pool: str | None*) → str | None

Encode pool into host info.

brick_attach_volume_encryptor(*context: RequestContext*, *attach_info: dict*, *encryption: dict*) → None

Attach encryption layer.

brick_detach_volume_encryptor(*attach_info: dict*, *encryption: dict*) → None

Detach encryption layer.

brick_get_connector(*protocol: str*, *driver=None*, *use_multipath: bool = False*, *device_scan_attempts: int = 3*, **args*, ***kwargs*)

Wrapper to get a brick connector object.

This automatically populates the required protocol as well as the root_helper needed to execute commands.

brick_get_connector_properties(*multipath: bool = False*, *enforce_multipath: bool = False*)

Wrapper to automatically set root_helper in brick calls.

Parameters

- **multipath** A boolean indicating whether the connector can support multipath.
- **enforce_multipath** If True, it raises exception when multipath=True is specified but multipathd is not running. If False, it falls back to multipath=False when multipathd is not running.

brick_get_encryptor(*connection_info: dict*, **args*, ***kwargs*)

Wrapper to get a brick encryptor object.

`check_already_managed_volume`(*vol_id: str | None*)

Check cinder db for already managed volume.

Parameters
`vol_id` volume id parameter

Returns
bool return True, if db entry with specified volume id exists, otherwise return False

Raises
ValueError if vol_id is not a valid uuid string

`check_encryption_provider`(*volume:* Volume, *context:* RequestContext) → dict

Check that this is a LUKS encryption provider.

Returns
encryption dict

`check_for_odirect_support`(*src: str*, *dest: str*, *flag: str = 'oflag=direct'*) → bool

`check_image_metadata`(*image_meta: dict[str, str | int]*, *vol_size: int*) → None

Validates the image metadata.

`clear_volume`(*volume_size: int*, *volume_path: str*, *volume_clear: str | None = None*, *volume_clear_size: int | None = None*, *volume_clear_ionice: str | None = None*, *throttle=None*) → None

Unprovision old volumes to prevent data leaking between users.

`clone_encryption_key`(*context:* RequestContext, *key_manager*, *encryption_key_id: str*) → str

`convert_config_string_to_dict`(*config_string: str*) → dict

Convert config file replication string to a dict.

The only supported form is as follows: {key-1=val-1 key-2=val-2}

Parameters
`config_string` Properly formatted string to convert to dict.

Response
dict of string values

`copy_image_to_volume`(*driver*, *context:* RequestContext, *volume:* Volume, *image_meta: dict*, *image_location: str | tuple[str | None, Any]*, *image_service*, *disable_sparse: bool = False*) → None

Downloads Glance image to the specified volume.

`copy_volume`(*src: str | ~typing.BinaryIO*, *dest: str | ~typing.BinaryIO*, *size_in_m: int*, *blocksize: str | int*, *sync=False*, *execute=<function execute>*, *ionice=None*, *throttle=None*, *sparse=False*) → None

Copy data from the source volume to the destination volume.

The parameters src and dest are both typically of type str, which represents the path to each volume on the filesystem. Connectors can optionally return a volume handle of type RawIOBase for volumes that are not available on the local filesystem for open/close operations.

If either src or dest are not of type str, then they are assumed to be of type RawIOBase or any derivative that supports file operations such as read and write. In this case, the handles are treated as file handles instead of file paths and, at present moment, throttling is unavailable.

create_encryption_key(*context: RequestContext*, *key_manager*, *volume_type_id: str*) → str | None

delete_encryption_key(*context: RequestContext*, *key_manager*, *encryption_key_id: str*) → None

enable_bootable_flag(*volume: Volume*) → None

extract_availability_zones_from_volume_type(*volume_type: VolumeType | dict*) → list[str] | None

extract_host(*host: str | None*, *level: str = 'backend'*, *default_pool_name: bool = False*) → str | None

Extract Host, Backend or Pool information from host string.

Parameters

- **host** String for host, which could include host@backend#pool info
- **level** Indicate which level of information should be extracted from host string. Level can be host, backend or pool, default value is backend
- **default_pool_name** this flag specify what to do if level == pool and there is no pool info encoded in host string. default_pool_name=True will return DEFAULT_POOL_NAME, otherwise we return None. Default value of this parameter is False.

Returns

expected information, string or None

Raises

exception.InvalidVolume

For example:

host = HostA@BackendB#PoolC ret = extract_host(host, host) # ret is HostA ret = extract_host(host, backend) # ret is HostA@BackendB ret = extract_host(host, pool) # ret is PoolC

host = HostX@BackendY ret = extract_host(host, pool) # ret is None ret = extract_host(host, pool, True) # ret is _pool0

extract_id_from_snapshot_name(*snap_name: str*) → str | None

Return a snapshots ID from its name on the backend.

extract_id_from_volume_name(*vol_name: str*) → str | None

generate_password(*length: int = 16*, *symbolgroups: tuple[str, ...] = ('23456789', 'ABCDEFGHJKLMNPQRSTUVWXYZ', 'abcdefghijkmnopqrstuvwxyz')*) → str

Generate a random password from the supplied symbol groups.

At least one symbol from each group will be included. Unpredictable results if length is less than the number of symbol groups.

Believed to be reasonably secure (with a reasonable password length!)

generate_username(*length: int = 20*, *symbolgroups: tuple[str, ...] = ('23456789', 'ABCDEFGHJKLMNPQRSTUVWXYZ', 'abcdefghijkmnopqrstuvwxyz')*) → str

get_all_physical_volumes(*vg_name=None*) → list

get_all_volume_groups(*vg_name=None*) → list

get_backend_configuration(*backend_name*, *backend_opts=None*)

Get a configuration object for a specific backend.

get_base_image_ref(*volume: Volume*)

get_max_over_subscription_ratio(*str_value: str | float*, *supports_auto: bool = False*) → str | float

Get the max_over_subscription_ratio from a string

As some drivers need to do some calculations with the value and we are now receiving a string value in the conf, this converts the value to float when appropriate.

Parameters

- **str_value** Configuration object
- **supports_auto** Tell if the calling driver supports auto MOSR.

Response
value of mosr

get_volume_image_metadata(*image_id: str*, *image_meta: dict[str, Any]*) → dict

group_get_by_id(*group_id*)

hosts_are_equivalent(*host_1: str*, *host_2: str*) → bool

image_conversion_dir() → str

is_all_zero(*chunk: bytes*) → bool

Return true if the chunk of bytes is all zeroes.

is_boolean_str(*str: str | None*) → bool

is_group_a_cg_snapshot_type(*group_or_snap*) → bool

is_group_a_type(*group: Group*, *key: str*) → bool

is_multiattach_spec(*extra_specs: dict*) → bool

is_replicated_spec(*extra_specs: dict*) → bool

log_unsupported_driver_warning(*driver*)

Annoy the log about unsupported drivers.

matching_backend_name(*src_volume_type*, *volume_type*) → bool

notify_about_backup_usage(*context: RequestContext*, *backup: Backup*, *event_suffix: str*, *extra_usage_info: dict | None = None*, *host: str | None = None*) → None

notify_about_capacity_usage(*context: RequestContext*, *capacity: dict*, *suffix: str*, *extra_usage_info: dict | None = None*, *host: str | None = None*) → None

notify_about_cgsnapshot_usage(*context: RequestContext, cgsnapshot: CGSnapshot, event_suffix: str, extra_usage_info: dict | None = None, host: str | None = None*) → None

notify_about_consistencygroup_usage(*context: RequestContext, group: Group, event_suffix: str, extra_usage_info: dict | None = None, host: str | None = None*) → None

notify_about_group_snapshot_usage(*context: RequestContext, group_snapshot: GroupSnapshot, event_suffix: str, extra_usage_info=None, host: str | None = None*) → None

notify_about_group_usage(*context: RequestContext, group: Group, event_suffix: str, extra_usage_info: dict | None = None, host: str | None = None*) → None

notify_about_snapshot_usage(*context: RequestContext, snapshot: Snapshot, event_suffix: str, extra_usage_info: dict | None = None, host: str | None = None*) → None

notify_about_volume_usage(*context: RequestContext, volume: Volume, event_suffix: str, extra_usage_info: dict | None = None, host: str | None = None*) → None

null_safe_str(*s: str | None*) → str

paginate_entries_list(*entries: list[dict], marker: dict | str | None, limit: int, offset: int | None, sort_keys: list[str], sort_dirs: list[str]*) → list

Paginate a list of entries.

Parameters
: **entries** list of dictionaries

Marker
: The last element previously returned

Limit
: The maximum number of items to return

Offset
: The number of items to skip from the marker or from the first element.

Sort_keys
: A list of keys in the dictionaries to sort by

Sort_dirs
: A list of sort directions, where each is either asc or dec

require_driver_initialized(*driver*)

Verifies if *driver* is initialized

If the driver is not initialized, an exception will be raised.

Params driver
: The driver instance.

Raises
: *exception.DriverNotInitialized*

resolve_hostname(*hostname: str*) → str

Resolves host name to IP address.

Resolves a host name (my.data.point.com) to an IP address (10.12.143.11). This routine also works if the data passed in hostname is already an IP. In this case, the same IP address will be returned.

Parameters
hostname Host name to resolve.

Returns
IP Address for Host name.

sanitize_host(*host: str*) → str

Ensure IPv6 addresses are enclosed in [] for iSCSI portals.

sanitize_hostname(*hostname*) → str

Return a hostname which conforms to RFC-952 and RFC-1123 specs.

setup_tracing(*trace_flags*)

Set global variables for each trace flag.

Sets variables TRACE_METHOD and TRACE_API, which represent whether to log methods or api traces.

Parameters
trace_flags a list of strings

supports_thin_provisioning() → bool

trace(**dec_args*, ***dec_kwargs*)

Trace calls to the decorated function.

This decorator should always be defined as the outermost decorator so it is defined last. This is important so it does not interfere with other decorators.

Using this decorator on a function will cause its execution to be logged at *DEBUG* level with arguments, return values, and exceptions.

Returns
a function decorator

trace_api(**dec_args*, ***dec_kwargs*)

Decorates a function if TRACE_API is true.

trace_method(*f*)

Decorates a function if TRACE_METHOD is true.

update_backup_error(*backup*, *err: str*, *status='error'*) → None

upload_volume(*context: RequestContext*, *image_service*, *image_meta*, *volume_path*, *volume: Volume*, *volume_format: str = 'raw'*, *run_as_root: bool = True*, *compress: bool = True*, *volume_fd=None*) → None

Module contents

API(**args*, ***kwargs*)

cinder.wsgi package

Submodules

cinder.wsgi.common module

Utility methods for working with WSGI servers.

class Application

Bases: `object`

Base WSGI application wrapper. Subclasses need to implement __call__.

classmethod factory(*global_config*, ***local_config*)

Used for paste app factories in paste.deploy config files.

Any local configuration (that is, values under the [app:APPNAME] section of the paste config) will be passed into the *__init__* method as kwargs.

A hypothetical configuration would look like:

> [app:wadl] latest_version = 1.3 paste.app_factory = cinder.api.fancy_api:Wadl.factory

which would result in a call to the *Wadl* class as

> import cinder.api.fancy_api fancy_api.Wadl(latest_version=1.3)

You could of course re-implement the *factory* method in subclasses, but using the kwarg passing it shouldnt be necessary.

class Middleware(*application*)

Bases: `Application`

Base WSGI middleware.

These classes require an application to be initialized that will be called next. By default the middleware will simply call its wrapped app, or you can override __call__ to customize its behavior.

classmethod factory(*global_config*, ***local_config*)

Used for paste app factories in paste.deploy config files.

Any local configuration (that is, values under the [filter:APPNAME] section of the paste config) will be passed into the *__init__* method as kwargs.

A hypothetical configuration would look like:

> [filter:analytics] redis_host = 127.0.0.1 paste.filter_factory = cinder.api.analytics:Analytics.factory

which would result in a call to the *Analytics* class as

> import cinder.api.analytics analytics.Analytics(app_from_paste, redis_host=127.0.0.1)

You could of course re-implement the *factory* method in subclasses, but using the kwarg passing it shouldnt be necessary.

process_request(*req*)

Called on each request.

If this returns None, the next application down the stack will be executed. If it returns a response then that response will be returned and execution will stop here.

process_response(*response*)

Do whatever youd like to the response.

class Request(*environ*, *charset=None*, *unicode_errors=None*, *decode_param_names=None*, ***kw*)

Bases: `Request`

cinder.wsgi.eventlet_server module

Methods for working with eventlet WSGI servers.

class Server(*conf*, *name*, *app*, *host='0.0.0.0'*, *port=0*, *pool_size=None*, *protocol=<class 'eventlet.wsgi.HttpProtocol'>*, *backlog=128*, *use_ssl=False*, *max_url_len=None*, *logger_name='eventlet.wsgi.server'*, *socket_family=None*, *socket_file=None*, *socket_mode=None*)

Bases: `Server`

Server class to manage a WSGI server, serving a WSGI application.

cinder.wsgi.wsgi module

Cinder OS API WSGI application.

initialize_application()

Module contents

cinder.zonemanager package

Submodules

cinder.zonemanager.fc_common module

class FCCommon(***kwargs*)

Bases: `object`

Common interface for FC operations.

VERSION = '1.0'

get_version()

cinder.zonemanager.fc_san_lookup_service module

Base Lookup Service for name server lookup to find the initiator to target port mapping for available SAN contexts. Vendor specific lookup classes are expected to implement the interfaces defined in this class.

class FCSanLookupService(***kwargs*)

Bases: *FCCommon*

Base Lookup Service.

Base Lookup Service for name server lookup to find the initiator to target port mapping for available SAN contexts.

get_device_mapping_from_network(*initiator_list*, *target_list*)

Get device mapping from FC network.

Gets a filtered list of initiator ports and target ports for each SAN available. :param initiator_list: list of initiator port WWN :param target_list: list of target port WWN :returns: device wwn map in following format

```
{
    <San name>: {
        'initiator_port_wwn_list':
        ('200000051E55A100', '200000051E55A121'..)
        'target_port_wwn_list':
        ('100000051E55A100', '100000051E55A121'..)
    }
}
```

Raises

Exception when a lookup service implementation is not specified in cinder.conf:fc_san_lookup_service

lookup_service = None

cinder.zonemanager.fc_zone_manager module

ZoneManager is responsible to manage access control using FC zoning when zoning mode is set as fabric. ZoneManager provides interfaces to add connection and remove connection for given initiator and target list associated with a FC volume attach and detach operation.

Related Flags

zone_driver

Used by:class:*ZoneManager*. Defaults to *cinder.zonemanager.drivers.brocade.brcd_fc_zone_driver.BrcdFCZoneDriver*

zoning_policy

Used by: class: ZoneManager. Defaults to none

class ZoneManager(**args*, ***kwargs*)

Bases: *FCCommon*

Manages Connection control during attach/detach.

Version History:

1.0 - Initial version 1.0.1 - Added __new__ for singleton 1.0.2 - Added friendly zone name

VERSION = '1.0.2'

add_connection(*conn_info*)

Add connection control.

Adds connection control for the given initiator target map. initiator_target_map - each initiator WWN mapped to a list of one or more target WWN:

```
e.g.:
{
    '10008c7cff523b01': ['20240002ac000a50', '20240002ac000a40']
}
```

delete_connection(*conn_info*)

Delete connection.

Updates/deletes connection control for the given initiator target map. initiator_target_map - each initiator WWN mapped to a list of one or more target WWN:

```
e.g.:
{
    '10008c7cff523b01': ['20240002ac000a50', '20240002ac000a40']
}
```

driver = None

fabric_names = []

get_san_context(*target_wwn_list*)

SAN lookup for end devices.

Look up each SAN configured and return a map of SAN (fabric IP) to list of target WWNs visible to the fabric.

get_valid_initiator_target_map(*initiator_target_map*, *add_control*)

Reference count check for end devices.

Looks up the reference count for each initiator-target pair from the map and returns a filtered list based on the operation type add_control - operation type can be true for add connection control and false for remove connection control

get_zoning_state_ref_count(*initiator_wwn*, *target_wwn*)

Zone management state check.

Performs state check for given I-T pair to return the current count of active attach for the pair.

property initialized

set_initialized(*value=True*)

cinder.zonemanager.fczm_constants module

Common constants used by FC Zone Manager.

cinder.zonemanager.utils module

Utility functions related to the Zone Manager.

add_fc_zone(*connection_info*)

Utility function to add a FC Zone.

create_lookup_service()

create_zone_manager()

If zoning is enabled, build the Zone Manager.

get_formatted_wwn(*wwn_str*)

Utility API that formats WWN to insert :.

remove_fc_zone(*connection_info*)

Utility function for FC drivers to remove zone.

Module contents

Submodules

cinder.context module

RequestContext: context for requests that persist through all of cinder.

class RequestContext(*user_id: str | None = None*, *project_id: str | None = None*, *is_admin: bool | None = None*, *read_deleted: str | None = 'no'*, *project_name: str | None = None*, *remote_address: str | None = None*, *timestamp=None*, *quota_class=None*, *service_catalog: dict | None = None*, *user_auth_plugin=None*, *message_resource_id=None*, *message_resource_type=None*, *message_action=None*, ***kwargs*)

Bases: `RequestContext`

Security context and request information.

Represents the user taking a given action within the system.

authorize(*action: str*, *target: dict | None = None*, *target_obj: dict | None = None*, *fatal: bool = True*)

Verify that the given action is valid on the target in this context.

Parameters

- **action** string representing the action to be checked.
- **target** dictionary representing the object of the action for object creation this should be a dictionary representing the location of the object e.g. `{'project_id': context.project_id}`. If None, then this default target will be considered: {project_id: self.project_id, user_id: self.user_id}
- **target_obj** dictionary representing the object which will be used to update target.
- **fatal** if False, will return False when an exception.PolicyNotAuthorized occurs.

Raises
`cinder.exception.NotAuthorized` if verification fails and fatal is True.

Returns
returns a non-False value (not necessarily True) if authorized and False if not authorized and fatal is False.

property connection

deepcopy() → *RequestContext*

elevated(*read_deleted: str | None = None*, *overwrite: bool = False*) → *RequestContext*

Return a version of this context with admin flag set.

classmethod from_dict(*values: dict*) → *RequestContext*

Construct a context object from a provided dictionary.

get_auth_plugin()

property read_deleted: str

property session

to_dict() → dict[str, Any]

Return a dictionary of context attributes.

to_policy_values() → dict

A dictionary of context attributes to enforce policy with.

oslo.policy enforcement requires a dictionary of attributes representing the current logged in user on which it applies policy enforcement. This dictionary defines a standard list of attributes that should be available for enforcement across services.

It is expected that services will often have to override this method with either deprecated values or additional attributes used by that service specific policy.

property transaction

property transaction_ctx

get_admin_context(*read_deleted: str | None = 'no'*) → *RequestContext*

get_internal_tenant_context() → *RequestContext* | None

Build and return the Cinder internal tenant context object

This request context will only work for internal Cinder operations. It will not be able to make requests to remote services. To do so it will need to use the keystone client to get an auth_token.

cinder.coordination module

Coordination and locking utilities.

class Coordinator(*agent_id: str | None = None*, *prefix: str = ''*)

Bases: `object`

Tooz coordination wrapper.

Coordination member id is created from concatenated *prefix* and *agent_id* parameters.

Parameters

- **agent_id** (*str*) Agent identifier
- **prefix** (*str*) Used to provide member identifier with a meaningful prefix.

get_lock(*name: str*)

Return a Tooz backend lock.

Parameters

name (*str*) The lock name that is used to identify it across all nodes.

remove_lock(*glob_name*)

start() → None

stop() → None

Disconnect from coordination backend and stop heartbeat.

synchronized(**lock_names: str*, *blocking: bool = True*, *coordinator: ~cinder.coordination.Coordinator = <cinder.coordination.Coordinator object>*) → Callable

Synchronization decorator.

Parameters

- **lock_names** (*str*) Arbitrary number of Lock names.
- **blocking** If True, blocks until the lock is acquired. If False, raises exception when not acquired. Otherwise, the value is used as a timeout value and if lock is not acquired after this number of seconds exception is raised. This is a keyword only argument.
- **coordinator** Coordinator class to use when creating lock. Defaults to the global coordinator. This is a keyword only argument.

Raises

tooz.coordination.LockAcquireFailed if lock is not acquired

Decorating a method like so:

```
@synchronized('mylock')
def foo(self, *args):
    ...
```

ensures that only one process will execute the foo method at a time.

Different methods can share the same lock:

```
@synchronized('mylock')
def foo(self, *args):
    ...

@synchronized('mylock')
def bar(self, *args):
    ...
```

This way only one of either foo or bar can be executing at a time.

Lock name can be formatted using Python format string syntax:

```
@synchronized('{f_name}-{vol.id}-{snap[name]}')
def foo(self, vol, snap):
    ...
```

Multiple locks can be requested simultaneously and the decorator will reorder the names by rendered lock names to prevent potential deadlocks.

@synchronized({f_name}-{vol.id}-{snap[name]},
{f_name}-{vol.id}.delete)

def foo(self, vol, snap):

Available field names are: decorated function parameters and *f_name* as a decorated function name.

synchronized_remove(*glob_name*, *coordinator=<cinder.coordination.Coordinator object>*)

cinder.exception module

Cinder base exception handling.

Includes decorator for re-raising Cinder-type exceptions.

SHOULD include dedicated exception logging.

exception APIException(*message=None*, ***kwargs*)

Bases: *CinderException*

message = 'Error while requesting %(service)s API.'

exception APITimeout(*message=None*, ***kwargs*)

Bases: *APIException*

message = 'Timeout while requesting %(service)s API.'

exception AdminRequired(*message: str | tuple | None = None*, ***kwargs*)

Bases: *NotAuthorized*

message = 'User does not have admin privileges'

exception AttachmentSpecsNotFound(*message: str | tuple | None = None*, ***kwargs*)

Bases: *NotFound*

message = 'Attachment %(attachment_id)s has no key %(specs_key)s.'

exception BackupDriverException(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = 'Backup driver reported an error: %(reason)s'

exception BackupInvalidCephArgs(*message: str | tuple | None = None*, ***kwargs*)

Bases: *BackupDriverException*

message = 'Invalid Ceph args provided for backup rbd operation'

exception BackupLimitExceeded(*message: str | tuple | None = None*, ***kwargs*)

Bases: *QuotaError*

message = 'Maximum number of backups allowed (%(allowed)d) exceeded'

exception BackupMetadataNotFound(*message: str | tuple | None = None*, ***kwargs*)

Bases: *NotFound*

message = 'Backup %(backup_id)s has no metadata with key %(metadata_key)s.'

exception BackupMetadataUnsupportedVersion(*message: str | tuple | None = None*, ***kwargs*)

Bases: *BackupDriverException*

message = 'Unsupported backup metadata version requested'

exception BackupNotFound(*message: str | tuple | None = None*, ***kwargs*)

Bases: *NotFound*

message = 'Backup %(backup_id)s could not be found.'

exception BackupOperationError(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Invalid*

message = 'An error has occurred during backup operation'

exception BackupRBDOperationFailed(*message: str | tuple | None = None*, ***kwargs*)

Bases: *BackupDriverException*

message = 'Backup RBD operation failed'

exception BackupRestoreCancel(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = 'Canceled backup %(back_id)s restore on volume %(vol_id)s'

exception BadHTTPResponseStatus(*message: str | tuple | None = None*, ***kwargs*)

Bases: *VolumeDriverException*

message = 'Bad HTTP response status %(status)s'

exception BadResetResourceStatus(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = 'Bad reset resource status : %(reason)s'

exception CappedVersionUnknown(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = "Unrecoverable Error: Versioned Objects in DB are capped to unknown version %(version)s. Most likely your environment contains only new services and you're trying to start an older one. Use `cinder-manage service list` to check that and upgrade this service."

exception CgSnapshotNotFound(*message: str | tuple | None = None*, ***kwargs*)

Bases: *NotFound*

message = 'CgSnapshot %(cgsnapshot_id)s could not be found.'

exception CinderAcceleratorError(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = 'Cinder accelerator %(accelerator)s encountered an error while compressing/decompressing image.\nCommand %(cmd)s execution failed.\n%(description)s\nReason: %(reason)s'

exception CinderException(*message: str | tuple | None = None*, ***kwargs*)

Bases: Exception

Base Cinder Exception

To correctly use this class, inherit from it and define a message property. That message will get printfd with the keyword arguments provided to the constructor.

code = 500

headers: dict = {}

message = 'An unknown exception occurred.'

safe = False

exception CleanableInUse(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Invalid*

message = '%(type)s with id %(id)s is already being cleaned up or another host has taken over it.'

exception ClusterExists(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Duplicate*

message = 'Cluster %(name)s already exists.'

exception ClusterHasHosts(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Invalid*

message = 'Cluster %(id)s still has hosts.'

exception ClusterNotFound(*message: str | tuple | None = None*, ***kwargs*)

Bases: *NotFound*

message = 'Cluster %(id)s could not be found.'

exception ConfigNotFound(*message: str | tuple | None = None*, ***kwargs*)

Bases: *NotFound*

message = 'Could not find config at %(path)s'

exception ConflictNovaUsingAttachment(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

code = 409

message = 'Detach volume from instance %(instance_id)s using the Compute API'

safe = True

exception ConsistencyGroupNotFound(*message: str | tuple | None = None*, ***kwargs*)

Bases: *NotFound*

message = 'ConsistencyGroup %(consistencygroup_id)s could not be found.'

exception ConvertedException(*code: int = 500*, *title: str = ''*, *explanation: str = ''*)

Bases: WSGIHTTPException

exception DeviceUnavailable(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Invalid*

message = 'The device in the path %(path)s is unavailable: %(reason)s'

exception DriverInitiatorDataExists(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Duplicate*

message = "Driver initiator data for initiator '%(initiator)s' and backend '%(namespace)s' with key '%(key)s' already exists."

exception DriverNotInitialized(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = 'Volume driver not ready.'

exception Duplicate(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

exception EncryptedBackupOperationFailed(*message: str | tuple | None = None*, ***kwargs*)

Bases: *BackupDriverException*

message = 'Backup operation of an encrypted volume failed.'

exception EvaluatorParseException

Bases: Exception

message = 'Error during evaluator parsing: %(reason)s'

exception ExportFailure(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Invalid*

message = 'Failed to export for volume: %(reason)s'

exception ExtendVolumeError(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = 'Error extending volume: %(reason)s'

exception FCSanLookupServiceException(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = 'Fibre Channel SAN Lookup failure: %(reason)s'

exception FCZoneDriverException(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = 'Fibre Channel Zone operation failed: %(reason)s'

exception FailedCmdWithDump(*message: str | tuple | None = None*, ***kwargs*)

Bases: *VolumeDriverException*

message = 'Operation failed with status=%(status)s. Full dump: %(data)s'

exception FileNotFound(*message: str | tuple | None = None*, ***kwargs*)

Bases: *NotFound*

message = 'File %(file_path)s could not be found.'

exception GlanceConnectionFailed(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = 'Connection to glance failed: %(reason)s'

exception GlanceMetadataExists(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Invalid*

message = 'Glance metadata cannot be updated, key %(key)s exists for volume id %(volume_id)s'

exception GlanceMetadataNotFound(*message: str | tuple | None = None*, ***kwargs*)

Bases: *NotFound*

message = 'Glance metadata for volume/snapshot %(id)s cannot be found.'

exception GlanceStoreNotFound(*message: str | tuple | None = None*, ***kwargs*)

Bases: *NotFound*

message = 'Store %(store_id)s not enabled in glance.'

exception GlanceStoreReadOnly(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Invalid*

message = 'Store %(store_id)s is read-only in glance.'

exception GroupLimitExceeded(*message: str | tuple | None = None*, ***kwargs*)

Bases: *QuotaError*

message = 'Maximum number of groups allowed (%(allowed)d) exceeded'

exception GroupNotFound(*message: str | tuple | None = None*, ***kwargs*)

Bases: *NotFound*

message = 'Group %(group_id)s could not be found.'

exception GroupSnapshotNotFound(*message: str | tuple | None = None*, ***kwargs*)

Bases: *NotFound*

message = 'GroupSnapshot %(group_snapshot_id)s could not be found.'

exception GroupTypeAccessExists(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Duplicate*

message = 'Group type access for %(group_type_id)s / %(project_id)s combination already exists.'

exception GroupTypeAccessNotFound(*message: str | tuple | None = None*, ***kwargs*)

Bases: *NotFound*

message = 'Group type access not found for %(group_type_id)s / %(project_id)s combination.'

exception GroupTypeCreateFailed(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = 'Cannot create group_type with name %(name)s and specs %(group_specs)s'

exception GroupTypeExists(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Duplicate*

message = 'Group Type %(id)s already exists.'

exception GroupTypeInUse(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = 'Group Type %(group_type_id)s deletion is not allowed with groups present with the type.'

exception GroupTypeNotFound(*message: str | tuple | None = None*, ***kwargs*)

Bases: *NotFound*

message = 'Group type %(group_type_id)s could not be found.'

exception GroupTypeNotFoundByName(*message: str | tuple | None = None*, ***kwargs*)

Bases: *GroupTypeNotFound*

message = 'Group type with name %(group_type_name)s could not be found.'

exception GroupTypeSpecsNotFound(*message: str | tuple | None = None*, ***kwargs*)

Bases: *NotFound*

message = 'Group Type %(group_type_id)s has no specs with key %(group_specs_key)s.'

exception GroupTypeUpdateFailed(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = 'Cannot update group_type %(id)s'

exception GroupVolumeTypeMappingExists(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Duplicate*

message = 'Group volume type mapping for %(group_id)s / %(volume_type_id)s combination already exists.'

exception HostNotFound(*message: str | tuple | None = None*, ***kwargs*)

Bases: *NotFound*

message = 'Host %(host)s could not be found.'

exception ISCSITargetAttachFailed(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = 'Failed to attach iSCSI target for volume %(volume_id)s.'

exception ISCSITargetCreateFailed(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = 'Failed to create iscsi target for volume %(volume_id)s.'

exception ISCSITargetDetachFailed(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = 'Failed to detach iSCSI target for volume %(volume_id)s.'

exception ISCSITargetHelperCommandFailed(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = '%(error_message)s'

exception ISCSITargetRemoveFailed(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = 'Failed to remove iscsi target for volume %(volume_id)s.'

exception ImageCompressionNotAllowed(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = 'Image compression upload disallowed, but container_format is compressed'

exception ImageConversionNotAllowed(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = 'Image Conversion disallowed for image %(image_id)s: %(reason)s'

exception ImageCopyFailure(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Invalid*

message = 'Failed to copy image to volume: %(reason)s'

exception ImageDownloadFailed(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = 'Failed to download image %(image_href)s, reason: %(reason)s'

exception ImageLimitExceeded(*message: str | tuple | None = None*, ***kwargs*)

Bases: *QuotaError*

message = 'Image quota exceeded'

exception ImageNotAuthorized(*message: str | tuple | None = None*, ***kwargs*)

Bases: `CinderException`

message = 'Not authorized for image %(image_id)s.'

exception ImageNotFound(*message: str | tuple | None = None*, ***kwargs*)

Bases: `NotFound`

message = 'Image %(image_id)s could not be found.'

exception ImageSignatureVerificationException(*message: str | tuple | None = None*, ***kwargs*)

Bases: `CinderException`

message = 'Failed to verify image signature, reason: %(reason)s.'

exception ImageTooBig(*message: str | tuple | None = None*, ***kwargs*)

Bases: `Invalid`

message = 'Image %(image_id)s size exceeded available disk space: %(reason)s'

exception ImageUnacceptable(*message: str | tuple | None = None*, ***kwargs*)

Bases: `Invalid`

message = 'Image %(image_id)s is unacceptable: %(reason)s'

exception Invalid(*message: str | tuple | None = None*, ***kwargs*)

Bases: `CinderException`

code = 400

message = 'Unacceptable parameters.'

exception InvalidAPIVersionString(*message: str | tuple | None = None*, ***kwargs*)

Bases: `Invalid`

message = 'API Version String %(version)s is of invalid format. Must be of format MajorNum.MinorNum.'

exception InvalidAuthKey(*message: str | tuple | None = None*, ***kwargs*)

Bases: `Invalid`

message = 'Invalid auth key: %(reason)s'

exception InvalidAvailabilityZone(*message: str | tuple | None = None*, ***kwargs*)

Bases: `Invalid`

message = "Availability zone '%(az)s' is invalid."

exception InvalidBackup(*message: str | tuple | None = None*, ***kwargs*)

Bases: `Invalid`

message = 'Invalid backup: %(reason)s'

exception InvalidCgSnapshot(*message: str | tuple | None = None*, ***kwargs*)

Bases: `Invalid`

message = 'Invalid CgSnapshot: %(reason)s'

exception InvalidConfigurationValue(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Invalid*

message = 'Value "%(value)s" is not valid for configuration option "%(option)s"'

exception InvalidConnectorException(*message: str | tuple | None = None*, ***kwargs*)

Bases: *VolumeDriverException*

message = "Connector doesn't have required information: %(missing)s"

exception InvalidConsistencyGroup(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Invalid*

message = 'Invalid ConsistencyGroup: %(reason)s'

exception InvalidContentType(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Invalid*

message = 'Invalid content type %(content_type)s.'

exception InvalidGlobalAPIVersion(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Invalid*

message = 'Version %(req_ver)s is not supported by the API. Minimum is %(min_ver)s and maximum is %(max_ver)s.'

exception InvalidGroup(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Invalid*

message = 'Invalid Group: %(reason)s'

exception InvalidGroupSnapshot(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Invalid*

message = 'Invalid GroupSnapshot: %(reason)s'

exception InvalidGroupSnapshotStatus(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Invalid*

message = 'Invalid GroupSnapshot Status: %(reason)s'

exception InvalidGroupStatus(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Invalid*

message = 'Invalid Group Status: %(reason)s'

exception InvalidGroupType(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Invalid*

message = 'Invalid group type: %(reason)s'

exception InvalidHost(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Invalid*

message = 'Invalid host: %(reason)s'

exception InvalidImageRef(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Invalid*

message = 'Invalid image href %(image_href)s.'

exception InvalidInput(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Invalid*

message = 'Invalid input received: %(reason)s'

exception InvalidMetadataType(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Invalid*

message = 'The type of metadata: %(metadata_type)s for volume/snapshot %(id)s is invalid.'

exception InvalidName(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Invalid*

message = "An invalid 'name' value was provided. %(reason)s"

exception InvalidParameterValue(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Invalid*

message = '%(err)s'

exception InvalidQoSSpecs(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Invalid*

message = 'Invalid qos specs: %(reason)s'

exception InvalidQuotaValue(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Invalid*

message = 'Change would make usage less than 0 for the following resources: %(unders)s'

exception InvalidReplicationTarget(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Invalid*

message = 'Invalid Replication Target: %(reason)s'

exception InvalidReservationExpiration(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Invalid*

message = 'Invalid reservation expiration %(expire)s.'

exception InvalidResults(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Invalid*

message = 'The results are invalid.'

exception InvalidSignatureImage(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Invalid*

message = 'Signature metadata is incomplete for image: %(image_id)s.'

exception InvalidSnapshot(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Invalid*

message = 'Invalid snapshot: %(reason)s'

exception InvalidTypeAvailabilityZones(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Invalid*

message = 'Volume type is only supported in these availability zones: %(az)s'

exception InvalidUUID(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Invalid*

message = 'Expected a UUID but received %(uuid)s.'

exception InvalidVolume(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Invalid*

message = 'Invalid volume: %(reason)s'

exception InvalidVolumeAttachMode(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Invalid*

message = "Invalid attaching mode '%(mode)s' for volume %(volume_id)s."

exception InvalidVolumeMetadata(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Invalid*

message = 'Invalid metadata: %(reason)s'

exception InvalidVolumeMetadataSize(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Invalid*

message = 'Invalid metadata size: %(reason)s'

exception InvalidVolumeType(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Invalid*

message = 'Invalid volume type: %(reason)s'

exception KeyManagerError(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = 'key manager error: %(reason)s'

exception LockCreationFailed(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = 'Unable to create lock. Coordination backend not started.'

exception MalformedRequestBody(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = 'Malformed message body: %(reason)s'

exception MalformedResponse(*message: str | tuple | None = None*, ***kwargs*)

Bases: *VolumeDriverException*

message = 'Malformed response to command %(cmd)s: %(reason)s'

exception ManageExistingAlreadyManaged(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = 'Unable to manage existing volume. Volume %(volume_ref)s already managed.'

exception ManageExistingInvalidReference(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = 'Manage existing volume failed due to invalid backend reference %(existing_ref)s: %(reason)s'

exception ManageExistingVolumeTypeMismatch(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = 'Manage existing volume failed due to volume type mismatch: %(reason)s'

exception MessageNotFound(*message: str | tuple | None = None*, ***kwargs*)

Bases: *NotFound*

message = 'Message %(message_id)s could not be found.'

exception MetadataAbsent(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = 'There is no metadata in DB object.'

exception MetadataCopyFailure(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Invalid*

message = 'Failed to copy metadata to volume: %(reason)s'

exception MetadataUpdateFailure(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Invalid*

message = 'Failed to update metadata for volume: %(reason)s'

exception NfsException(*message: str | tuple | None = None*, ***kwargs*)

Bases: *RemoteFSException*

message = 'Unknown NFS exception'

exception NfsNoSharesMounted(*message: str | tuple | None = None*, ***kwargs*)

Bases: *RemoteFSNoSharesMounted*

message = 'No mounted NFS shares found'

exception NfsNoSuitableShareFound(*message: str | tuple | None = None*, ***kwargs*)

Bases: *RemoteFSNoSuitableShareFound*

message = 'There is no share which can host %(volume_size)sG'

exception NoValidBackend(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = 'No valid backend was found. %(reason)s'

exception NotAuthorized(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

code = 403

message = 'Not authorized.'

exception NotFound(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

code = 404

message = 'Resource could not be found.'

safe = True

exception NotSupportedOperation(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Invalid*

code = 405

message = 'Operation not supported: %(operation)s.'

exception OverQuota(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = 'Quota exceeded for resources: %(overs)s'

exception ParameterNotFound(*message: str | tuple | None = None*, ***kwargs*)

Bases: *NotFound*

message = 'Could not find parameter %(param)s'

exception PolicyNotAuthorized(*message: str | tuple | None = None*, ***kwargs*)

Bases: *NotAuthorized*

message = "Policy doesn't allow %(action)s to be performed."

exception ProgrammingError(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = 'Programming error in Cinder: %(reason)s'

exception ProjectQuotaNotFound(*message: str | tuple | None = None*, ***kwargs*)

Bases: *QuotaNotFound*

message = 'Quota for project %(project_id)s could not be found.'

exception QoSSpecsAssociateFailed(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = 'Failed to associate qos_specs: %(specs_id)s with type %(type_id)s.'

exception QoSSpecsCreateFailed(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = 'Failed to create qos_specs: %(name)s with specs %(qos_specs)s.'

exception QoSSpecsDisassociateFailed(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = 'Failed to disassociate qos_specs: %(specs_id)s with type %(type_id)s.'

exception QoSSpecsExists(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Duplicate*

message = 'QoS Specs %(specs_id)s already exists.'

exception QoSSpecsInUse(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = 'QoS Specs %(specs_id)s is still associated with entities.'

exception QoSSpecsKeyNotFound(*message: str | tuple | None = None*, ***kwargs*)

Bases: *NotFound*

message = 'QoS spec %(specs_id)s has no spec with key %(specs_key)s.'

exception QoSSpecsNotFound(*message: str | tuple | None = None*, ***kwargs*)

Bases: *NotFound*

message = 'No such QoS spec %(specs_id)s.'

exception QoSSpecsUpdateFailed(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = 'Failed to update qos_specs: %(specs_id)s with specs %(qos_specs)s.'

exception QuotaClassNotFound(*message: str | tuple | None = None*, ***kwargs*)

Bases: *QuotaNotFound*

message = 'Quota class %(class_name)s could not be found.'

exception QuotaError(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

code = 413

headers: dict = {'Retry-After': '0'}

message = 'Quota exceeded: code=%(code)s'

safe = True

exception QuotaNotFound(*message: str | tuple | None = None*, ***kwargs*)

Bases: *NotFound*

message = 'Quota could not be found'

exception QuotaResourceUnknown(*message: str | tuple | None = None*, ***kwargs*)

Bases: *QuotaNotFound*

message = 'Unknown quota resources %(unknown)s.'

exception QuotaUsageNotFound(*message: str | tuple | None = None*, ***kwargs*)

Bases: *QuotaNotFound*

message = 'Quota usage for project %(project_id)s could not be found.'

exception RPCTimeout(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

code = 502

message = 'Timeout while requesting capabilities from backend %(service)s.'

exception RekeyNotSupported(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = 'Rekey not supported.'

exception RemoteFSConcurrentRequest(*message: str | tuple | None = None*, ***kwargs*)

Bases: *RemoteFSException*

message = 'A concurrent, possibly contradictory, request has been made.'

exception RemoteFSException(*message: str | tuple | None = None*, ***kwargs*)

Bases: *VolumeDriverException*

message = 'Unknown RemoteFS exception'

exception RemoteFSInvalidBackingFile(*message: str | tuple | None = None*, ***kwargs*)

Bases: *VolumeDriverException*

message = 'File %(path)s has invalid backing file %(backing_file)s.'

exception RemoteFSNoSharesMounted(*message: str | tuple | None = None*, ***kwargs*)

Bases: *RemoteFSException*

message = 'No mounted shares found'

exception RemoteFSNoSuitableShareFound(*message: str | tuple | None = None*, ***kwargs*)

Bases: *RemoteFSException*

message = 'There is no share which can host %(volume_size)sG'

exception RemoveExportException(*message: str | tuple | None = None*, ***kwargs*)

Bases: *VolumeDriverException*

message = 'Failed to remove export for volume %(volume)s: %(reason)s'

exception ReplicationError(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = 'Volume %(volume_id)s replication error: %(reason)s'

exception ReplicationGroupError(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = 'Group %(group_id)s replication error: %(reason)s.'

exception RequirementMissing(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = 'Requirement %(req)s is not installed.'

exception SSHInjectionThreat(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = 'SSH command injection detected: %(command)s'

exception SchedulerHostFilterNotFound(*message: str | tuple | None = None*, ***kwargs*)

Bases: *NotFound*

message = 'Scheduler Host Filter %(filter_name)s could not be found.'

exception SchedulerHostWeigherNotFound(*message: str | tuple | None = None*, ***kwargs*)

Bases: *NotFound*

message = 'Scheduler Host Weigher %(weigher_name)s could not be found.'

exception ServerNotFound(*message: str | tuple | None = None*, ***kwargs*)

Bases: *NotFound*

message = 'Instance %(uuid)s could not be found.'

exception ServiceNotFound(*message=None*, ***kwargs*)

Bases: *NotFound*

exception ServiceTooOld(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Invalid*

message = 'Service is too old to fulfil this request.'

exception ServiceUnavailable(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Invalid*

message = 'Service is unavailable at this time.'

exception ServiceUserTokenNoAuth(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = 'The [service_user] send_service_user_token option was requested, but no service auth could be loaded. Please check the [service_user] configuration section.'

exception SnapshotIsBusy(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = 'deleting snapshot %(snapshot_name)s that has dependent volumes'

exception SnapshotLimitExceeded(*message: str | tuple | None = None*, ***kwargs*)

Bases: *QuotaError*

message = 'Maximum number of snapshots allowed (%(allowed)d) exceeded'

exception SnapshotLimitReached(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = 'Exceeded the configured limit of %(set_limit)s snapshots per volume.'

exception SnapshotMetadataNotFound(*message: str | tuple | None = None*, ***kwargs*)

Bases: *NotFound*

message = 'Snapshot %(snapshot_id)s has no metadata with key %(metadata_key)s.'

exception SnapshotNotFound(*message: str | tuple | None = None*, ***kwargs*)

Bases: *NotFound*

message = 'Snapshot %(snapshot_id)s could not be found.'

exception SnapshotUnavailable(*message: str | tuple | None = None*, ***kwargs*)

Bases: *VolumeBackendAPIException*

message = 'The snapshot is unavailable: %(data)s'

exception SwiftConnectionFailed(*message: str | tuple | None = None*, ***kwargs*)

Bases: *BackupDriverException*

message = 'Connection to swift failed: %(reason)s'

exception TargetUpdateFailed(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = 'Failed to update target for volume %(volume_id)s.'

exception TransferNotFound(*message: str | tuple | None = None*, ***kwargs*)

Bases: *NotFound*

message = 'Transfer %(transfer_id)s could not be found.'

exception UnableToFailOver(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = 'Unable to failover to replication target: %(reason)s).'

exception UnavailableDuringUpgrade(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Invalid*

message = 'Cannot perform %(action)s during system upgrade.'

exception UnexpectedOverQuota(*message: str | tuple | None = None*, ***kwargs*)

Bases: *QuotaError*

message = 'Unexpected over quota on %(name)s.'

exception UnknownCmd(*message: str | tuple | None = None*, ***kwargs*)

Bases: *VolumeDriverException*

message = 'Unknown or unsupported command %(cmd)s'

exception ValidationError(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Invalid*

message = '%(detail)s'

exception VersionNotFoundForAPIMethod(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Invalid*

message = 'API version %(version)s is not supported on this method.'

exception VolumeAttached(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Invalid*

message = 'Volume %(volume_id)s is still attached, detach volume first.'

exception VolumeAttachmentNotFound(*message: str | tuple | None = None*, ***kwargs*)

Bases: *NotFound*

message = 'Volume attachment could not be found with filter: %(filter)s.'

exception VolumeBackendAPIException(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = 'Bad or unexpected response from the storage volume backend API: %(data)s'

exception VolumeBackupSizeExceedsAvailableQuota(*message: str | tuple | None = None*, ***kwargs*)

Bases: *QuotaError*

message = 'Requested backup exceeds allowed Backup gigabytes quota. Requested %(requested)sG, quota is %(quota)sG and %(consumed)sG has been consumed.'

exception VolumeDeviceNotFound(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = 'Volume device not found at %(device)s.'

exception VolumeDriverException(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = 'Volume driver reported an error: %(message)s'

exception VolumeGroupCreationFailed(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = 'Failed to create Volume Group: %(vg_name)s'

exception VolumeGroupNotFound(*message: str | tuple | None = None*, ***kwargs*)

Bases: `CinderException`

message = 'Unable to find Volume Group: %(vg_name)s'

exception VolumeIsBusy(*message: str | tuple | None = None*, ***kwargs*)

Bases: `CinderException`

message = 'deleting volume %(volume_name)s that has snapshot'

exception VolumeLimitExceeded(*message=None*, ***kwargs*)

Bases: `QuotaError`

message = "Maximum number of volumes allowed (%(allowed)d) exceeded for quota '%(name)s'."

exception VolumeMetadataBackupExists(*message: str | tuple | None = None*, ***kwargs*)

Bases: `BackupDriverException`

message = 'Metadata backup already exists for this volume'

exception VolumeMetadataNotFound(*message: str | tuple | None = None*, ***kwargs*)

Bases: `NotFound`

message = 'Volume %(volume_id)s has no metadata with key %(metadata_key)s.'

exception VolumeMigrationFailed(*message: str | tuple | None = None*, ***kwargs*)

Bases: `CinderException`

message = 'Volume migration failed: %(reason)s'

exception VolumeNotDeactivated(*message: str | tuple | None = None*, ***kwargs*)

Bases: `CinderException`

message = 'Volume %(name)s was not deactivated in time.'

exception VolumeNotFound(*message: str | tuple | None = None*, ***kwargs*)

Bases: `NotFound`

message = 'Volume %(volume_id)s could not be found.'

exception VolumeSizeExceedsAvailableQuota(*message=None*, ***kwargs*)

Bases: `QuotaError`

message = 'Requested volume or snapshot exceeds allowed %(name)s quota. Requested %(requested)sG, quota is %(quota)sG and %(consumed)sG has been consumed.'

exception VolumeSizeExceedsLimit(*message: str | tuple | None = None*, ***kwargs*)

Bases: `QuotaError`

message = 'Requested volume size %(size)dG is larger than maximum allowed limit %(limit)dG.'

exception VolumeSnapshotNotFound(*message: str | tuple | None = None*, ***kwargs*)

Bases: `NotFound`

message = 'No snapshots found for volume %(volume_id)s.'

exception VolumeTypeAccessExists(*message: str | tuple | None = None*, ***kwargs*)

Bases: `Duplicate`

message = 'Volume type access for %(volume_type_id)s / %(project_id)s combination already exists.'

exception VolumeTypeAccessNotFound(*message: str | tuple | None = None*, ***kwargs*)

Bases: `NotFound`

message = 'Volume type access not found for %(volume_type_id)s / %(project_id)s combination.'

exception VolumeTypeCreateFailed(*message: str | tuple | None = None*, ***kwargs*)

Bases: `CinderException`

message = 'Cannot create volume_type with name %(name)s and specs %(extra_specs)s'

exception VolumeTypeDefaultDeletionError(*message: str | tuple | None = None*, ***kwargs*)

Bases: `Invalid`

message = 'The volume type %(volume_type_id)s is a default volume type and cannot be deleted.'

exception VolumeTypeDefaultMisconfiguredError(*message: str | tuple | None = None*, ***kwargs*)

Bases: `CinderException`

message = 'The request cannot be fulfilled as the default volume type %(volume_type_name)s cannot be found.'

exception VolumeTypeDeletionError(*message: str | tuple | None = None*, ***kwargs*)

Bases: `Invalid`

message = 'The volume type %(volume_type_id)s is the only currently defined volume type and cannot be deleted.'

exception VolumeTypeEncryptionExists(*message: str | tuple | None = None*, ***kwargs*)

Bases: `Invalid`

message = 'Volume type encryption for type %(type_id)s already exists.'

exception VolumeTypeEncryptionNotFound(*message: str | tuple | None = None*, ***kwargs*)

Bases: `NotFound`

message = 'Volume type encryption for type %(type_id)s does not exist.'

exception VolumeTypeExists(*message: str | tuple | None = None*, ***kwargs*)

Bases: `Duplicate`

message = 'Volume Type %(id)s already exists.'

exception VolumeTypeExtraSpecsNotFound(*message: str | tuple | None = None*, ***kwargs*)

Bases: *NotFound*

message = 'Volume Type %(volume_type_id)s has no extra specs with key %(extra_specs_key)s.'

exception VolumeTypeInUse(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = 'Volume Type %(volume_type_id)s deletion is not allowed with volumes present with the type.'

exception VolumeTypeNotFound(*message: str | tuple | None = None*, ***kwargs*)

Bases: *NotFound*

message = 'Volume type %(volume_type_id)s could not be found.'

exception VolumeTypeNotFoundByName(*message: str | tuple | None = None*, ***kwargs*)

Bases: *VolumeTypeNotFound*

message = 'Volume type with name %(volume_type_name)s could not be found.'

exception VolumeTypeProjectDefaultNotFound(*message: str | tuple | None = None*, ***kwargs*)

Bases: *NotFound*

message = 'Default type for project %(project_id)s not found.'

exception VolumeTypeUpdateFailed(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = 'Cannot update volume_type %(id)s'

exception WorkerExists(*message: str | tuple | None = None*, ***kwargs*)

Bases: *Duplicate*

message = 'Worker for %(type)s %(id)s already exists.'

exception WorkerNotFound(*message=None*, ***kwargs*)

Bases: *NotFound*

message = 'Worker with %s could not be found.'

exception ZoneManagerException(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = 'Fibre Channel connection control failure: %(reason)s'

exception ZoneManagerNotInitialized(*message: str | tuple | None = None*, ***kwargs*)

Bases: *CinderException*

message = 'Fibre Channel Zone Manager not initialized'

cinder.flow_utils module

class CinderTask(*addons: list[str] | None = None*, ***kwargs: Any*)

Bases: `Task`

The root task class for all cinder tasks.

It automatically names the given task using the module and class that implement the given task as the task name.

classmethod make_name(*addons: list[str] | None = None*) → str

class DynamicLogListener(*engine*, *task_listen_for=('*',)*, *flow_listen_for=('*',)*, *retry_listen_for=('*',)*, *logger=<KeywordArgumentAdapter cinder.flow_utils (WARNING)>*)

Bases: `DynamicLoggingListener`

This is used to attach to taskflow engines while they are running.

It provides a bunch of useful features that expose the actions happening inside a taskflow engine, which can be useful for developers for debugging, for operations folks for monitoring and tracking of the resource actions and more

class SpecialFormatter(*engine*)

Bases: `FailureFormatter`

format(*fail*, *atom_matcher*)

Returns a (exc_info, details) tuple about the failure.

The `exc_info` tuple should be a standard three element (exctype, value, traceback) tuple that will be used for further logging. A non-empty string is typically returned for `details`; it should contain any string info about the failure (with any specific details the `exc_info` may not have/contain).

cinder.i18n module

oslo.i18n integration module.

See https://docs.openstack.org/oslo.i18n/latest/user/index.html .

enable_lazy(*enable=True*)

get_available_languages()

translate(*value*, *user_locale=None*)

cinder.manager module

Base Manager class.

Managers are responsible for a certain aspect of the system. It is a logical grouping of code relating to a portion of the system. In general other components should be using the manager to make changes to the components that it is responsible for.

For example, other components that need to deal with volumes in some way, should do so by calling methods on the VolumeManager instead of directly changing fields in the database. This allows us to keep all of the code relating to volumes in the same place.

We have adopted a basic strategy of Smart managers and dumb data, which means rather than attaching methods to data objects, components should call manager methods that act on the data.

Methods on managers that can be executed locally should be called directly. If a particular method must execute on a remote host, this should be done via rpc to the service that wraps the manager

Managers should be responsible for most of the db access, and non-implementation specific data. Anything implementation specific that cant be generalized should be done by the Driver.

In general, we prefer to have one manager with multiple drivers for different implementations, but sometimes it makes sense to have multiple managers. You can think of it this way: Abstract different overall strategies at the manager level(FlatNetwork vs VlanNetwork), and different implementations at the driver level(LinuxNetDriver vs CiscoNetDriver).

Managers will often provide methods for initial setup of a host or periodic tasks to a wrapping service.

This module provides Manager, a base class for managers.

`class CleanableManager`

Bases: `object`

do_cleanup(*context: RequestContext*, *cleanup_request: CleanupRequest*) → None

init_host(*service_id*, *added_to_cluster=None*, ***kwargs*)

`class Manager`(*host: HostAddress | None = None*, *cluster=None*, ***_kwargs*)

Bases: `Base`, `PeriodicTasks`

RPC_API_VERSION = '1.0'

get_log_levels(*context*, *log_request*)

init_host(*service_id*, *added_to_cluster=None*)

Handle initialization if this is a standalone service.

A hook point for services to execute tasks before the services are made available (i.e. showing up on RPC and starting to accept RPC calls) to other components. Child classes should override this method.

Parameters

- **service_id** ID of the service where the manager is running.
- **added_to_cluster** True when a hosts cluster configuration has changed from not being defined or being to any other value and the DB service record reflects this new value.

init_host_with_rpc()

A hook for service to do jobs after RPC is ready.

Like init_host(), this method is a hook where services get a chance to execute tasks that *need* RPC. Child classes should override this method.

is_working()

Method indicating if service is working correctly.

This method is supposed to be overridden by subclasses and return if manager is working correctly.

reset()

Method executed when SIGHUP is caught by the process.

Were utilizing it to reset RPC API version pins to avoid restart of the service when rolling upgrade is completed.

property service_topic_queue

set_log_levels(*context*, *log_request*)

target = <Target version=1.0>

class PeriodicTasks

Bases: `PeriodicTasks`

class SchedulerDependentManager(*host=None*, *service_name='undefined'*, *cluster=None*, **args*, ***kwargs*)

Bases: *ThreadPoolManager*

Periodically send capability updates to the Scheduler services.

Services that need to update the Scheduler of their capabilities should derive from this class. Otherwise they can derive from manager.Manager directly. Updates are only sent after update_service_capabilities is called with non-None values.

reset()

Method executed when SIGHUP is caught by the process.

Were utilizing it to reset RPC API version pins to avoid restart of the service when rolling upgrade is completed.

update_service_capabilities(*capabilities*)

Remember these capabilities to send on next periodic update.

class ThreadPoolManager(**args*, ***kwargs*)

Bases: *Manager*

cinder.opts module

list_opts()

cinder.policy module

Policy Engine For Cinder

authorize(*context*, *action: str*, *target: dict*, *do_raise: bool = True*, *exc: Type[Exception] | None = None*)

Verifies that the action is valid on the target in this context.

Parameters

- **context** cinder context
- **action** string representing the action to be checked this should be colon separated for clarity. i.e. `compute:create_instance`, `compute:attach_volume`, `volume:attach_volume`

- **target** dictionary representing the object of the action for object creation this should be a dictionary representing the location of the object e.g. `{'project_id': context.project_id}`
- **do_raise** if True (the default), raises PolicyNotAuthorized; if False, returns False
- **exc** Class of the exception to raise if the check fails. Any remaining arguments passed to `authorize()` (both positional and keyword arguments) will be passed to the exception class. If not specified, `PolicyNotAuthorized` will be used.

Raises
cinder.exception.PolicyNotAuthorized if verification fails and do_raise is True. Or if exc is specified it will raise an exception of that type.

Returns
returns a non-False value (not necessarily True) if authorized, and the exact value False if not authorized and do_raise is False.

check_is_admin(*context: RequestContext*)

Whether or not user is admin according to policy setting.

enforce(*context*, *action: str*, *target: dict*)

Verifies that the action is valid on the target in this context.

Parameters

- **context** cinder context
- **action** string representing the action to be checked this should be colon separated for clarity. i.e. `compute:create_instance`, `compute:attach_volume`, `volume:attach_volume`
- **target** dictionary representing the object of the action for object creation this should be a dictionary representing the location of the object e.g. `{'project_id': context.project_id}`

Raises
PolicyNotAuthorized if verification fails.

get_enforcer() → Enforcer

get_rules()

init(*use_conf=True*, *suppress_deprecation_warnings: bool = False*) → None

Init an Enforcer class.

Parameters
use_conf Whether to load rules from config file.

register_rules(*enforcer*)

reset() → None

set_rules(*rules: dict*, *overwrite: bool = True*, *use_conf: bool = False*) → None

Set rules based on the provided dict of rules.

Parameters

- **rules** New rules to use. It should be an instance of dict.
- **overwrite** Whether to overwrite current rules or update them with the new rules.
- **use_conf** Whether to reload rules from config file.

cinder.quota module

Quotas for volumes.

class BaseResource(*name*, *flag=None*, *parent_project_id=None*)

Bases: object

Describe a single resource for quota checking.

property default

Return the default value of the quota.

quota(*driver*, *context*, ***kwargs*)

Given a driver and context, obtain the quota for this resource.

Parameters

- **driver** A quota driver.
- **context** The request context.
- **project_id** The project to obtain the quota value for. If not provided, it is taken from the context. If it is given as None, no project-specific quota will be searched for.
- **quota_class** The quota class corresponding to the project, or for which the quota is to be looked up. If not provided, it is taken from the context. If it is given as None, no quota class-specific quota will be searched for. Note that the quota class defaults to the value in the context, which may not correspond to the project if project_id is not the same as the one in the context.

class DbQuotaDriver

Bases: object

Driver to perform check to enforcement of quotas.

Also allows to obtain quota information. The default driver utilizes the local database.

commit(*context*, *reservations*, *project_id=None*)

Commit reservations.

Parameters

- **context** The request context, for access checks.
- **reservations** A list of the reservation UUIDs, as returned by the reserve() method.
- **project_id** Specify the project_id if current context is admin and admin wants to impact on common users tenant.

`destroy_by_project`(*context*, *project_id*)

Destroy all limit quotas associated with a project.

Leave usage and reservation quotas intact.

Parameters

- `context` The request context, for access checks.
- `project_id` The ID of the project being deleted.

`expire`(*context*)

Expire reservations.

Explores all currently existing reservations and rolls back any that have expired.

Parameters

`context` The request context, for access checks.

`get_by_class`(*context*, *quota_class*, *resource_name*)

Get a specific quota by quota class.

`get_by_project`(*context*, *project_id*, *resource_name*)

Get a specific quota by project.

`get_class_quotas`(*context*, *resources*, *quota_class*, *defaults=True*)

Given list of resources, retrieve the quotas for given quota class.

Parameters

- `context` The request context, for access checks.
- `resources` A dictionary of the registered resources.
- `quota_class` The name of the quota class to return quotas for.
- `defaults` If True, the default value will be reported if there is no specific value for the resource.

`get_default`(*context*, *resource*, *project_id*)

Get a specific default quota for a resource.

`get_defaults`(*context*, *resources*, *project_id=None*)

Given a list of resources, retrieve the default quotas.

Use the class quotas named *_DEFAULT_QUOTA_NAME* as default quotas, if it exists.

Parameters

- `context` The request context, for access checks.
- `resources` A dictionary of the registered resources.
- `project_id` The id of the current project

`get_project_quotas`(*context*, *resources*, *project_id*, *quota_class=None*, *defaults=True*, *usages=True*)

Retrieve quotas for a project.

Given a list of resources, retrieve the quotas for the given project.

Parameters

- **context** The request context, for access checks.
- **resources** A dictionary of the registered resources.
- **project_id** The ID of the project to return quotas for.
- **quota_class** If project_id != context.project_id, the quota class cannot be determined. This parameter allows it to be specified. It will be ignored if project_id == context.project_id.
- **defaults** If True, the quota class value (or the default value, if there is no value from the quota class) will be reported if there is no specific value for the resource.
- **usages** If True, the current in_use and reserved counts will also be returned.

limit_check(*context*, *resources*, *values*, *project_id=None*)

Check simple quota limits.

For limitsthose quotas for which there is no usage synchronization functionthis method checks that a set of proposed values are permitted by the limit restriction.

This method will raise a QuotaResourceUnknown exception if a given resource is unknown or if it is not a simple limit resource.

If any of the proposed values is over the defined quota, an OverQuota exception will be raised with the sorted list of the resources which are too high. Otherwise, the method returns nothing.

Parameters

- **context** The request context, for access checks.
- **resources** A dictionary of the registered resources.
- **values** A dictionary of the values to check against the quota.
- **project_id** Specify the project_id if current context is admin and admin wants to impact on common users tenant.

reserve(*context*, *resources*, *deltas*, *expire=None*, *project_id=None*)

Check quotas and reserve resources.

For counting quotasthose quotas for which there is a usage synchronization functionthis method checks quotas against current usage and the desired deltas.

This method will raise a QuotaResourceUnknown exception if a given resource is unknown or if it does not have a usage synchronization function.

If any of the proposed values is over the defined quota, an OverQuota exception will be raised with the sorted list of the resources which are too high. Otherwise, the method returns a list of reservation UUIDs which were created.

Parameters

- **context** The request context, for access checks.
- **resources** A dictionary of the registered resources.
- **deltas** A dictionary of the proposed delta changes.

- **expire** An optional parameter specifying an expiration time for the reservations. If it is a simple number, it is interpreted as a number of seconds and added to the current time; if it is a datetime.timedelta object, it will also be added to the current time. A datetime.datetime object will be interpreted as the absolute expiration time. If None is specified, the default expiration time set by default-reservation-expire will be used (this value will be treated as a number of seconds).
- **project_id** Specify the project_id if current context is admin and admin wants to impact on common users tenant.

rollback(*context*, *reservations*, *project_id=None*)

Roll back reservations.

Parameters

- **context** The request context, for access checks.
- **reservations** A list of the reservation UUIDs, as returned by the reserve() method.
- **project_id** Specify the project_id if current context is admin and admin wants to impact on common users tenant.

class GroupQuotaEngine(*quota_driver_class=None*)

Bases: `QuotaEngine`

Represent the group quotas.

register_resource(*resource*)

Register a resource.

register_resources(*resources*)

Register a list of resources.

property resources

Fetches all possible quota resources.

class QuotaEngine(*quota_driver_class=None*)

Bases: `object`

Represent the set of recognized quotas.

add_volume_type_opts(*context*, *opts*, *volume_type_id*)

Add volume type resource options.

Adds elements to the opts hash for volume type quotas. If a resource is being reserved (gigabytes, etc) and the volume type is set up for its own quotas, these reservations are copied into keys for gigabytes_<volume type name>, etc.

Parameters

- **context** The request context, for access checks.
- **opts** The reservations options hash.
- **volume_type_id** The volume type id for this reservation.

`commit`(*context*, *reservations*, *project_id=None*)

Commit reservations.

Parameters

- `context` The request context, for access checks.
- `reservations` A list of the reservation UUIDs, as returned by the reserve() method.
- `project_id` Specify the project_id if current context is admin and admin wants to impact on common users tenant.

`destroy_by_project`(*context*, *project_id*)

Destroy all quota limits associated with a project.

Parameters

- `context` The request context, for access checks.
- `project_id` The ID of the project being deleted.

`expire`(*context*)

Expire reservations.

Explores all currently existing reservations and rolls back any that have expired.

Parameters

`context` The request context, for access checks.

`get_by_class`(*context*, *quota_class*, *resource_name*)

Get a specific quota by quota class.

`get_by_project`(*context*, *project_id*, *resource_name*)

Get a specific quota by project.

`get_by_project_or_default`(*context*, *project_id*, *resource_name*)

Get specific quota by project or default quota if doesnt exists.

`get_class_quotas`(*context*, *quota_class*, *defaults=True*)

Retrieve the quotas for the given quota class.

Parameters

- `context` The request context, for access checks.
- `quota_class` The name of the quota class to return quotas for.
- `defaults` If True, the default value will be reported if there is no specific value for the resource.

`get_default`(*context*, *resource*, *parent_project_id=None*)

Get a specific default quota for a resource.

Parameters

`parent_project_id` The id of the current projects parent, if any.

`get_defaults`(*context*, *project_id=None*)

Retrieve the default quotas.

Parameters

- `context` The request context, for access checks.
- `project_id` The id of the current project

`get_project_quotas`(*context*, *project_id*, *quota_class=None*, *defaults=True*, *usages=True*)

Retrieve the quotas for the given project.

Parameters

- `context` The request context, for access checks.
- `project_id` The ID of the project to return quotas for.
- `quota_class` If project_id != context.project_id, the quota class cannot be determined. This parameter allows it to be specified.
- `defaults` If True, the quota class value (or the default value, if there is no value from the quota class) will be reported if there is no specific value for the resource.
- `usages` If True, the current in_use and reserved counts will also be returned.

`limit_check`(*context*, *project_id=None*, ***values*)

Check simple quota limits.

For limitsthose quotas for which there is no usage synchronization functionthis method checks that a set of proposed values are permitted by the limit restriction. The values to check are given as keyword arguments, where the key identifies the specific quota limit to check, and the value is the proposed value.

This method will raise a QuotaResourceUnknown exception if a given resource is unknown or if it is not a simple limit resource.

If any of the proposed values is over the defined quota, an OverQuota exception will be raised with the sorted list of the resources which are too high. Otherwise, the method returns nothing.

Parameters

- `context` The request context, for access checks.
- `project_id` Specify the project_id if current context is admin and admin wants to impact on common users tenant.

`register_resource`(*resource*)

Register a resource.

`register_resources`(*resources*)

Register a list of resources.

`reserve`(*context*, *expire=None*, *project_id=None*, ***deltas*)

Check quotas and reserve resources.

For counting quotasthose quotas for which there is a usage synchronization functionthis method checks quotas against current usage and the desired deltas. The deltas are given as keyword arguments, and current usage and other reservations are factored into the quota check.

This method will raise a QuotaResourceUnknown exception if a given resource is unknown or if it does not have a usage synchronization function.

If any of the proposed values is over the defined quota, an OverQuota exception will be raised with the sorted list of the resources which are too high. Otherwise, the method returns a list of reservation UUIDs which were created.

Parameters

- `context` The request context, for access checks.
- `expire` An optional parameter specifying an expiration time for the reservations. If it is a simple number, it is interpreted as a number of seconds and added to the current time; if it is a datetime.timedelta object, it will also be added to the current time. A datetime.datetime object will be interpreted as the absolute expiration time. If None is specified, the default expiration time set by default-reservation-expire will be used (this value will be treated as a number of seconds).
- `project_id` Specify the project_id if current context is admin and admin wants to impact on common users tenant.

`property resource_names`

`property resources`

`rollback`(*context*, *reservations*, *project_id=None*)

Roll back reservations.

Parameters

- `context` The request context, for access checks.
- `reservations` A list of the reservation UUIDs, as returned by the reserve() method.
- `project_id` Specify the project_id if current context is admin and admin wants to impact on common users tenant.

`class ReservableResource`(*name*, *sync*, *flag=None*)

Bases: `BaseResource`

Describe a reservable resource.

`class VolumeTypeQuotaEngine`(*quota_driver_class=None*)

Bases: `QuotaEngine`

Represent the set of all quotas.

`register_resource`(*resource*)

Register a resource.

`register_resources`(*resources*)

Register a list of resources.

`property resources`

Fetches all possible quota resources.

`update_quota_resource`(*context*, *old_type_name*, *new_type_name*)

Update resource in quota.

This is to update resource in quotas, quota_classes, and quota_usages once the name of a volume type is changed.

Parameters

- **context** The request context, for access checks.
- **old_type_name** old name of volume type.
- **new_type_name** new name of volume type.

`class VolumeTypeResource(part_name, volume_type)`

Bases: `ReservableResource`

ReservableResource for a specific volume type.

cinder.quota_utils module

`get_volume_type_reservation(ctxt, volume, type_id, reserve_vol_type_only=False, negative=False)`

`process_reserve_over_quota(context, over_quota_exception, resource, size=None)`

Handle OverQuota exception.

Analyze OverQuota exception, and raise new exception related to resource type. If there are unexpected items in overs, UnexpectedOverQuota is raised.

Parameters

- **context** security context
- **over_quota_exception** OverQuota exception
- **resource** can be backups, snapshots, and volumes
- **size** requested size in reservation

cinder.rpc module

`class RequestContextSerializer(base)`

Bases: `Serializer`

`deserialize_context(context)`

Deserialize a dictionary into a request context.

Parameters
ctxt Request context dictionary

Returns
Deserialized form of entity

`deserialize_entity(context, entity)`

Deserialize something from primitive form.

Parameters

- **ctxt** Request context, in deserialized form
- **entity** Primitive to be deserialized

Returns
Deserialized form of entity

serialize_context(*context*)

Serialize a request context into a dictionary.

Parameters
`ctxt` Request context

Returns
Serialized form of context

serialize_entity(*context*, *entity*)

Serialize something to primitive form.

Parameters

- `ctxt` Request context, in deserialized form
- `entity` Entity to be serialized

Returns
Serialized form of entity

add_extra_exmods(**args*)

cleanup()

clear_extra_exmods()

get_allowed_exmods()

get_client(*target*, *version_cap=None*, *serializer=None*) → RPCClient

get_notifier(*service: str = None*, *host: str = None*, *publisher_id: str = None*) → Notifier

get_server(*target*, *endpoints*, *serializer=None*) → RPCServer

init(*conf*) → None

set_defaults(*control_exchange*)

cinder.service module

Generic Node base class for all workers that run on hosts.

class Launcher

Bases: `object`

class Service(*host: str*, *binary: str*, *topic: str*, *manager: str*, *report_interval: int | None = None*, *periodic_interval: int | None = None*, *periodic_fuzzy_delay: int | None = None*, *service_name: str | None = None*, *coordination: bool = False*, *cluster: str | None = None*, **args*, ***kwargs*)

Bases: `Service`

Service object for binaries running on hosts.

A service takes a manager and enables rpc by listening to queues based on topic. It also periodically runs tasks on the manager and reports it state to the database services table.

basic_config_check() → None

Perform basic config checks before starting service.

classmethod **create**(*host: str | None = None*, *binary: str | None = None*, *topic: str | None = None*, *manager: str | None = None*, *report_interval: int | None = None*, *periodic_interval: int | None = None*, *periodic_fuzzy_delay: int | None = None*, *service_name: str | None = None*, *coordination: bool = False*, *cluster: str | None = None*, ***kwargs*) → *Service*

Instantiates class and passes back application object.

Parameters

- **host** defaults to CONF.host
- **binary** defaults to basename of executable
- **topic** defaults to bin_name - cinder- part
- **manager** defaults to CONF.<topic>_manager
- **report_interval** defaults to CONF.report_interval
- **periodic_interval** defaults to CONF.periodic_interval
- **periodic_fuzzy_delay** defaults to CONF.periodic_fuzzy_delay
- **cluster** Defaults to None, as only some services will have it

periodic_tasks(*raise_on_error: bool = False*) → None

Tasks to be run at a periodic interval.

report_state() → None

Update the state of this service in the datastore.

reset() → None

Reset a service in case it received a SIGHUP.

service_id = None

start() → None

Start a service.

stop() → None

Stop a service.

Parameters

graceful indicates whether to wait for all threads to finish or terminate them instantly

wait() → None

Wait for a service to shut down.

class **WSGIService**(*name*, *loader=None*)

Bases: `ServiceBase`

Provides ability to launch API from a paste configuration.

reset() → None

Reset server greenpool size to default.

Returns
None

start() → None

Start serving this service using loaded configuration.

Also, retrieve updated port number in case 0 was passed in, which indicates a random port should be used.

Returns
None

stop() → None

Stop serving this API.

Returns
None

wait() → None

Wait for the service to stop serving this API.

Returns
None

class **WindowsProcessLauncher**

Bases: `object`

add_process(*cmd*)

wait()

get_launcher() → ProcessLauncher

process_launcher() → ProcessLauncher

serve(*server*, *workers=None*)

setup_profiler(*binary: str*, *host: str*) → None

wait() → None

cinder.service_auth module

get_auth_plugin(*context*, *auth=None*)

get_service_auth_plugin()

get_service_session()

reset_globals()

For async unit test consistency.

cinder.ssh_utils module

Utilities related to SSH connection management.

class SSHPool(*ip*, *port*, *conn_timeout*, *login*, *password=None*, *privatekey=None*, **args*, ***kwargs*)

Bases: `Pool`

A simple eventlet pool to hold ssh connections.

create()

Generate a new pool item. In order for the pool to function, either this method must be overriden in a subclass or the pool must be constructed with the *create* argument. It accepts no arguments and returns a single instance of whatever thing the pool is supposed to contain.

In general, `create()` is called whenever the pool exceeds its previous high-water mark of concurrently-checked-out-items. In other words, in a new pool with *min_size* of 0, the very first call to `get()` will result in a call to `create()`. If the first caller calls `put()` before some other caller calls `get()`, then the first item will be returned, and `create()` will not be called a second time.

get()

Return an item from the pool, when one is available.

This may cause the calling greenthread to block. Check if a connection is active before returning it.

For dead connections create and return a new connection.

put(*conn*)

Put an item back into the pool, when done. This may cause the putting greenthread to block.

remove(*ssh*)

Close an ssh client and remove it from free_items.

cinder.utils module

Utilities and helper functions for all Cinder code.

This file is for utilities useful in all of Cinder, including cinder-manage, the api service, the scheduler, etc.

Code related to volume drivers and connecting to volumes should be placed in volume_utils instead.

class ComparableMixin

Bases: `object`

class DoNothing

Bases: `str`

Class that literrally does nothing.

We inherit from str in case its called with json.dumps.

class Semaphore(*limit*)

Bases: `object`

Custom semaphore to workaround eventlet issues with multiprocessing.

api_clean_volume_file_locks(*volume_id*)

as_int(*obj: int | float | str*, *quiet: bool = True*) → int

build_or_str(*elements: None | str | Iterable[str]*, *str_format: str | None = None*) → str

Builds a string of elements joined by or.

Will join strings with the or word and if a str_format is provided it will be used to format the resulted joined string. If there are no elements an empty string will be returned.

Parameters

- **elements** (*String or iterable of strings.*) Elements we want to join.
- **str_format** (*String.*) String to use to format the response.

calculate_capacity_factors(*total_capacity: float*, *free_capacity: float*, *provisioned_capacity: float*, *thin_provisioning_support: bool*, *max_over_subscription_ratio: float*, *reserved_percentage: int*, *thin: bool*) → dict

Create the various capacity factors of the a particular backend.

Based off of definition of terms cinder-specs/specs/queens/provisioning-improvements.html Description of factors calculated where units of gb are Gibibytes. reserved_capacity - The amount of space reserved from the total_capacity as reported by the backend. total_reserved_available_capacity - The total capacity minus reserved capacity total_available_capacity - The total capacity available to cinder calculated from total_reserved_available_capacity (for thick) OR for thin total_reserved_available_capacity max_over_subscription_ratio calculated_free_capacity - total_available_capacity - provisioned_capacity virtual_free_capacity - The calculated free capacity available to cinder to allocate new storage. For thin: calculated_free_capacity For thick: the reported free_capacity can be less than the calculated capacity, so we use free_capacity - reserved_capacity.

free_percent - the percentage of the virtual_free_capacity and total_available_capacity is left over provisioned_ratio - The ratio of provisioned storage to total_available_capacity

Parameters

- **total_capacity** (*float*) The reported total capacity in the backend.
- **free_capacity** (*float*) The free space/capacity as reported by the backend.
- **provisioned_capacity** (*float*) as reported by backend or volume manager from allocated_capacity_gb
- **thin_provisioning_support** (*bool*) Is thin provisioning supported?
- **max_over_subscription_ratio** (*float*) as reported by the backend
- **reserved_percentage** (*int, 0-100*) the % amount to reserve as unavailable. 0-100
- **thin** (*bool*) calculate based on thin provisioning if enabled by thin_provisioning_support

Returns

A dictionary of all of the capacity factors.

Return type
dict

`calculate_max_over_subscription_ratio`(*capability: dict*, *global_max_over_subscription_ratio: float*) → float

`calculate_virtual_free_capacity`(*total_capacity: float*, *free_capacity: float*, *provisioned_capacity: float*, *thin_provisioning_support: bool*, *max_over_subscription_ratio: float*, *reserved_percentage: int*, *thin: bool*) → float

Calculate the virtual free capacity based on multiple factors.

Parameters

- **total_capacity** total_capacity_gb of a host_state or pool.
- **free_capacity** free_capacity_gb of a host_state or pool.
- **provisioned_capacity** provisioned_capacity_gb of a host_state or pool.
- **thin_provisioning_support** thin_provisioning_support of a host_state or a pool.
- **max_over_subscription_ratio** max_over_subscription_ratio of a host_state or a pool
- **reserved_percentage** reserved_percentage of a host_state or a pool.
- **thin** whether volume to be provisioned is thin

Returns
the calculated virtual free capacity.

`check_exclusive_options`(***kwargs: dict | str | bool | None*) → None

Checks that only one of the provided options is actually not-none.

Iterates over all the kwargs passed in and checks that only one of said arguments is not-none, if more than one is not-none then an exception will be raised with the names of those arguments who were not-none.

`check_metadata_properties`(*metadata: dict[str, str] | None*) → None

Checks that the volume metadata properties are valid.

`check_ssh_injection`(*cmd_list: list[str]*) → None

`check_string_length`(*value: str*, *name: str*, *min_length: int = 0*, *max_length: int | None = None*, *allow_all_spaces: bool = True*) → None

Check the length of specified string.

Parameters

- **value** the value of the string
- **name** the name of the string
- **min_length** the min_length of the string
- **max_length** the max_length of the string

`clean_snapshot_file_locks`(*snapshot_id*, *driver*)

clean_volume_file_locks(*volume_id*, *driver*)

Remove file locks used by Cinder.

This doesnt take care of driver locks, those should be handled in drivers delete_volume method.

convert_str(*text: str | bytes*) → str

Convert to native string.

Convert bytes and Unicode strings to native strings:

- convert to bytes on Python 2: encode Unicode using encodeutils.safe_encode()
- convert to Unicode on Python 3: decode bytes from UTF-8

create_ordereddict(*adict: dict*) → OrderedDict

Given a dict, return a sorted OrderedDict.

execute(**cmd: str*, ***kwargs: bool | str*) → tuple[str, str]

Convenience wrapper around oslos execute() method.

get_blkdev_major_minor(*path: str*, *lookup_for_file: bool = True*) → str | None

Get major:minor number of block device.

Get the devices major:minor number of a block device to control I/O ratelimit of the specified path. If lookup_for_file is True and the path is a regular file, lookup a disk device which the file lies on and returns the result for the device.

get_bool_param(*param_string: str*, *params: dict*, *default: bool = False*) → bool

get_file_gid(*path: str*) → int

This primarily exists to make unit testing easier.

get_file_mode(*path: str*) → int

This primarily exists to make unit testing easier.

get_file_size(*path: str*) → int

Returns the file size.

get_log_levels(*prefix: str*) → dict

get_log_method(*level_string: str*) → int

get_root_helper() → str

if_notifications_enabled(*f: Callable*) → Callable

Calls decorated method only if notifications are enabled.

is_blk_device(*dev: str*) → bool

last_completed_audit_period(*unit: str | None = None*) → tuple[datetime | timedelta, datetime | timedelta]

This method gives you the most recently *completed* audit period.

arguments:

units: string, one of hour, day, month, year

Periods normally begin at the beginning (UTC) of the period unit (So a day period begins at midnight UTC, a month unit on the 1st, a year on Jan, 1) unit string may be appended with an optional offset like so: day@18 This will begin the period at 18:00

UTC. month@15 starts a monthly period on the 15th, and year@3 begins a yearly one on March 1st.

returns: 2 tuple of datetimes (begin, end)
The begin timestamp of this audit period is the same as the end of the previous.

limit_operations(*func: Callable*) → Callable

Decorator to limit the number of concurrent operations.

This method decorator expects to have a _semaphore attribute holding an initialized semaphore in the self instance object.

We can get the appropriate semaphore with the semaphore_factory method.

make_dev_path(*dev: str*, *partition: str | None = None*, *base: str = '/dev'*) → str

Return a path to a particular device.

```
>>> make_dev_path('xvdc')
/dev/xvdc
```

```
>>> make_dev_path('xvdc', 1)
/dev/xvdc1
```

monkey_patch() → None

Patches decorators for all functions in a specified module.

If the CONF.monkey_patch set as True, this function patches a decorator for all functions in specified modules.

You can set decorators for each modules using CONF.monkey_patch_modules. The format is Module path:Decorator function. Example: cinder.api.ec2.cloud: cinder.openstack.common.notifier.api.notify_decorator

Parameters of the decorator are as follows. (See cinder.openstack.common.notifier.api.notify_decorator)

Parameters

- **name** name of the function
- **function** object of the function

notifications_enabled(*conf*)

Check if oslo notifications are enabled.

paths_normcase_equal(*path_a: str*, *path_b: str*) → bool

retry(*retry_param: None | ~typing.Type[Exception] | tuple[~typing.Type[Exception], ...] | int | tuple[int, ...], interval: float = 1, retries: int = 3, backoff_rate: float = 2, wait_random: bool = False, retry=<class 'tenacity.retry.retry_if_exception_type'>*) → Callable

class retry_if_exit_code(*codes*)

Bases: `retry_if_exception`

Retry on ProcessExecutionError specific exit codes.

`robust_file_write`(*directory: str*, *filename: str*, *data: str*) → None

Robust file write.

Use write to temp file and rename model for writing the persistence file.

Parameters

- **directory** Target directory to create a file.
- **filename** File name to store specified data.
- **data** String data.

`semaphore_factory`(*limit: int*, *concurrent_processes: int*) → Semaphore | *Semaphore*

Get a semaphore to limit concurrent operations.

The semaphore depends on the limit we want to set and the concurrent processes that need to be limited.

`service_expired_time`(*with_timezone: bool | None = False*) → datetime

`set_log_levels`(*prefix: str*, *level_string: str*) → None

`tempdir`(***kwargs*) → Iterator[str]

`temporary_chown`(*path: str*, *owner_uid: int | None = None*) → Iterator[None]

Temporarily chown a path.

Params owner_uid

UID of temporary owner (defaults to current user)

`validate_dictionary_string_length`(*specs: dict*) → None

Check the length of each key and value of dictionary.

cinder.version module

Module contents

Root Cinder module.

CHAPTER

FIVE

FOR REVIEWERS

- *Python 2 to Python 3 transition guidelines*

CHAPTER

SIX

ADDITIONAL REFERENCE

Contents:

6.1 Glossary

This glossary offers a list of terms and definitions to define a vocabulary for Cinder concepts.

Logical Volume Manager (LVM)
Provides a method of allocating space on mass-storage devices that is more flexible than conventional partitioning schemes.

iSCSI Qualified Name (IQN)
IQN is the format most commonly used for iSCSI names, which uniquely identify nodes in an iSCSI network. All IQNs follow the pattern `iqn.yyyy-mm.domain:identifier`, where `yyyy-mm` is the year and month in which the domain was registered, `domain` is the reversed domain name of the issuing organization, and `identifier` is an optional string which makes each IQN under the same domain unique.

For example: `iqn.2015-10.org.openstack.408ae959bce1`

NVMe Qualified Name (NQN)
NQN is the format most commonly used for NVMe names, which uniquely identify hosts or NVM subsystems in a network. NQNs can follow one of two supported formats.

The first format, used by organizations that own a domain, is `nqn.yyyy-mm.domain:identifier`, where `yyyy-mm` is the year and month in which the domain was registered, `domain` is the reversed domain name of the issuing organization, and `identifier` is an optional string which makes each NQN unique under the same domain name.

For example: `nqn.2014-08.com.example:nvme:nvm-subsystem-sn-d78432`

The second format is used to create unique identifiers when there is not a naming authority or there is not a requirement for a human interpretable string. This format is `nqn.2014-08.org.nvmexpress:uuid:identifier`, where only the `identifier` is variable and consists of a 128-bit UUID based on the definition in RFC 4122 represented as a string.

For example: `nqn.2014-08.org.nvmexpress:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf6`