Zun Documentation

Release 9.0.1.dev4

Zun development team

Jun 22, 2023

CONTENTS

CHAPTER

ONE

WHAT IS ZUN?

Zun is an OpenStack Container service. It aims to provide an API service for running application containers without the need to manage servers or clusters.

It requires the following additional OpenStack services for basic function:

- Keystone
- Neutron
- Kuryr-libnetwork

It can also integrate with other services to include:

- Cinder
- Heat
- Glance

CHAPTER

TWO

FOR END USERS

As an end user of Zun, youll use Zun to create and manage containerized workload with either tools or the API directly. All end user (and some administrative) features of Zun are exposed via a REST API, which can be consumed directly. The following resources will help you get started with consuming the API directly.

- API Reference

Alternatively, end users can consume the REST API via various tools or SDKs. These tools are collected below.

- Horizon: The official web UI for the OpenStack Project.
- OpenStack Client: The official CLI for OpenStack Projects.
- Zun Client: The Python client for consuming the Zuns API.

CHAPTER

THREE

FOR OPERATORS

3.1 Installation

The detailed install guide for Zun. A functioning Zun will also require having installed Keystone, Neutron, and Kuryr-libnetwork. Please ensure that you follow their install guides first.

3.1.1 Zun Installation Guide

Overview

The Container service provides OpenStack-native API for launching and managing application containers without any virtual machine managements.

Also known as the zun project, the OpenStack Container service may, depending upon configuration, interact with several other OpenStack services. This includes:

- The OpenStack Identity service (`keystone`) for request authentication and to locate other OpenStack services
- The OpenStack Networking service (`neutron`) for DHCP and network configuration
- The Docker remote network driver for OpenStack (`kuryr-libnetwork`)
- The OpenStack Placement service (`placement`) for resource tracking and container allocation claiming.
- The OpenStack Block Storage (`cinder`) provides volumes for container (optional).
- The OpenStack Image service (`glance`) from which to retrieve container images (optional).
- The OpenStack Dashboard service (`horizon`) for providing the web UI (optional).
- The OpenStack Orchestration service (`heat`) for providing orchestration between containers and other OpenStack resources (optional).

Zun requires at least two nodes (Controller node and Compute node) to run a container. Optional services such as Block Storage require additional nodes.

Controller

The controller node runs the Identity service, Image service, management portions of Zun, management portion of Networking, various Networking agents, and the Dashboard. It also includes supporting services such as an SQL database, message queue, and Network Time Protocol (NTP).

Optionally, the controller node runs portions of the Block Storage, Object Storage, and Orchestration services.

The controller node requires a minimum of two network interfaces.

Compute

The compute node runs the engine portion of Zun that operates containers. By default, Zun uses Docker as container engine. The compute node also runs a Networking service agent that connects containers to virtual networks and provides firewalling services to instances via security groups.

You can deploy more than one compute node. Each node requires a minimum of two network interfaces.

Container service overview

The Container service consists of the following components:

zun-api An OpenStack-native REST API that processes API requests by sending them to the `zun-compute` over Remote Procedure Call (RPC).

zun-compute A worker daemon that creates and terminates containers or capsules (pods) through container engine API. Manage containers, capsules and compute resources in local host.

zun-wsproxy Provides a proxy for accessing running containers through a websocket connection.

zun-cni-daemon Provides a CNI daemon service that provides implementation for the Zun CNI plugin.

Optionally, one may wish to utilize the following associated projects for additional functionality:

python-zunclient A command-line interface (CLI) and python bindings for interacting with the Container service.

zun-ui The Horizon plugin for providing Web UI for Zun.

Install and configure controller node

This section describes how to install and configure the Container service on the controller node for Ubuntu 16.04 (LTS) and CentOS 7.

Prerequisites

Before you install and configure Zun, you must create a database, service credentials, and API endpoints.

1. To create the database, complete these steps:

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

     ```
     # mysql
     ```

   - Create the `zun` database:

     ```
     MariaDB [(none)] CREATE DATABASE zun;
     ```

   - Grant proper access to the `zun` database:

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

     Replace `ZUN_DBPASS` with a suitable password.

   - 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:

   - Create the `zun` user:

     ```
     $ openstack user create --domain default --password-prompt zun
     User Password:
     Repeat User Password:
     +-----------+----------------------------------+
     | Field     | Value                            |
     +-----------+----------------------------------+
     | domain_id | e0353a670a9e496da891347c589539e9 |
     | enabled   | True                             |
     | id        | ca2e175b851943349be29a328cc5e360 |
     | name      | zun                              |
     +-----------+----------------------------------+
     ```

   - Add the `admin` role to the `zun` user:

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

     Note: This command provides no output.

   - Create the `zun` service entities:

```
$ openstack service create --name zun \
    --description "Container Service" container
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description | Container Service                |
| enabled     | True                             |
| id          | 727841c6f5df4773baa4e8a5ae7d72eb |
| name        | zun                              |
| type        | container                        |
+-------------+----------------------------------+
```

4. Create the Container service API endpoints:

```
$ openstack endpoint create --region RegionOne \
    container public http://controller:9517/v1
+--------------+------------------------------------------+
| Field        | Value                                    |
+--------------+------------------------------------------+
| enabled      | True                                     |
| id           | 3f4dab34624e4be7b000265f25049609         |
| interface    | public                                   |
| region       | RegionOne                                |
| region_id    | RegionOne                                |
| service_id   | 727841c6f5df4773baa4e8a5ae7d72eb         |
| service_name | zun                                      |
| service_type | container                                |
| url          | http://controller:9517/v1                |
+--------------+------------------------------------------+

$ openstack endpoint create --region RegionOne \
    container internal http://controller:9517/v1
+--------------+------------------------------------------+
| Field        | Value                                    |
+--------------+------------------------------------------+
| enabled      | True                                     |
| id           | 9489f78e958e45cc85570fec7e836d98         |
| interface    | internal                                 |
| region       | RegionOne                                |
| region_id    | RegionOne                                |
| service_id   | 727841c6f5df4773baa4e8a5ae7d72eb         |
| service_name | zun                                      |
| service_type | container                                |
| url          | http://controller:9517/v1                |
+--------------+------------------------------------------+

$ openstack endpoint create --region RegionOne \
    container admin http://controller:9517/v1
+--------------+------------------------------------------+
| Field        | Value                                    |
```

(continues on next page)

(continued from previous page)

```
+--------------+------------------------------------------+
| enabled      | True                                     |
| id           | 76091559514b40c6b7b38dde790efe99         |
| interface    | admin                                    |
| region       | RegionOne                                |
| region_id    | RegionOne                                |
| service_id   | 727841c6f5df4773baa4e8a5ae7d72eb         |
| service_name | zun                                      |
| service_type | container                                |
| url          | http://controller:9517/v1                |
+--------------+------------------------------------------+
```

Install and configure components

1. Create zun user and necessary directories:

   - Create user:

   ```
   # groupadd --system zun
   # useradd --home-dir "/var/lib/zun" \
         --create-home \
         --system \
         --shell /bin/false \
         -g zun \
         zun
   ```

   - Create directories:

   ```
   # mkdir -p /etc/zun
   # chown zun:zun /etc/zun
   ```

2. Install the following dependencies:

   For Ubuntu, run:

   ```
   # apt-get install python3-pip git
   ```

   For CentOS, run:

   ```
   # yum install python3-pip git python3-devel libffi-devel gcc openssl-devel
   ```

3. Clone and install zun:

   ```
   # cd /var/lib/zun
   # git clone https://opendev.org/openstack/zun.git
   # chown -R zun:zun zun
   # cd zun
   # pip3 install -r requirements.txt
   # python3 setup.py install
   ```

4. Generate a sample configuration file:

```
# su -s /bin/sh -c "oslo-config-generator \
    --config-file etc/zun/zun-config-generator.conf" zun
# su -s /bin/sh -c "cp etc/zun/zun.conf.sample \
    /etc/zun/zun.conf" zun
```

5. Copy api-paste.ini:

```
# su -s /bin/sh -c "cp etc/zun/api-paste.ini /etc/zun" zun
```

6. Edit the `/etc/zun/zun.conf`:

   - 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 `[api]` section, configure the IP address that Zun API server is going to listen:

     ```
     [api]
     ...
     host_ip = 10.0.0.11
     port = 9517
     ```

     Replace `10.0.0.11` with the management interface IP address of the controller node if different.

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

     ```
     [database]
     ...
     connection = mysql+pymysql://zun:ZUN_DBPASS@controller/zun
     ```

     Replace `ZUN_DBPASS` with the password you chose for the zun database.

   - In the `[keystone_auth]` section, configure Identity service access:

     ```
     [keystone_auth]
     memcached_servers = controller:11211
     www_authenticate_uri = http://controller:5000
     project_domain_name = default
     project_name = service
     user_domain_name = default
     password = ZUN_PASS
     username = zun
     auth_url = http://controller:5000
     auth_type = password
     auth_version = v3
     auth_protocol = http
     service_token_roles_required = True
     endpoint_type = internalURL
     ```

- In the `[keystone_authtoken]` section, configure Identity service access:

```
[keystone_authtoken]
...
memcached_servers = controller:11211
www_authenticate_uri = http://controller:5000
project_domain_name = default
project_name = service
user_domain_name = default
password = ZUN_PASS
username = zun
auth_url = http://controller:5000
auth_type = password
auth_version = v3
auth_protocol = http
service_token_roles_required = True
endpoint_type = internalURL
```

  Replace ZUN_PASS with the password you chose for the zun user in the Identity service.

- In the `[oslo_concurrency]` section, configure the `lock_path`:

```
[oslo_concurrency]
...
lock_path = /var/lib/zun/tmp
```

- In the `[oslo_messaging_notifications]` section, configure the `driver`:

```
[oslo_messaging_notifications]
...
driver = messaging
```

- In the `[websocket_proxy]` section, configure the IP address that the websocket proxy is going to listen to:

```
[websocket_proxy]
...
wsproxy_host = 10.0.0.11
wsproxy_port = 6784
base_url = ws://controller:6784/
```

  Note: This `base_url` will be used by end users to access the console of their containers so make sure this URL is accessible from your intended users and the port `6784` is not blocked by firewall.

  Replace `10.0.0.11` with the management interface IP address of the controller node if different.

Note: Make sure that `/etc/zun/zun.conf` still have the correct permissions. You can set the permissions again with:

```
# chown zun:zun /etc/zun/zun.conf
```

7. Populate Zun database:

```
# su -s /bin/sh -c "zun-db-manage upgrade" zun
```

Finalize installation

1. Create an upstart config, it could be named as `/etc/systemd/system/zun-api.service`:

Note: CentOS might install binary files into `/usr/bin/`. If it does, replace `/usr/local/bin/` directory with the correct in the following example files.

```
[Unit]
Description = OpenStack Container Service API

[Service]
ExecStart = /usr/local/bin/zun-api
User = zun

[Install]
WantedBy = multi-user.target
```

2. Create an upstart config, it could be named as `/etc/systemd/system/zun-wsproxy.service`:

```
[Unit]
Description = OpenStack Container Service Websocket Proxy

[Service]
ExecStart = /usr/local/bin/zun-wsproxy
User = zun

[Install]
WantedBy = multi-user.target
```

3. Enable and start zun-api and zun-wsproxy:

```
# systemctl enable zun-api
# systemctl enable zun-wsproxy
```

```
# systemctl start zun-api
# systemctl start zun-wsproxy
```

4. Verify that zun-api and zun-wsproxy services are running:

```
# systemctl status zun-api
# systemctl status zun-wsproxy
```

Install and configure a compute node

This section describes how to install and configure the Compute service on a compute node.

Note: This section assumes that you are following the instructions in this guide step-by-step to configure the first compute node. If you want to configure additional compute nodes, prepare them in a similar fashion. Each additional compute node requires a unique IP address.

Prerequisites

Before you install and configure Zun, you must have Docker and Kuryr-libnetwork installed properly in the compute node, and have Etcd installed properly in the controller node. Refer Get Docker for Docker installation and Kuryr libnetwork installation guide, Etcd installation guide

Install and configure components

1. Create zun user and necessary directories:

   - Create user:

   ```
   # groupadd --system zun
   # useradd --home-dir "/var/lib/zun" \
         --create-home \
         --system \
         --shell /bin/false \
         -g zun \
         zun
   ```

   - Create directories:

   ```
   # mkdir -p /etc/zun
   # chown zun:zun /etc/zun
   ```

   - Create CNI directories:

   ```
   # mkdir -p /etc/cni/net.d
   # chown zun:zun /etc/cni/net.d
   ```

2. Install the following dependencies:

   For Ubuntu, run:

   ```
   # apt-get install python3-pip git numactl
   ```

   For CentOS, run:

   ```
   # yum install python3-pip git python3-devel libffi-devel gcc openssl-
   ↪devel numactl
   ```

3. Clone and install zun:

```
# cd /var/lib/zun
# git clone https://opendev.org/openstack/zun.git
# chown -R zun:zun zun
# cd zun
# pip3 install -r requirements.txt
# python3 setup.py install
```

4. Generate a sample configuration file:

```
# su -s /bin/sh -c "oslo-config-generator \
    --config-file etc/zun/zun-config-generator.conf" zun
# su -s /bin/sh -c "cp etc/zun/zun.conf.sample \
    /etc/zun/zun.conf" zun
# su -s /bin/sh -c "cp etc/zun/rootwrap.conf \
    /etc/zun/rootwrap.conf" zun
# su -s /bin/sh -c "mkdir -p /etc/zun/rootwrap.d" zun
# su -s /bin/sh -c "cp etc/zun/rootwrap.d/* \
    /etc/zun/rootwrap.d/" zun
# su -s /bin/sh -c "cp etc/cni/net.d/* /etc/cni/net.d/" zun
```

5. Configure sudoers for zun users:

Note: CentOS might install binary files into /usr/bin/. If it does, replace /usr/local/bin/ directory with the correct in the following command.

```
# echo "zun ALL=(root) NOPASSWD: /usr/local/bin/zun-rootwrap \
    /etc/zun/rootwrap.conf *" | sudo tee /etc/sudoers.d/zun-rootwrap
```

6. Edit the /etc/zun/zun.conf:

   - 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] section, configure the path that is used by Zun to store the states:

     ```
     [DEFAULT]
     ...
     state_path = /var/lib/zun
     ```

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

     ```
     [database]
     ...
     connection = mysql+pymysql://zun:ZUN_DBPASS@controller/zun
     ```

     Replace ZUN_DBPASS with the password you chose for the zun database.

- In the `[keystone_auth]` section, configure Identity service access:

```
[keystone_auth]
memcached_servers = controller:11211
www_authenticate_uri = http://controller:5000
project_domain_name = default
project_name = service
user_domain_name = default
password = ZUN_PASS
username = zun
auth_url = http://controller:5000
auth_type = password
auth_version = v3
auth_protocol = http
service_token_roles_required = True
endpoint_type = internalURL
```

- In the `[keystone_authtoken]` section, configure Identity service access:

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

  Replace ZUN_PASS with the password you chose for the zun user in the Identity service.

- In the `[oslo_concurrency]` section, configure the `lock_path`:

```
[oslo_concurrency]
...
lock_path = /var/lib/zun/tmp
```

- (Optional) If you want to run both containers and nova instances in this compute node, in the `[compute]` section, configure the `host_shared_with_nova`:

```
[compute]
...
host_shared_with_nova = true
```

Note: Make sure that `/etc/zun/zun.conf` still have the correct permissions. You can set the permissions again with:

chown zun:zun /etc/zun/zun.conf

---

7. Configure Docker and Kuryr:

- Create the directory `/etc/systemd/system/docker.service.d`

```
# mkdir -p /etc/systemd/system/docker.service.d
```

- Create the file `/etc/systemd/system/docker.service.d/docker.conf`. Configure docker to listen to port 2375 as well as the default unix socket. Also, configure docker to use etcd3 as storage backend:

```
[Service]
ExecStart=
ExecStart=/usr/bin/dockerd --group zun -H tcp://compute1:2375 -H␣
↪unix:///var/run/docker.sock --cluster-store etcd://controller:2379
```

- Restart Docker:

```
# systemctl daemon-reload
# systemctl restart docker
```

- Edit the Kuryr config file `/etc/kuryr/kuryr.conf`. Set `capability_scope` to `global` and `process_external_connectivity` to `False`:

```
[DEFAULT]
...
capability_scope = global
process_external_connectivity = False
```

- Restart Kuryr-libnetwork:

```
# systemctl restart kuryr-libnetwork
```

8. Configure containerd:

- Generate config file for containerd:

```
# containerd config default > /etc/containerd/config.toml
```

- Edit the `/etc/containerd/config.toml`. In the `[grpc]` section, configure the `gid` as the group ID of the `zun` user:

```
[grpc]
  ...
  gid = ZUN_GROUP_ID
```

Replace `ZUN_GROUP_ID` with the real group ID of `zun` user. You can retrieve the ID by (for example):

```
# getent group zun | cut -d: -f3
```

Note: Make sure that `/etc/containerd/config.toml` still have the correct permissions. You can set the permissions again with:

chown zun:zun /etc/containerd/config.toml

- Restart containerd:

```
# systemctl restart containerd
```

9. Configure CNI:

- Download and install the standard loopback plugin:

```
# mkdir -p /opt/cni/bin
# curl -L https://github.com/containernetworking/plugins/releases/
↪download/v0.7.1/cni-plugins-amd64-v0.7.1.tgz \
      | tar -C /opt/cni/bin -xzvf - ./loopback
```

- Install the Zun CNI plugin:

```
# install -o zun -m 0555 -D /usr/local/bin/zun-cni /opt/cni/bin/zun-
↪cni
```

Note: CentOS might install binary files into `/usr/bin/`. If it does, replace `/usr/local/bin/zun-cni` with the correct path in the command above.

Finalize installation

1. Create an upstart config for zun compute, it could be named as `/etc/systemd/system/zun-compute.service`:

Note: CentOS might install binary files into `/usr/bin/`. If it does, replace `/usr/local/bin/` directory with the correct in the following example file.

```
[Unit]
Description = OpenStack Container Service Compute Agent

[Service]
ExecStart = /usr/local/bin/zun-compute
User = zun

[Install]
WantedBy = multi-user.target
```

2. Create an upstart config for zun cni daemon, it could be named as `/etc/systemd/system/zun-cni-daemon.service`:

Note: CentOS might install binary files into `/usr/bin/`, If it does, replace `/usr/local/bin/` directory with the correct in the following example file.

```
[Unit]
Description = OpenStack Container Service CNI daemon

[Service]
ExecStart = /usr/local/bin/zun-cni-daemon
User = zun

[Install]
WantedBy = multi-user.target
```

3. Enable and start zun-compute:

```
# systemctl enable zun-compute
# systemctl start zun-compute
```

4. Enable and start zun-cni-daemon:

```
# systemctl enable zun-cni-daemon
# systemctl start zun-cni-daemon
```

5. Verify that zun-compute and zun-cni-daemon services are running:

```
# systemctl status zun-compute
# systemctl status zun-cni-daemon
```

Enable Kata Containers (Optional)

By default, `runc` is used as the container runtime. If you want to use Kata Containers instead, this section describes the additional configuration steps.

Note: Kata Containers requires nested virtualization or bare metal. See the official document for details.

---

1. Enable the repository for Kata Containers:

   For Ubuntu, run:

```
# curl -sL http://download.opensuse.org/repositories/home:/
↪katacontainers:/releases:/$(arch):/master/xUbuntu_$(lsb_release -rs)/
↪Release.key | apt-key add -
# add-apt-repository "deb http://download.opensuse.org/repositories/home:/
↪katacontainers:/releases:/$(arch):/master/xUbuntu_$(lsb_release -rs)/ /"
```

   For CentOS, run:

```
# yum-config-manager --add-repo "http://download.opensuse.org/
↪repositories/home:/katacontainers:/releases:/$(arch):/master/CentOS_7/
↪home:katacontainers:releases:$(arch):master.repo"
```

2. Install Kata Containers:

   For Ubuntu, run:

```
# apt-get update
# apt install kata-runtime kata-proxy kata-shim
```

For CentOS, run:

```
# yum install kata-runtime kata-proxy kata-shim
```

3. Configure Docker to add Kata Container as runtime:

   - Edit the file `/etc/systemd/system/docker.service.d/docker.conf`. Append `--add-runtime` option to add kata-runtime to Docker:

   ```
   [Service]
   ExecStart=
   ExecStart=/usr/bin/dockerd --group zun -H tcp://compute1:2375 -H␣
   ↪unix:///var/run/docker.sock --cluster-store etcd://controller:2379␣
   ↪--add-runtime kata=/usr/bin/kata-runtime
   ```

   - Restart Docker:

   ```
   # systemctl daemon-reload
   # systemctl restart docker
   ```

4. Configure containerd to add Kata Containers as runtime:

   - Edit the `/etc/containerd/config.toml`. In the `[plugins.cri.containerd]` section, add the kata runtime configuration:

   ```
   [plugins]
     ...
     [plugins.cri]
       ...
       [plugins.cri.containerd]
         ...
         [plugins.cri.containerd.runtimes.kata]
           runtime_type = "io.containerd.kata.v2"
   ```

   - Restart containerd:

   ```
   # systemctl restart containerd
   ```

5. Configure Zun to use Kata runtime:

   - Edit the `/etc/zun/zun.conf`. In the `[DEFAULT]` section, configure `container_runtime` as kata:

   ```
   [DEFAULT]
   ...
   container_runtime = kata
   ```

   - Restart zun-compute:

   ```
   # systemctl restart zun-compute
   ```

Verify operation

Verify operation of the Container service.

Note: Perform these commands on the controller node.

1. Install python-zunclient:

```
#  pip3 install python-zunclient
```

2. Source the `admin` tenant credentials:

```
$ . admin-openrc
```

3. List service components to verify successful launch and registration of each process:

```
$ openstack appcontainer service list
+----+------------------------+-------------+-------+----------+-----------
↪------+----------------------------+--------------------+
| Id | Host                   | Binary      | State | Disabled | Disabled␣
↪Reason | Updated At                 | Availability Zone  |
+----+------------------------+-------------+-------+----------+-----------
↪------+----------------------------+--------------------+
|  1 | localhost.localdomain | zun-compute | up    | False    | None      ␣
↪      | 2018-03-13 14:15:40+00:00 | nova               |
+----+------------------------+-------------+-------+----------+-----------
↪------+----------------------------+--------------------+
```

Launch a container

In environments that include the Container service, you can launch a container.

1. Source the `demo` credentials to perform the following steps as a non-administrative project:

```
$ . demo-openrc
```

2. Determine available networks.

```
$ openstack network list
+--------------------------------------+-------------+--------------------
↪------------------+
| ID                                   | Name        | Subnets            ␣
↪                  |
+--------------------------------------+-------------+--------------------
↪------------------+
| 4716ddfe-6e60-40e7-b2a8-42e57bf3c31c | selfservice | 2112d5eb-f9d6-45fd-
↪906e-7cabd38b7c7c |
| b5b6993c-ddf9-40e7-91d0-86806a42edb8 | provider    | 310911f6-acf0-4a47-
↪824e-3032916582ff |
+--------------------------------------+-------------+--------------------
↪------------------+
```

(continues on next page)

(continued from previous page)

```
```

Note: This output may differ from your environment.

3. Set the `NET_ID` environment variable to reflect the ID of a network. For example, using the self-service network:

```
$ export NET_ID=$(openstack network list | awk '/ selfservice / { print
↪$2 }')
```

4. Run a CirrOS container on the selfservice network:

```
$ openstack appcontainer run --name container --net network=$NET_ID␣
↪cirros ping 8.8.8.8
```

5. After a short time, verify successful creation of the container:

```
$ openstack appcontainer list
+--------------------------------------+-----------+--------+---------+---
↪---------+------------------------------------------------------+-------+
| uuid                                 | name      | image  | status  |␣
↪task_state | addresses                                            | ports |
+--------------------------------------+-----------+--------+---------+---
↪---------+------------------------------------------------------+-------+
| 4ec10d48-1ed8-492a-be5a-402be0abc66a | container | cirros | Running |␣
↪None       | 10.0.0.11, fd13:fd51:ebe8:0:f816:3eff:fe9c:7612 | []    |
+--------------------------------------+-----------+--------+---------+---
↪---------+------------------------------------------------------+-------+
```

6. Access the container and verify access to the internet:

```
$ openstack appcontainer exec --interactive container /bin/sh
# ping -c 4 openstack.org
# exit
```

7. Stop and delete the container.

```
$ openstack appcontainer stop container
$ openstack appcontainer delete container
```

Next steps

Your OpenStack environment now includes the zun service.

To add more services, see the additional documentation on installing OpenStack .

To learn more about the zun service, read the Zun developer documentation.

This chapter assumes a working setup of OpenStack following the OpenStack Installation Tutorial.

CHAPTER

FOUR

FOR CONTRIBUTORS

If you are new to Zun, the developer quick-start guide should help you quickly setup the development environment and get started. There are also a number of technical references on various topics collected in contributors guide.

4.1 Developer Quick-Start

This is a quick walkthrough to get you started developing code for Zun. This assumes you are already familiar with submitting code reviews to an OpenStack project.

See also:

https://docs.openstack.org/infra/manual/developers.html

4.1.1 Exercising the Services Using Devstack

This session has been tested on Ubuntu 16.04 (Xenial) only.

Clone devstack:

```
# Create a root directory for devstack if needed
$ sudo mkdir -p /opt/stack
$ sudo chown $USER /opt/stack
$ git clone https://opendev.org/openstack/devstack /opt/stack/devstack
```

We will run devstack with minimal local.conf settings required to enable required OpenStack services:

```
$ HOST_IP=<your ip>
$ git clone https://opendev.org/openstack/zun /opt/stack/zun
$ cat /opt/stack/zun/devstack/local.conf.sample \
    | sed "s/HOST_IP=.*/HOST_IP=$HOST_IP/" \
    > /opt/stack/devstack/local.conf
```

Note: By default, *KURYR_CAPABILITY_SCOPE=global*. It will work in both all-in-one and multi-node scenario. You still can change it to *local* (in **all-in-one scenario only**):

```
$ sed -i "s/KURYR_CAPABILITY_SCOPE=.*/KURYR_CAPABILITY_SCOPE=local/" /opt/
↪stack/devstack/local.conf
```

More devstack configuration information can be found at Devstack Configuration

More neutron configuration information can be found at Devstack Neutron Configuration

Run devstack:

```
$ cd /opt/stack/devstack
$ ./stack.sh
```

Note: If the developer have a previous devstack environment and they want to re-stack the environment, they need to uninstall the pip packages before restacking:

```
$ ./unstack.sh
$ ./clean.sh
$ pip freeze | grep -v '^\-e' | xargs sudo pip uninstall -y
$ ./stack.sh
```

Prepare your session to be able to use the various openstack clients including nova, neutron, and glance. Create a new shell, and source the devstack openrc script:

```
$ source /opt/stack/devstack/openrc admin admin
```

4.1.2 Using the service

We will create and run a container that pings the address 8.8.8.8 four times:

```
$ zun run --name test cirros ping -c 4 8.8.8.8
```

Above command will use the Docker image `cirros` from DockerHub which is a public image repository. Alternatively, you can use Docker image from Glance which serves as a private image repository:

```
$ docker pull cirros
$ docker save cirros | openstack image create cirros --public --container-
↪format docker --disk-format raw
$ zun run --image-driver glance cirros ping -c 4 8.8.8.8
```

You should see a similar output to:

```
$ zun list
  +--------------------------------------+------+--------+---------+----------
↪--+------------+-------+
  | uuid                                 | name | image  | status  | task_
↪state | addresses  | ports |
  +--------------------------------------+------+--------+---------+----------
↪--+------------+-------+
  | 46dd001b-7474-412c-a0f4-7adc047aaedf | test | cirros | Stopped | None    ␣
↪  | 172.17.0.2 | []    |
  +--------------------------------------+------+--------+---------+----------
↪--+------------+-------+
```

(continues on next page)

(continued from previous page)

```
$ zun logs test
  PING 8.8.8.8 (8.8.8.8): 56 data bytes
  64 bytes from 8.8.8.8: seq=0 ttl=40 time=25.513 ms
  64 bytes from 8.8.8.8: seq=1 ttl=40 time=25.348 ms
  64 bytes from 8.8.8.8: seq=2 ttl=40 time=25.226 ms
  64 bytes from 8.8.8.8: seq=3 ttl=40 time=25.275 ms

  --- 8.8.8.8 ping statistics ---
  4 packets transmitted, 4 packets received, 0% packet loss
  round-trip min/avg/max = 25.226/25.340/25.513 ms
```

Delete the container:

```
$ zun delete test
```

4.2 Contributors Guide

In this section you will find information on how to contribute to Zun. Content includes architectural overviews, tips and tricks for setting up a development environment, and information on Cinders lower level programming APIs.

4.2.1 HowTos and Tutorials

If you are new to Zun, this section contains information that should help you quickly get started.

There are documents that should help you develop and contribute to the project.

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 Zun.

Communication

- IRC channel: #openstack-zun
- Mailing lists prefix: [zun]
- Office Hours:

  This is general Zun team meeting. Anyone can bring up a topic to discuss with the Zun team.

  - time: http://eavesdrop.openstack.org/#Zun_Team_Meeting

Contacting the Core Team

The list of current Zun core reviewers is available on gerrit.

New Feature Planning

Zun team uses Launchpad to propose new features. A blueprint should be submitted in Launchpad first. Such blueprints need to be discussed and approved by the Zun driver team

Task Tracking

We track our tasks in Launchpad

> https://bugs.launchpad.net/zun

If youre looking for some smaller, easier work item to pick up and get started on, search for the low-hanging-fruit tag.

Reporting a Bug

You found an issue and want to make sure we are aware of it? You can do so on Launchpad.

Getting Your Patch Merged

All changes proposed to the Zun project require one or two +2 votes from Zun core reviewers before one of the core reviewers can approve patch by giving `Workflow +1` vote.

Project Team Lead Duties

All common PTL duties are enumerated in the PTL guide.

Installing the API via WSGI

This document provides two WSGI deployments as examples: uwsgi and mod_wsgi.

See also:

https://governance.openstack.org/tc/goals/pike/deploy-api-in-wsgi.html#uwsgi-vs-mod-wsgi

Installing the API behind mod_wsgi

Zun comes with a few example files for configuring the API service to run behind Apache with `mod_wsgi`.

app.wsgi

The file `zun/api/app.wsgi` sets up the V2 API WSGI application. The file is installed with the rest of the zun application code, and should not need to be modified.

etc/apache2/zun.conf

The `etc/apache2/zun.conf` file contains example settings that work with a copy of zun installed via devstack.

```
# Licensed under the Apache License, Version 2.0 (the "License"); you may
# not use this file except in compliance with the License. You may obtain
# a copy of the License at
#
#      http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS, WITHOUT
# WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the
# License for the specific language governing permissions and limitations
# under the License.

# This is an example Apache2 configuration file for using the
# zun API through mod_wsgi.

# Note: If you are using a Debian-based system then the paths
# "/var/log/httpd" and "/var/run/httpd" will use "apache2" instead
# of "httpd".
#
# The number of processes and threads is an example only and should
# be adjusted according to local requirements.

Listen %PUBLICPORT%
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-agent}i\" %D(us)
↪" zun_combined

<VirtualHost *:%PUBLICPORT%>
    WSGIDaemonProcess zun-api user=%USER% processes=5 threads=1 display-name=%
↪{GROUP}
    WSGIScriptAlias / %PUBLICWSGI%
    WSGIProcessGroup zun-api
    ErrorLogFormat "%M"
    ErrorLog /var/log/%APACHE_NAME%/zun_api.log
    LogLevel info
    CustomLog /var/log/%APACHE_NAME%/zun_access.log zun_combined
```

(continues on next page)

(continued from previous page)

```
    <Directory /opt/stack/zun/zun/api>
        WSGIProcessGroup zun-api
        WSGIApplicationGroup %{GLOBAL}
        AllowOverride All
        Require all granted
    </Directory>
</VirtualHost>
```

1. On deb-based systems copy or symlink the file to `/etc/apache2/sites-available`. For rpm-based systems the file will go in `/etc/httpd/conf.d`.
2. Modify the `WSGIDaemonProcess` directive to set the `user` and `group` values to an appropriate user on your server. In many installations `zun` will be correct. Modify the `WSGIScriptAlias` directive to set the path of the wsgi script. If you are using devstack, the value should be `/opt/stack/zun/zun/api/app.wsgi`. In the `ErrorLog` and `CustomLog` directives, replace `%APACHE_NAME%` with `apache2`.
3. Enable the zun site. On deb-based systems:

   ```
   $ a2ensite zun
   $ service apache2 reload
   ```

   On rpm-based systems:

   ```
   $ service httpd reload
   ```

Installing the API with uwsgi

Create zun-uwsgi.ini file:

```
[uwsgi]
http = 0.0.0.0:9517
wsgi-file = <path_to_zun>/zun/api/app.wsgi
plugins = python
# This is running standalone
master = true
# Set die-on-term & exit-on-reload so that uwsgi shuts down
exit-on-reload = true
die-on-term = true
# uwsgi recommends this to prevent thundering herd on accept.
thunder-lock = true
# Override the default size for headers from the 4k default. (mainly for␣
↪keystone token)
buffer-size = 65535
enable-threads = true
# Set the number of threads usually with the returns of command nproc
threads = 8
# Make sure the client doesn't try to re-use the connection.
add-header = Connection: close
```

(continues on next page)

(continued from previous page)

```
# Set uid and gip to a appropriate user on your server. In many
# installations ``zun`` will be correct.
uid = zun
gid = zun
```

Then start the uwsgi server:

```
uwsgi ./zun-uwsgi.ini
```

Or start in background with:

```
uwsgi -d ./zun-uwsgi.ini
```

Run tempest tests locally

This is a guide for developers who want to run tempest tests in their local machine.

Zun contains a suite of tempest tests in the zun/tests/tempest directory. Tempest tests are primary for testing integration between Zun and its depending software stack (i.e. Docker, other OpenStack services). Any proposed code change will be automatically rejected by the gate if the change causes tempest test failures. If this happens, contributors are suggested to refer this document to re-run the tests locally and perform any necessary trouble-shooting.

Prerequisite

You need to deploy Zun in a devstack environment.

Refer the `Exercising the Services Using Devstack` session at Developer Quick-Start Guide for details.

Run the test

Edit `/opt/stack/tempest/etc/tempest.conf`:

- Add the `[container_service]` section, configure `min_microversion` and `max_microversion`:

```
[container_service]
min_microversion=1.26
max_microversion=1.26
```

Note: You might need to modify the min/max microversion based on your test environment.

Navigate to tempest directory:

```
cd /opt/stack/tempest
```

Run this command:

```
tempest run --regex zun_tempest_plugin.tests.tempest.api
```

To run a single test case, run with the test case name, for example:

```
tempest run --regex zun_tempest_plugin.tests.tempest.api.test_containers.
↪TestContainer.test_list_containers
```

Run unit tests

This is a guide for developers who want to run unit tests in their local machine.

Prerequisite

Zun source code should be pulled directly from git:

```
# from your home or source directory
cd ~
git clone https://opendev.org/openstack/zun
cd zun
```

Install the prerequisite packages listed in the `bindep.txt` file.

On Debian-based distributions (e.g., Debian/Mint/Ubuntu):

```
# Ubuntu/Debian (recommend Ubuntu 16.04):
sudo apt-get update
sudo apt-get install python-pip
sudo pip install tox
tox -e bindep
sudo apt-get install <indicated missing package names>
```

On Fedora-based distributions (e.g., Fedora/RHEL/CentOS/Scientific Linux):

```
sudo yum install python-pip
sudo pip install tox
tox -e bindep
sudo yum install <indicated missing package names>
```

On openSUSE-based distributions (SLES 12, openSUSE Leap 42.1 or Tumbleweed):

```
sudo zypper in python-pip
sudo pip install tox
tox -e bindep
sudo zypper in <indicated missing package names>
```

Running the tests

All unit tests should be run using tox. To run Zuns entire test suite:

```
# run all tests (unit and pep8)
tox
```

To run a specific test, use a positional argument for the unit tests:

```
# run a specific test for Python 2.7
tox -epy27 -- test_container
```

You may pass options to the test programs using positional arguments:

```
# run all the Python 2.7 unit tests (in parallel!)
tox -epy27 -- --parallel
```

To run only the pep8/flake8 syntax and style checks:

```
tox -epep8
```

Multi-host Devstack

This is a guide for developers who want to setup Zun in more than one hosts.

Prerequisite

You need to deploy Zun in a devstack environment in the first host.

Refer the `Exercising the Services Using Devstack` session at Developer Quick-Start Guide for details.

Enable additional zun host

Refer to the Multi-Node lab for more information.

On the second host, clone devstack:

```
# Create a root directory for devstack if needed
$ sudo mkdir -p /opt/stack
$ sudo chown $USER /opt/stack

$ git clone https://opendev.org/openstack/devstack /opt/stack/devstack
```

The second host will only need zun-compute service along with kuryr-libnetwork support. You also need to tell devstack where the SERVICE_HOST is:

```
$ SERVICE_HOST=<controller's ip>
$ HOST_IP=<your ip>
$ git clone https://opendev.org/openstack/zun /opt/stack/zun
```

(continues on next page)

(continued from previous page)

```
$ cat /opt/stack/zun/devstack/local.conf.subnode.sample \
    | sed "s/HOST_IP=.*/HOST_IP=$HOST_IP/" \
    | sed "s/SERVICE_HOST=.*/SERVICE_HOST=$SERVICE_HOST/" \
    > /opt/stack/devstack/local.conf
```

Run devstack:

```
$ cd /opt/stack/devstack
$ ./stack.sh
```

On the controller host, you can see 2 zun-compute hosts available:

```
$ zun service-list
+----+-------------+-------------+-------+----------+-----------------+-------
↪--------------------+----------------------------+
| Id | Host        | Binary      | State | Disabled | Disabled Reason |␣
↪Updated At                | Availability Zone               |
+----+-------------+-------------+-------+----------+-----------------+-------
↪--------------------+----------------------------+
| 1  | zun-hosts-1 | zun-compute | up    | False    | None            | 2018-
↪03-13 14:15:40+00:00 | Nova                        |
| 2  | zun-hosts-2 | zun-compute | up    | False    | None            | 2018-
↪03-13 14:15:41+00:00 | Nova                        |
+----+-------------+-------------+-------+----------+-----------------+-------
↪--------------------+----------------------------+
```

There are some other important documents also that helps new contributors to contribute effectively towards code standards to the project.

API Microversions

Background

Zun uses a framework we call 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 has as its value a string containing the name of the service, `container`, and a monotonically increasing semantic version number starting from `1.1`. The full form of the header takes the form:

```
OpenStack-API-Version: container 1.1
```

If a user makes a request without specifying a version, they will get the `BASE_VER` as defined in `zun/api/controllers/versions.py`. This value is currently `1.1` and is expected to remain so for quite a long time.

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 container/{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` container/{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 list of attributes and data structures accepted.

    Example: adding a new attribute locked: True/False to the request body
- the Response
  - the list of attributes and data structures returned

    Example: adding a new attribute locked: True/False to the output of container/{ID}
  - the allowed values of non free form fields

    Example: adding a new allowed `status` to container/{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.

    See[2] for the 400, 403, 404 and 415 cases.
  - changing a status code on a particular response

    Example: changing the return code of an API from 501 to 400.

    Note: Fixing a bug so that a 400+ code is returned rather than a 500 or 503 does not require a microversion change. Its assumed that clients are not expected to handle a 500 or 503 response and therefore should not need to opt-in to microversion changes that fixes a 500 or 503 response from happening. According to the OpenStack API Working Group, a **500**

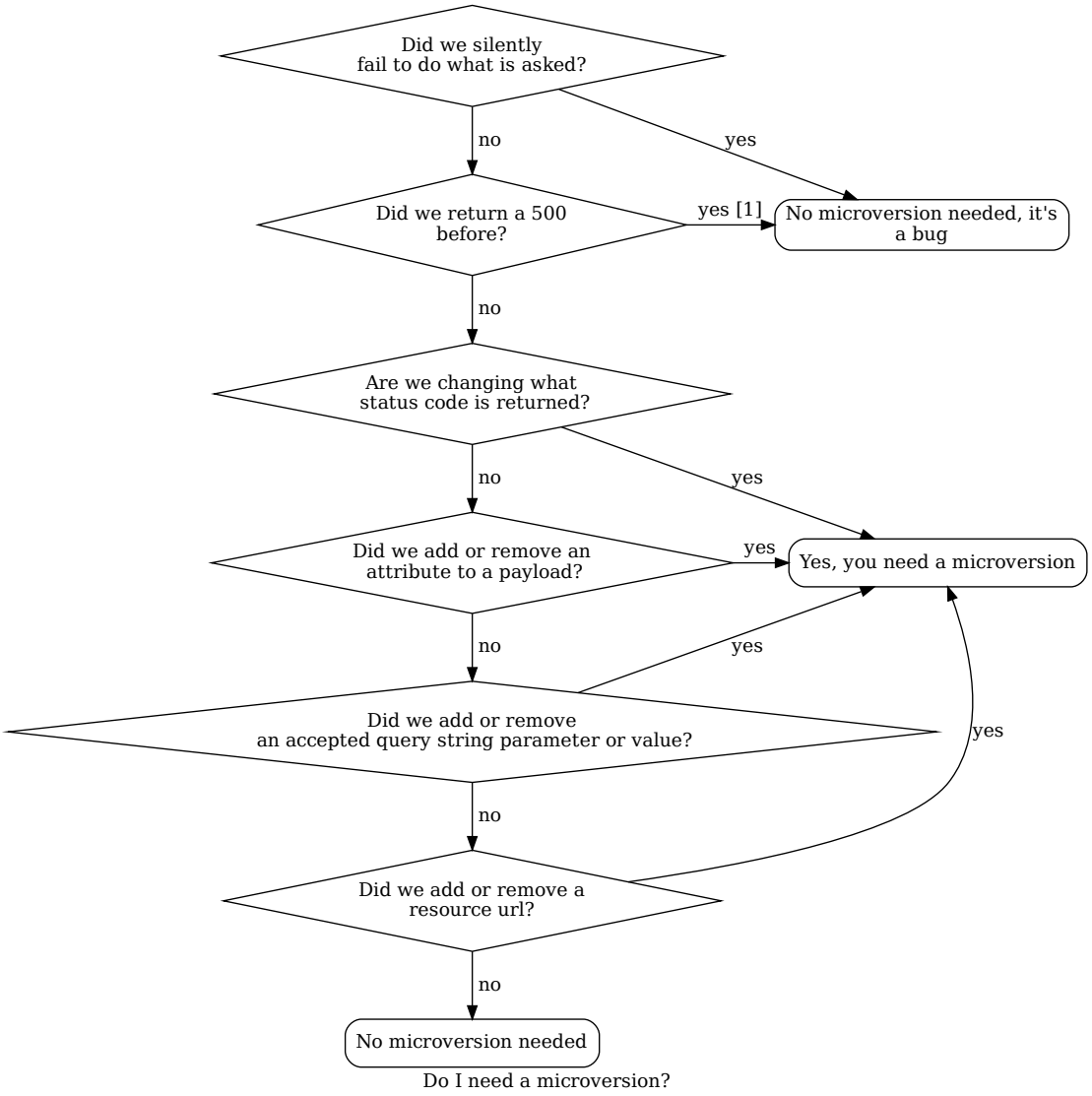
[2] The exception to not needing a microversion when returning a previously unspecified error code is the 400, 403, 404 and 415 cases. This is considered OK to return even if previously unspecified in the code since its implied given keystone authentication can fail with a 403 and API validation can fail with a 400 for invalid JSON request body. Request to url/resource that does not exist always fails with 404. Invalid content types are handled before API methods are called which results in a 415.

Note: When in doubt about whether or not a microversion is required for changing an error response code, consult the Zun Team.

Internal Server Error should **not** be returned to the user for failures due to user error that can be fixed by changing the request on the client side. See[1].

- new headers returned on a response

The following flow chart attempts to walk through the process of do we need a 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 (except in[7]).

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 Zun. 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 Zun versions, you probably dont need a microversion. If, on the other hand, there is any ambiguity, a microversion is probably needed.

When a microversion is not needed

A microversion is not needed in the following situation:

- the response
  - Changing the error message without changing the response code does not require a new microversion.
  - Removing an inapplicable HTTP header, for example, suppose the Retry-After HTTP header is being returned with a 4xx code. This header should only be returned with a 503 or 3xx response, so it may be removed without bumping the microversion.

In Code

In `zun/api/controllers/base.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:

```
@base.Controller.api_version("1.2")
def my_api_method(self, req, id):
    ....
```

This method would only be available if the caller had specified an `OpenStack-API-Version` of `>= 1.2`. If they had specified a lower version (or not specified it and received the default of `1.1`) the server would respond with `HTTP/406`.

Removing an API method

In the controller class:

```
@base.Controller.api_version("1.2", "1.3")
def my_api_method(self, req, id):
    ....
```

This method would only be available if the caller had specified an `OpenStack-API-Version` of `>= 1.2` and `OpenStack-API-Version` of `<= 1.3`. If `1.4` or later is specified the server will respond with `HTTP/406`.

Changing a methods behavior

In the controller class:

```
@base.Controller.api_version("1.2", "1.3")
def my_api_method(self, req, id):
    .... method_1 ...

@base.Controller.api_version("1.4")  # noqa
def my_api_method(self, req, id):
    .... method_2 ...
```

If a caller specified `1.2`, `1.3` (or received the default of `1.1`) they would see the result from `method_1`, and for `1.4` or later they would see the result from `method_2`.

It is vital that the two methods have the same name, so the second of them will need `# noqa` to avoid failing flake8s `F811` rule. The two methods may be different in any kind of semantics (schema validation, return values, response codes, etc)

When not using decorators

When you dont want to use the `@api_version` decorator on a method or you want to change behavior 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 accessed with `pecan.request`). Every API method has an versions object attached to the request object and that can be used to modify behavior based on its value:

```
def index(self):
    <common code>

    req_version = pecan.request.version
    req1_min = versions.Version('', '', '', "1.1")
    req1_max = versions.Version('', '', '', "1.5")
    req2_min = versions.Version('', '', '', "1.6")
    req2_max = versions.Version('', '', '', "1.10")

    if req_version.matches(req1_min, req1_max):
        ....stuff....
    elif req_version.matches(req2min, req2_max):
        ....other stuff....
    elif req_version > versions.Version("1.10"):
        ....more stuff.....

    <common code>
```

The first argument to the matches method is the minimum acceptable version and the second is maximum acceptable version. If the specified minimum version and maximum version are null then `ValueError` is returned.

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 `zun/api/controllers/versions.py`
- Update `CURRENT_MAX_VER` in `zun/api/controllers/versions.py`
- Add a verbose description to `zun/api/rest_api_version_history.rst`. There should be enough information that it could be used by the docs team for release notes.
- Update `min_microversion` in `.zuul.yaml`.
- Update the expected versions in affected tests, for example in `zun/tests/unit/api/controllers/test_root.py`.
- Update `CURRENT_VERSION` in `zun/tests/unit/api/base.py`.
- Make a new commit to python-zunclient and update corresponding files to enable the newly added microversion API.
- If the microversion changes the response schema, a new schema and test for the microversion must be added to Tempest.

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 zun spec for the change, the minor number of `CURRENT_MAX_VER` 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 `CURRENT_MAX_VER`.

Versioned Objects

Zun uses the oslo.versionedobjects library to construct an object model that can be communicated via RPC. These objects have a version history and functionality to convert from one version to a previous version. This allows for 2 different levels of the code to still pass objects to each other, as in the case of rolling upgrades.

Object Version Testing

In order to ensure object versioning consistency is maintained, oslo.versionedobjects has a fixture to aid in testing object versioning. oslo.versionedobjects.fixture.ObjectVersionChecker generates fingerprints of each object, which is a combination of the current version number of the object, along with a hash of the RPC-critical parts of the object (fields and remotable methods).

The tests hold a static mapping of the fingerprints of all objects. When an object is changed, the hash generated in the test will differ from that held in the static mapping. This will signal to the developer that the version of the object needs to be increased. Following this version increase, the fingerprint that is then generated by the test can be copied to the static mapping in the tests. This symbolizes that if the code change is approved, this is the new state of the object to compare against.

Object Change Example

The following example shows the unit test workflow when changing an object (Container was updated to hold a new foo field):

```
tox -e py27 zun.tests.unit.objects.test_objects
```

This results in a unit test failure with the following output:

```
testtools.matchers._impl.MismatchError: !=:
reference = {'Container': '1.0-35edde13ad178e9419e7ea8b6d580bcd'}
actual    = {'Container': '1.0-22b40e8eed0414561ca921906b189820'}
```

```
: Fields or remotable methods in some objects have changed. Make␣
↪sure the versions of the objects has been bumped, and update the␣
↪hashes in the static fingerprints tree (object_data). For more␣
↪information, read https://docs.openstack.org/zun/latest/.
```

This is an indication that me adding the foo field to Container means I need to bump the version of Container, so I increase the version and add a comment saying what I changed in the new version:

```
@base.ZunObjectRegistry.register
class Container(base.ZunPersistentObject, base.ZunObject,
                base.ZunObjectDictCompat):
    # Version 1.0: Initial version
    # Version 1.1: Add container_id column
    # Version 1.2: Add memory column
    # Version 1.3: Add task_state column
    # Version 1.4: Add cpu, workdir, ports, hostname and labels␣
↪columns
```

(continues on next page)

(continued from previous page)

```
    # Version 1.5: Add meta column
    # Version 1.6: Add addresses column
    # Version 1.7: Add host column
    # Version 1.8: Add restart_policy
    # Version 1.9: Add status_detail column
    # Version 1.10: Add tty, stdin_open
    # Version 1.11: Add image_driver
    VERSION = '1.11'
```

Now that I have updated the version, I will run the tests again and let the test tell me the fingerprint that I now need to put in the static tree:

```
testtools.matchers._impl.MismatchError: !=:
reference = {'Container': '1.10-35edde13ad178e9419e7ea8b6d580bcd'}
actual    = {'Container': '1.11-ddffeb42cb5472decab6d73534fe103f'}
```

I can now copy the new fingerprint needed (1.11-ddffeb42cb5472decab6d73534fe103f), to the object_data map within zun/tests/unit/objects/test_objects.py:

```
object_data = {
    'Container': '1.11-ddffeb42cb5472decab6d73534fe103f',
    'Image': '1.0-0b976be24f4f6ee0d526e5c981ce0633',
    'NUMANode': '1.0-cba878b70b2f8b52f1e031b41ac13b4e',
    'NUMATopology': '1.0-b54086eda7e4b2e6145ecb6ee2c925ab',
    'ResourceClass': '1.0-2c41abea55d0f7cb47a97bdb345b37fd',
    'ResourceProvider': '1.0-92b427359d5a4cf9ec6c72cbe630ee24',
    'ZunService': '1.0-2a19ab9987a746621b2ada02d8aadf22',
}
```

Running the unit tests now shows no failure.

If I did not update the version, and rather just copied the new hash to the object_data map, the review would show the hash (but not the version) was updated in object_data. At that point, a reviewer should point this out, and mention that the object version needs to be updated.

If a remotable method were added/changed, the same process is followed, because this will also cause a hash change.

4.2.2 Documentation Contribution

Contributing Documentation to Zun

Zuns documentation has been moved from the openstack-manuals repository to the `docs` directory in the Zun repository. This makes it even more important that Zun add and maintain good documentation.

This page provides guidance on how to provide documentation for those who may not have previously been active writing documentation for OpenStack.

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 Zuns 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. Currently, we do not have the build configured to treat warnings as errors, so it is also important to check the build output to ensure that no warnings are produced by Sphinx.

Note: Many Sphinx warnings result in improperly formatted pages being generated.

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 zun.conf.sample are generated and put into `doc/soure/_static`.

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 `gate-zun-docs-ubuntu-xenial!`

Once a patch is approved it is immediately released to the docs.openstack.org website and can be seen under Zuns Documentation Page at https://docs.openstack.org/zun/latest . When a new release is cut a snapshot of that documentation will be kept at https://docs.openstack.org/zun/<release> . Changes from master can be backported to previous branches if necessary.

Doc Directory Structure

The main location for Zuns documentation is the `doc/source` directory. The top level index file that is seen at https://docs.openstack.org/zun/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:

Zun Administration Documentation (source/admin)

This directory is intended to hold any documentation that is related to how to run or operate Zun.

Zun CLI Documentation (source/cli)

This directory is intended to hold any documentation that relates to Zuns 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.

Zun Configuration Documentation (source/configuration)

This directory is intended to hold any documentation that relates to how to configure Zun. It is intended that some of this content be automatically generated in the future. At the moment, however, it is not. Changes to configuration options for Zun or its drivers needs to be put under this directory.

Zun Contributor Documentation (source/contributor)

This directory is intended to hold any documentation that relates to how to contribute to Zun 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.

Zun Installation Documentation (source/install)

Introduction:

This directory is intended to hold any installation documentation for Zun. Documentation that explains how to bring Zun 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>*.

4.2.3 Other Resources

Project hosting with Launchpad

Launchpad hosts the Zun project. The Zun project homepage on Launchpad is http://launchpad.net/zun.

Mailing list

The mailing list email is `openstack@lists.openstack.org`. This is a common mailing list across the OpenStack projects. To participate in the mailing list:

#. Subscribe to the list at http://lists.openstack.org/cgi-bin/mailman/listinfo/openstack

The mailing list archives are at http://lists.openstack.org/pipermail/openstack/.

Bug tracking

Report Zun bugs at https://bugs.launchpad.net/zun

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.

Feature requests (Blueprints)

Zun uses Launchpad Blueprints to track feature requests. Blueprints are at https://blueprints.launchpad.net/zun.

Technical support (Answers)

Zun no longer uses Launchpad Answers to track Zun technical support questions.

Note that Ask OpenStack (which is not hosted on Launchpad) can be used for technical support requests.

Code Reviews with Gerrit

Zun 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 Zun repository will be ignored.*

See Gerrit Workflow Quick Reference for information about how to get started using Gerrit. See Development Workflow for more detailed documentation on how to work with Gerrit.

Continuous Integration with Jenkins

Zun uses a Jenkins server to automate development tasks.

Jenkins performs tasks such as:

gate-zun-pep8-ubuntu-xenial Run PEP8 checks on proposed code changes that have been reviewed.

gate-zun-python27-ubuntu-xenial Run unit tests using python2.7 on proposed code changes that have been reviewed.

gate-zun-python35 Run unit tests using python3.5 on proposed code changes that have been reviewed.

gate-zun-docs-ubuntu-xenial Build this documentation and push it to OpenStack Zun.

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

Zun 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:

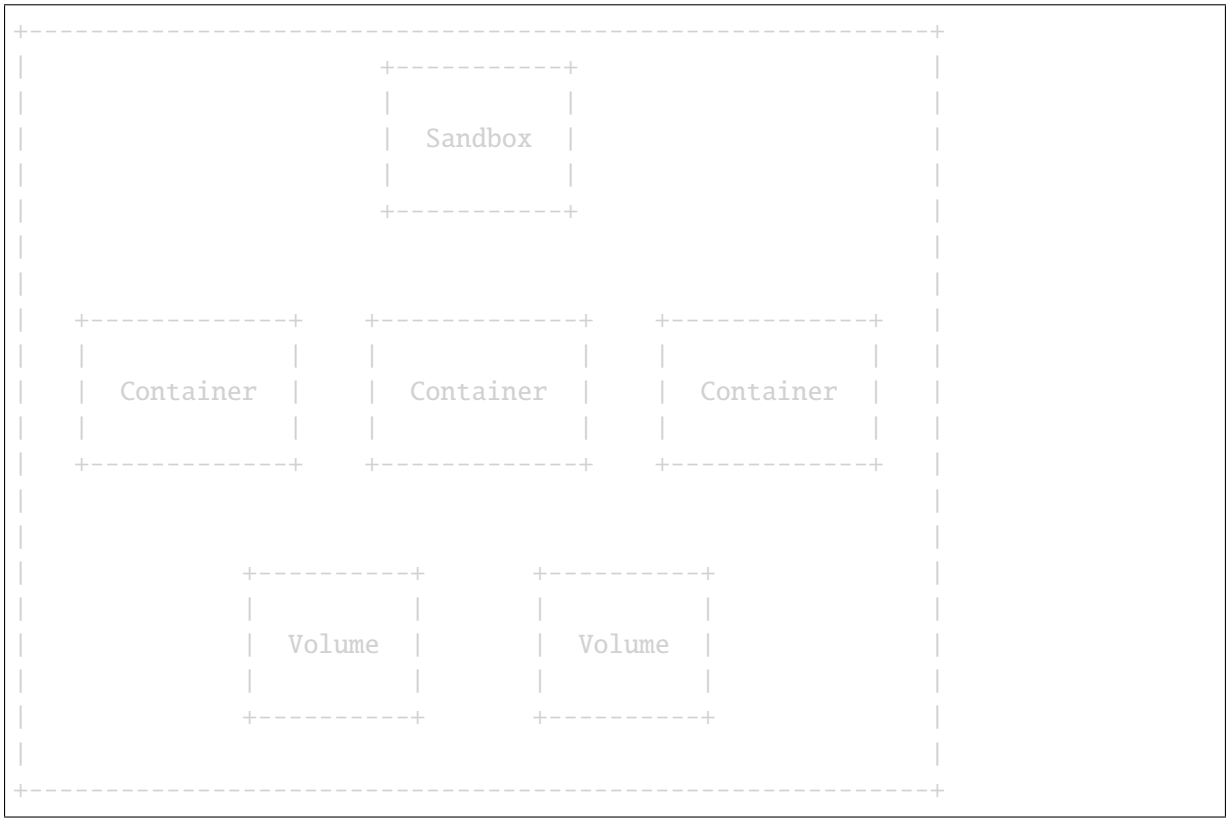
```
$ git commit  # Commit the change because reno scans git log.

$ tox -e releasenotes
```

Then look at the generated release notes files in releasenotes/build/html in your favorite browser.

Capsule Quick Start

Capsule is a container composition unit that includes sandbox container, multiple application containers and multiple volumes. All container inside the capsule share the same network, ipc, pid namespaces. In general, it is the same unit like Azure Container Instance(ACI) or Kubernetes Pod.

The diagram below is an overview of the structure of `capsule`.



Capsule API is currently in v1 phase now.

Now basic capsule functions are supported. Capsule API methods:

- Create: Create a capsule based on special yaml file or json file.
- Delete: Delete an existing capsule.
- Describe: Get detailed information about selected capsule.
- List: List all the capsules with essential fields.

Note: Volume is not yet supported, but it is in the roadmap. It will be implemented after Zun volume support has been finished.

If you need to access to the capsule port, you might need to open the port in security group rules and access the port via the floating IP that assigned to the capsule. The capsule example below assumes that a capsule has been launched with security group default and user want to access the port 22, 80 and 3306:

```
# use "-" because that the fields have many items
capsuleVersion: beta
kind: capsule
```

(continues on next page)

(continued from previous page)

```
metadata:
  name: template
  labels:
    app: web
    foo: bar
restartPolicy: Always
spec:
  containers:
  - image: ubuntu
    command:
      - "/bin/bash"
    imagePullPolicy: ifnotpresent
    workDir: /root
    ports:
      - name: ssh-port
        containerPort: 22
        hostPort: 22
        protocol: TCP
    resources:
      requests:
        cpu: 1
        memory: 1024
    env:
      ENV1: /usr/local/bin
      ENV2: /usr/sbin
    volumeMounts:
    - name: volume1
      mountPath: /data1
      readOnly: True
  - image: centos
    command:
      - "/bin/bash"
    args:
      - "-c"
      - "\"while true; do echo hello world; sleep 1; done\""
    imagePullPolicy: ifnotpresent
    workDir: /root
    ports:
      - name: nginx-port
        containerPort: 80
        hostPort: 80
        protocol: TCP
      - name: mysql-port
        containerPort: 3306
        hostPort: 3306
        protocol: TCP
    resources:
      requests:
        cpu: 1
```

(continues on next page)

(continued from previous page)

```
        memory: 1024
    env:
      ENV2: /usr/bin/
    volumeMounts:
    - name: volume2
      mountPath: /data2
    - name: volume3
      mountPath: /data3
  volumes:
  - name: volume1
    cinder:
      size: 5
      autoRemove: True
  - name: volume2
    cinder:
      volumeID: 9f81cbb2-10f9-4bab-938d-92fe33c57a24
  - name: volume3
    cinder:
      volumeID: 67618d54-dd55-4f7e-91b3-39ffb3ba7f5f
```

Pay attention, the volume2 and volume3 referred in the above yaml are already created by Cinder. Also capsule doesnt support Cinder multiple attach now. One volume only could be attached to one Container.

Capsule management commands in details:

Create capsule, it will create capsule based on capsule.yaml:

```
$ source ~/devstack/openrc demo demo
$ zun capsule-create -f capsule.yaml
```

If you want to get access to the port, you need to set the security group rules for it.

```
$ openstack security group rule create default \
  --protocol tcp --dst-port 3306:3306 --remote-ip 0.0.0.0/0
$ openstack security group rule create default \
  --protocol tcp --dst-port 80:80 --remote-ip 0.0.0.0/0
$ openstack security group rule create default \
  --protocol tcp --dst-port 22:22 --remote-ip 0.0.0.0/0
```

Delete capsule:

```
$ zun capsule-delete <uuid>
$ zun capsule-delete <capsule-name>
```

List capsule:

```
$ zun capsule-list
```

Describe capsule:

```
$ zun capsule-describe <uuid>
$ zun capsule-describe <capsule-name>
```

TODO

Add security group set to Capsule Build this documentation and push it to .

Add Gophercloud support for Capsule See Gophercloud support for Zun

Add Kubernetes connect to Capsule see zun connector for k8s.

Technical Vision for Zun

This document is a self-evaluation of Zun with regard to the Technical Committees technical vision.

Mission Statement

Zuns mission is to provide an OpenStack containers service that integrates with various container technologies for managing application containers on OpenStack.

Vision for OpenStack

Self-service

Zun are self-service. It provides users with the ability to deploy containerized applications on demand without having to wait for human action. Zun containers are isolated between tenants. Containers controlled by one tenant are not accessible by other tenants. Quotas are used to limit the number of containers or compute resources (i.e. CPU, RAM) within a tenant.

Application Control

Zun allows application control of containers by offering RESTful API, CLI, and Python API binding. In addition, there are third-party tools like Gophercloud that provide API binding for other programming languages. The access of Zuns API is secured by Keystone so applications that are authenticatable with Keystone can access Zuns API securely.

Interoperability

Zun containers (and other API resources) are designed to be deployable and portable across a variety of public and private OpenStack clouds. Zuns API hides differences between container engines and exposes standard container abstraction.

Bidirectional Compatibility

Zun implements API microversion. API consumers can query the min/max API version that an OpenStack cloud supports, as well as pinning a specific API version to guarantee consistent API behavior across different versions of OpenStack.

Cross-Project Dependencies

Zun depends on Keystone for authentication, Neutron for container networks, Cinder for container volumes. Zun aims to integrate with Placement for tracking compute resources and retrieving allocation candidates. Therefore, Placement is expected to be another dependency of Zun in the near future.

Partitioning

It is totally fine to deploy Zun in multiple OpenStack regions, and each region could have a Zun endpoint in Keystone service catalog. Zun also supports the concept of availability zones - groupings within a region that share no common points of failure.

Basic Physical Data Center Management

Zun interfaces with external systems like Docker engine, which consumes compute resources in data center and offers compute capacity to end-users in the form of containers. Zun APIs provide a consistent interface to various container technologies, which can be implemented by different Open Source projects.

Hardware Virtualisation

Similar to Nova, Zun also aims to virtualize hardware resources (essentially physical servers) and provide them to users via a vendor-independent API. The difference is that Zun delivers compute resources in the form of containers instead of VMs. Operators have a choice of container runtimes which could be a hypervisor-based runtime (i.e. Kata Container) or a traditional runtime (i.e. runc). The choice of container runtime is a trade-off between tenant isolation and performance.

Plays Well With Others

Zun plays well with Container Orchestration Engines like Kubernetes. In particular, there is an OpenStack provider for Virtual Kubelet, which mimics Kubelet to register itself as a node in a Kubernetes cluster. The OpenStack provider leverages Zun to launch container workloads that Kubernetes schedules to the virtual node.

Infinite, Continuous Scaling

Zun facilitates infinite and continuous scaling of applications. It allows users to scale up their applications by spinning up containers on demand (without pre-creating a container host or cluster). Containers allow sharing of physical resources in data center at a more fine-grained level than a VM thus resulting in a better utilization of resources.

Built-in Reliability and Durability

Unlike VMs, containers are usually transient and allowed to be deleted and re-created in response to failure. In this context, Zun aims to provide primitives for deployers to deploy a highly available applications. For example, it allows deployers to deploy their applications across different availability zones. It supports health check of containers so that orchestrators can quickly detect failure and perform recover actions.

Customizable Integration

Zun is integrated with Heat, which allows users to wire containers with resources provided by other services (i.e. networks, volumes, security groups, floating IPs, load balancers, or even VMs). In addition, the Kubernetes integration feature provides another option to wire containers to customize the topology of application deployments.

Graphical User Interface

Zun has a Horizon plugin, which allows users to consume Zun services through a graphical user interface provided by Horizon.

CHAPTER

FIVE

ADDITIONAL MATERIAL

5.1 Zun Command Line Guide

In this section you will find information on Zuns command line interface.

5.1.1 zun-status

CLI interface for Zun status commands

Synopsis

```
zun-status <category> <command> [<args>]
```

Description

zun-status is a tool that provides routines for checking the status of a Zun deployment.

Options

The standard pattern for executing a **zun-status** command is:

```
zun-status <category> <command> [<args>]
```

Run without arguments to see a list of available command categories:

```
zun-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:

```
zun-status upgrade
```

These sections describe the available categories and arguments for **zun-status**.

Upgrade

`zun-status upgrade check` Performs a release-specific readiness check before restarting services with new code. For example, missing or changed configuration options, incompatible object states, or other conditions that could lead to failures while upgrading.

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

3.0.0 (Stein)

- Sample check to be filled in with checks as they are added in Stein.

Note: Other command line guide for Zun to be added. The work will be tracked here: https://blueprints.launchpad.net/zun/+spec/zun-cli-guide

---

5.2 Administrators Guide

5.2.1 Installation & Operations

If you are a system administrator running Zun, this section contains information that should help you understand how to deploy, operate, and upgrade the services.

Use OSProfiler in Zun

This is the demo for Zun integrating with osprofiler. Zun is an OpenStack container management services, while OSProfiler provides a tiny but powerful library that is used by most OpenStack projects and their python clients.

Install Redis database

After osprofiler 1.4.0, user can choose mongodb or redis as the backend storage option without using ceilometer. Here just use Redis as an example, user can choose mongodb, elasticsearch, and etc. Install Redis as the centralized collector. Redis in container is easy to launch, choose Redis Docker and run:

```
$ docker run --name some-redis -p 6379:6379 -d redis
```

Now there is a redis database which has an expose port to access. OSProfiler will send data to this key-value database.

Change the configure file

Change the /etc/zun/zun.conf, add the following lines, change the <ip-address> to the real IP:

```
[profiler]
enabled = True
trace_sqlalchemy = True
hmac_keys = SECRET_KEY
connection_string = redis://<ip-address>:6379/
```

Then restart zun-api and zun-compute (Attention, the newest version of Zun has move zun-api service to apache2 server. You cant restart the service just in screen. Use systemctl restart apache2 will work).

Use below commands to get the trace information:

```
$ zun --profile SECRET_KEY list
```

Use <TRACE-ID>, you will get a <TRACE-ID> for trace:

```
$ osprofiler trace show <TRACE-ID> --connection-string=redis://<ip-address>
↪:6379 --html
```

Troubleshooting

How to check whether the integration is fine: Stop the Redis container, then run the command:

```
$ zun --profile SECRET_KEY list
```

In the zun-api log, will see ConnectionError: Error 111 connecting to <ip-address>:6379. ECONNREFUSED. That means that osprofiler will write the trace data to redis, but cant connect it. So the integration is fine. When /etc/zun/api-paste.ini file changed (change the pipeline), you need to re-deploy the zun service.

Clear Containers in Zun

Zun now supports running Clear Containers with regular Docker containers. Clear containers run containers as very lightweight virtual machines which boot up really fast and has low memory footprints. It provides security to the containers with an isolated environment. You can read more about Clear Containers here.

Installation with DevStack

It is possible to run Clear Containers with Zun. Follow the *Developer Quick-Start* to download DevStack, Zun code and copy the local.conf file. Now perform the following steps to install Clear Containers with DevStack:

```
cd /opt/stack/devstack
echo "ENABLE_CLEAR_CONTAINER=true" >> local.conf
./stack.sh
```

Verify the installation by:

```
$ sudo docker info | grep Runtimes
Runtimes: cor runc
```

Using Clear Containers with Zun

To create Clear Containers with Zun, specify the *runtime* option:

```
zun run --name clear-container --runtime cor cirros ping -c 4 8.8.8.8
```

Note: Clear Containers support in Zun is not production ready. It is recommended not to running Clear Containers and runc containers on the same host.

Keep Containers Alive

As we know, the Docker daemon shuts down all running containers during daemon downtime. Starting with Docker Engine 1.12, users can configure the daemon so that containers remain running when the docker service becomes unavailable. This functionality is called live restore. You can read more about Live Restore here.

Installation with DevStack

It is possible to keep containers alive. Follow the *Developer Quick-Start* to download DevStack, Zun code and copy the local.conf file. Now perform the following steps to install Zun with DevStack:

```
cd /opt/stack/devstack
echo "ENABLE_LIVE_RESTORE=true" >> local.conf
./stack.sh
```

Verify the installation by:

```
$ sudo docker info | grep "Live Restore"
Live Restore Enabled: true
```

Manage container security

Security groups are sets of IP filter rules that define networking access to the container. Group rules are project specific; project members can edit the default rules for their group and add new rule sets.

All projects have a `default` security group which is applied to any container that has no other defined security group. Unless you change the default, this security group denies all incoming traffic and allows only outgoing traffic to your container.

Create a container with security group

When adding a new security group, you should pick a descriptive but brief name. This name shows up in brief descriptions of the containers that use it where the longer description field often does not. For example, seeing that a container is using security group http is much easier to understand than bobs_group or secgrp1.

1. Add the new security group, as follows:

```
$ openstack security group create SEC_GROUP_NAME --description Description
```

For example:

```
$ openstack security group create global_http --description "Allows Web␣
↪traffic anywhere on the Internet."
+-----------------+-------------------------------------------------------
↪--------------------------------------------------------------+
| Field           | Value                                                 ␣
↪                                                                |
+-----------------+-------------------------------------------------------
↪--------------------------------------------------------------+
| created_at      | 2016-11-03T13:50:53Z                                  ␣
↪                                                                |
| description     | Allows Web traffic anywhere on the Internet.          ␣
↪                                                                |
| headers         |                                                       ␣
↪                                                                |
| id              | c0b92b20-4575-432a-b4a9-eaf2ad53f696                  ␣
↪                                                                |
| name            | global_http                                           ␣
↪                                                                |
| project_id      | 5669caad86a04256994cdf755df4d3c1                      ␣
↪                                                                |
| project_id      | 5669caad86a04256994cdf755df4d3c1                      ␣
↪                                                                |
| revision_number | 1                                                     ␣
↪                                                                |
| rules           | created_at='2016-11-03T13:50:53Z', direction='egress',
↪ ethertype='IPv4', id='4d8cec94-e0ee-4c20-9f56-8fb67c21e4df',      |
|                 | project_id='5669caad86a04256994cdf755df4d3c1',␣
↪revision_number='1', updated_at='2016-11-03T13:50:53Z'               ␣
↪   |
|                 | created_at='2016-11-03T13:50:53Z', direction='egress',
↪ ethertype='IPv6', id='31be2ad1-be14-4aef-9492-ecebede2cf12',      |
|                 | project_id='5669caad86a04256994cdf755df4d3c1',␣
↪revision_number='1', updated_at='2016-11-03T13:50:53Z'               ␣
↪   |
| updated_at      | 2016-11-03T13:50:53Z                                  ␣
↪                                                                |
+-----------------+-------------------------------------------------------
↪--------------------------------------------------------------+
```

2. Add a new group rule, as follows:

```
$ openstack security group rule create SEC_GROUP_NAME \
    --protocol PROTOCOL --dst-port FROM_PORT:TO_PORT --remote-ip CIDR
```

The arguments are positional, and the `from-port` and `to-port` arguments specify the local port range connections are allowed to access, not the source and destination ports of the connection. For example:

```
$ openstack security group rule create global_http \
    --protocol tcp --dst-port 80:80 --remote-ip 0.0.0.0/0
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| created_at        | 2016-11-06T14:02:00Z                 |
| description       |                                      |
| direction         | ingress                              |
| ethertype         | IPv4                                 |
| headers           |                                      |
| id                | 2ba06233-d5c8-43eb-93a9-8eaa94bc9eb5 |
| port_range_max    | 80                                   |
| port_range_min    | 80                                   |
| project_id        | 5669caad86a04256994cdf755df4d3c1     |
| project_id        | 5669caad86a04256994cdf755df4d3c1     |
| protocol          | tcp                                  |
| remote_group_id   | None                                 |
| remote_ip_prefix  | 0.0.0.0/0                            |
| revision_number   | 1                                    |
| security_group_id | c0b92b20-4575-432a-b4a9-eaf2ad53f696 |
| updated_at        | 2016-11-06T14:02:00Z                 |
+-------------------+--------------------------------------+
```

3. Create a container with the new security group, as follows:

```
$ openstack appcontainer run --security-group SEC_GROUP_NAME IMAGE
```

For example:

```
$ openstack appcontainer run --security-group global_http nginx
```

Find containers security groups

If you cannot access your application inside the container, you might want to check the security groups of the container to ensure the rules dont block the traffic.

1. List the containers, as follows:

```
$ openstack appcontainer list
+--------------------------------------+--------------------+-------+-----
↪----+------------+-----------+-------+
| uuid                                 | name               | image |␣
↪status  | task_state | addresses | ports |
```

(continues on next page)

(continued from previous page)

```
+--------------------------------------+--------------------+-------+-----
↪----+------------+----------+-------+
| 6595aff8-6c1c-4e64-8aad-bfd3793efa54 | delta-24-container | nginx |␣
↪Running | None       | 10.5.0.14 | [80]  |
+--------------------------------------+--------------------+-------+-----
↪----+------------+----------+-------+
```

2. Find all your containers ports, as follows:

```
$ openstack port list --fixed-ip ip-address=10.5.0.14
+--------------------------------------+-----------------------------------
↪--------------------------------------+-------------------+--------------
↪-----------------------------------------------------------+--------+
| ID                                   | Name                              ␣
↪                                     | MAC Address       | Fixed IP␣
↪Addresses                                                    |␣
↪Status |
+--------------------------------------+-----------------------------------
↪--------------------------------------+-------------------+--------------
↪-----------------------------------------------------------+--------+
| b02df384-fd58-43ee-a44a-f17be9dd4838 |␣
↪405061f9eeda5dbfa10701a72051c91a5555d19f6ef7b3081078d102fe6f60ab-port |␣
↪fa:16:3e:52:3c:0c | ip_address='10.5.0.14', subnet_id='7337ad8b-7314-
↪4a33-ba54-7362f0a7a680' | ACTIVE |
+--------------------------------------+-----------------------------------
↪--------------------------------------+-------------------+--------------
↪-----------------------------------------------------------+--------+
```

3. View the details of each port to retrieve the list of security groups, as follows:

```
$ openstack port show b02df384-fd58-43ee-a44a-f17be9dd4838
+-----------------------+-----------------------------------------------------
↪-------------------------+
| Field                 | Value                                               ␣
↪                       |
+-----------------------+-----------------------------------------------------
↪-------------------------+
| admin_state_up        | UP                                                  ␣
↪                       |
| allowed_address_pairs |                                                     ␣
↪                       |
| binding_host_id       | None                                                ␣
↪                       |
| binding_profile       | None                                                ␣
↪                       |
| binding_vif_details   | None                                                ␣
↪                       |
| binding_vif_type      | None                                                ␣
↪                       |
| binding_vnic_type     | normal                                              ␣
↪                       |
```

(continues on next page)

(continued from previous page)

```
| created_at            | 2018-05-11T21:58:42Z                                            ␣
↪                          |
| data_plane_status     | None                                                            ␣
↪                          |
| description           |                                                                 ␣
↪                          |
| device_id             | 6595aff8-6c1c-4e64-8aad-bfd3793efa54                            ␣
↪                          |
| device_owner          | compute:kuryr                                                   ␣
↪                          |
| dns_assignment        | None                                                            ␣
↪                          |
| dns_name              | None                                                            ␣
↪                          |
| extra_dhcp_opts       |                                                                 ␣
↪                          |
| fixed_ips             | ip_address='10.5.0.14', subnet_id='7337ad8b-
↪7314-4a33-ba54-7362f0a7a680' |
| id                    | b02df384-fd58-43ee-a44a-f17be9dd4838                            ␣
↪                          |
| ip_address            | None                                                            ␣
↪                          |
| mac_address           | fa:16:3e:52:3c:0c                                               ␣
↪                          |
| name                  |␣
↪405061f9eeda5dbfa10701a72051c91a5555d19f6ef7b3081078d102fe6f60ab-port ␣
↪ |
| network_id            | 695aff90-66c6-4383-b37c-7484c4046a64                            ␣
↪                          |
| option_name           | None                                                            ␣
↪                          |
| option_value          | None                                                            ␣
↪                          |
| port_security_enabled | True                                                            ␣
↪                          |
| project_id            | c907162152fe41f288912e991762b6d9                                ␣
↪                          |
| qos_policy_id         | None                                                            ␣
↪                          |
| revision_number       | 9                                                               ␣
↪                          |
| security_group_ids    | ba20b63e-8a61-40e4-a1a3-5798412cc36b                            ␣
↪                          |
| status                | ACTIVE                                                          ␣
↪                          |
| subnet_id             | None                                                            ␣
↪                          |
| tags                  | kuryr.port.existing                                             ␣
↪                          |
```

(continues on next page)

(continued from previous page)

```
| trunk_details          | None                                                 ␣
↪                          |
| updated_at             | 2018-05-11T21:58:47Z                                 ␣
↪                          |
+------------------------+------------------------------------------------------
↪-------------------------+
```

4. View the rules of security group showed up at `security_group_ids` field of the port, as follows:

```
$ openstack security group rule list ba20b63e-8a61-40e4-a1a3-5798412cc36b
+--------------------------------------+-------------+-----------+--------
↪----+-------------------------+
| ID                                   | IP Protocol | IP Range  | Port␣
↪Range | Remote Security Group |
+--------------------------------------+-------------+-----------+--------
↪----+-------------------------+
| 24ebfdb8-591c-40bb-a7d3-f5b5eadc72ca | None        | None      |       ␣
↪    | None                    |
| 907bf692-3dbb-4b34-ba7a-22217e6dbc4f | None        | None      |       ␣
↪    | None                    |
| bbcd3b46-0214-4966-8050-8b5d2f9121d1 | tcp         | 0.0.0.0/0 | 80:80 ␣
↪    | None                    |
+--------------------------------------+-------------+-----------+--------
↪----+-------------------------+
```

How to use private docker registry with Zun

Zun by default pull container images from Docker Hub. However, it is possible to configure Zun to pull images from a private registry.

This document provides an example to deploy and configure a docker registry for Zun. For a comprehensive guide about deploying a docker registry, see here

Deploy Private Docker Registry

A straightforward approach to install a private docker registry is to deploy it as a Zun container:

```
$ openstack appcontainer create \
    --restart always \
    --expose-port 443 \
    --name registry \
    --environment REGISTRY_HTTP_ADDR=0.0.0.0:443 \
    --environment REGISTRY_HTTP_TLS_CERTIFICATE=/domain.crt \
    --environment REGISTRY_HTTP_TLS_KEY=/domain.key \
    registry:2
```

Note: Depending on the configuration of your tenant network, you might need to make sure the container is accessible from other tenants of your cloud. For example, you might need to associate a floating IP to

the container.

In order to make your registry accessible to external hosts, you must use a TLS certificate (issued by a certificate issuer) or create self-signed certificates. This document shows you how to generate and use self-signed certificates:

```
$ mkdir -p certs
$ cat > certs/domain.conf <<EOF
[req]
distinguished_name = req_distinguished_name
req_extensions     = req_ext
prompt = no
[req_distinguished_name]
CN = zunregistry.com
[req_ext]
subjectAltName = IP:172.24.4.49
EOF
$ openssl req \
    -newkey rsa:4096 -nodes -sha256 -keyout certs/domain.key \
    -x509 -days 365 -out certs/domain.crt -config certs/domain.conf
```

Note: Replace `zunregistry.com` with the domain name of your registry.

Note: Replace `172.24.4.49` with the IP address of your registry.

Note: You need to make sure the domain name (i.e. `zunregistry.com`) will be resolved to the IP address (i.e. `172.24.4.49`). For example, you might need to edit `/etc/hosts` accordingly.

Copy the certificates to registry:

```
$ openstack appcontainer cp certs/domain.key registry:/
$ openstack appcontainer cp certs/domain.crt registry:/
```

Configure docker daemon to accept the certificates:

```
# mkdir -p /etc/docker/certs.d/zunregistry.com
# cp certs/domain.crt /etc/docker/certs.d/zunregistry.com/ca.crt
```

Note: Replace `zunregistry.com` with the domain name of your registry.

Note: Perform this steps in every compute nodes.

Start the registry:

```
$ openstack appcontainer start registry
```

Verify the registry is working:

```
$ docker pull ubuntu:16.04
$ docker tag ubuntu:16.04 zunregistry.com/my-ubuntu
$ docker push zunregistry.com/my-ubuntu
$ openstack appcontainer run --interactive zunregistry.com/my-ubuntu /bin/bash
```

Note: Replace `zunregistry.com` with the domain name of your registry.

5.3 Sample Configuration File

5.3.1 Zun Configuration Options

The following is a sample Zun configuration for adaptation and use. It is auto-generated from Zun when this documentation is built, so if you are having issues with an option, please compare your version of Zun with the version of this documentation.

5.3.2 Policy configuration

Configuration

Warning: JSON formatted policy file is deprecated since Zun 7.0.0 (Wallaby). This oslopolicy-convert-json-to-yaml tool will migrate your existing JSON-formatted policy file to YAML in a backward-compatible way.

The following is an overview of all available policies in Zun. For a sample configuration file.

zun

context_is_admin

Default `role:admin`

(no description provided)

admin_or_owner

Default `is_admin:True or project_id:%(project_id)s`

(no description provided)

admin_api

Default `rule:context_is_admin`

(no description provided)

deny_everybody

Default !

Default rule for deny everybody.

container:create

Default is_admin:True or project_id:%(project_id)s

Operations

- **POST** /v1/containers

Create a new container.

container:create:runtime

Default rule:context_is_admin

Operations

- **POST** /v1/containers

Create a new container with specified runtime.

container:create:privileged

Default rule:deny_everybody

Operations

- **POST** /v1/containers

Create a new privileged container.Warning: the privileged container has a big security risk so be caution if you want to enable this feature

container:create:requested_destination

Default rule:context_is_admin

Operations

- **POST** /v1/containers

Create a container on the requested compute host.

container:create:image_pull_policy

Default rule:context_is_admin

Operations

- **POST** /v1/containers

Create a new container with specified image pull policy.

container:delete

Default is_admin:True or project_id:%(project_id)s

Operations

- **DELETE** /v1/containers/{container_ident}

Delete a container.

container:delete_all_projects

Default `rule:context_is_admin`

Operations

- **DELETE** `/v1/containers/{container_ident}`

Delete a container from all projects.

`container:delete_force`

Default `rule:context_is_admin`

Operations

- **DELETE** `/v1/containers/{container_ident}`

Forcibly delete a container.

`container:get_one`

Default `is_admin:True or project_id:%(project_id)s`

Operations

- **GET** `/v1/containers/{container_ident}`

Retrieve the details of a specific container.

`container:get_one:host`

Default `rule:context_is_admin`

Operations

- **GET** `/v1/containers/{container_ident}`
- **GET** `/v1/containers`
- **POST** `/v1/containers`
- **PATCH** `/v1/containers/{container_ident}`

Retrieve the host field of containers.

`container:get_one:image_pull_policy`

Default `rule:context_is_admin`

Operations

- **GET** `/v1/containers/{container_ident}`
- **GET** `/v1/containers`
- **POST** `/v1/containers`
- **PATCH** `/v1/containers/{container_ident}`

Retrieve the image_pull_policy field of containers.

`container:get_one:privileged`

Default `rule:context_is_admin`

Operations

- **GET** `/v1/containers/{container_ident}`

- **GET** /v1/containers
- **POST** /v1/containers
- **PATCH** /v1/containers/{container_ident}

Retrieve the privileged field of containers.

`container:get_one:runtime`

Default rule:context_is_admin

Operations

- **GET** /v1/containers/{container_ident}
- **GET** /v1/containers
- **POST** /v1/containers
- **PATCH** /v1/containers/{container_ident}

Retrieve the runtime field of containers.

`container:get_one_all_projects`

Default rule:context_is_admin

Operations

- **GET** /v1/containers/{container_ident}

Retrieve the details of a specific container from all projects.

`container:get_all`

Default is_admin:True or project_id:%(project_id)s

Operations

- **GET** /v1/containers

Retrieve the details of all containers.

`container:get_all_all_projects`

Default rule:context_is_admin

Operations

- **GET** /v1/containers

Retrieve the details of all containers across projects.

`container:update`

Default is_admin:True or project_id:%(project_id)s

Operations

- **PATCH** /v1/containers/{container_ident}

Update a container.

`container:start`

Default is_admin:True or project_id:%(project_id)s

Operations

- **POST** /v1/containers/{container_ident}/start

Start a container.

`container:stop`

Default is_admin:True or project_id:%(project_id)s

Operations

- **POST** /v1/containers/{container_ident}/stop

Stop a container.

`container:reboot`

Default is_admin:True or project_id:%(project_id)s

Operations

- **POST** /v1/containers/{container_ident}/reboot

Reboot a container.

`container:pause`

Default is_admin:True or project_id:%(project_id)s

Operations

- **POST** /v1/containers/{container_ident}/pause

Pause a container.

`container:unpause`

Default is_admin:True or project_id:%(project_id)s

Operations

- **POST** /v1/containers/{container_ident}/unpause

Unpause a container.

`container:logs`

Default is_admin:True or project_id:%(project_id)s

Operations

- **GET** /v1/containers/{container_ident}/logs

Get the log of a container

`container:execute`

Default is_admin:True or project_id:%(project_id)s

Operations

- **POST** /v1/containers/{container_ident}/execute

Execute command in a running container

`container:execute_resize`

Default `is_admin:True or project_id:%(project_id)s`

Operations

- **POST** `/v1/containers/{container_ident}/execute_resize`

Resize the TTY used by an execute command.

`container:kill`

Default `is_admin:True or project_id:%(project_id)s`

Operations

- **POST** `/v1/containers/{container_ident}/kill`

Kill a running container

`container:rename`

Default `is_admin:True or project_id:%(project_id)s`

Operations

- **POST** `/v1/containers/{container_ident}/rename`

Rename a container.

`container:attach`

Default `is_admin:True or project_id:%(project_id)s`

Operations

- **GET** `/v1/containers/{container_ident}/attach`

Attach to a running container

`container:resize`

Default `is_admin:True or project_id:%(project_id)s`

Operations

- **POST** `/v1/containers/{container_ident}/resize`

Resize a container.

`container:top`

Default `is_admin:True or project_id:%(project_id)s`

Operations

- **GET** `/v1/containers/{container_ident}/top`

Display the running processes inside the container.

`container:get_archive`

Default `is_admin:True or project_id:%(project_id)s`

Operations

- **GET** `/v1/containers/{container_ident}/get_archive`

Get a tar archive of a path of container.

container:put_archive

Default is_admin:True or project_id:%(project_id)s

Operations

- **PUT** /v1/containers/{container_ident}/put_archive

Put a tar archive to be extracted to a path of container

container:stats

Default is_admin:True or project_id:%(project_id)s

Operations

- **GET** /v1/containers/{container_ident}/stats

Display the statistics of a container

container:commit

Default is_admin:True or project_id:%(project_id)s

Operations

- **POST** /v1/containers/{container_ident}/commit

Commit a container

container:add_security_group

Default is_admin:True or project_id:%(project_id)s

Operations

- **POST** /v1/containers/{container_ident}/add_security_group

Add a security group to a specific container.

container:network_detach

Default is_admin:True or project_id:%(project_id)s

Operations

- **POST** /v1/containers/{container_ident}/network_detach

Detach a network from a container.

container:network_attach

Default is_admin:True or project_id:%(project_id)s

Operations

- **POST** /v1/containers/{container_ident}/network_attach

Attach a network from a container.

container:remove_security_group

Default is_admin:True or project_id:%(project_id)s

Operations

- **POST** /v1/containers/{container_ident}/remove_security_group

Remove security group from a specific container.

`container:rebuild`

Default `is_admin:True or project_id:%(project_id)s`

Operations

- **POST** `/v1/containers/{container_ident}/rebuild`

Rebuild a container.

`container:resize_container`

Default `is_admin:True or project_id:%(project_id)s`

Operations

- **POST** `/v1/containers/{container_ident}/resize_container`

Resize an existing container.

`image:pull`

Default `rule:context_is_admin`

Operations

- **POST** `/v1/images`

Pull an image.

`image:get_all`

Default `rule:context_is_admin`

Operations

- **GET** `/v1/images`

Print a list of available images.

`image:get_one`

Default `rule:context_is_admin`

Operations

- **GET** `/v1/images/{image_id}`

Retrieve the details of a specific image.

`image:search`

Default `is_admin:True or project_id:%(project_id)s`

Operations

- **GET** `/v1/images/{image_ident}/search`

Search an image.

`image:delete`

Default `rule:context_is_admin`

Operations

- **DELETE** /v1/images/{image_ident}

Delete an image.

zun-service:delete

Default rule:context_is_admin

Operations

- **DELETE** /v1/services

Delete a service.

zun-service:disable

Default rule:context_is_admin

Operations

- **PUT** /v1/services/disable

Disable a service.

zun-service:enable

Default rule:context_is_admin

Operations

- **PUT** /v1/services/enable

Enable a service.

zun-service:force_down

Default rule:context_is_admin

Operations

- **PUT** /v1/services/force_down

Forcibly shutdown a service.

zun-service:get_all

Default rule:context_is_admin

Operations

- **GET** /v1/services

Show the status of a service.

host:get_all

Default rule:context_is_admin

Operations

- **GET** /v1/hosts

List all compute hosts.

host:get

Default rule:context_is_admin

Operations

- **GET** /v1/hosts/{host_ident}

Show the details of a specific compute host.

`capsule:create`

Default is_admin:True or project_id:%(project_id)s

Operations

- **POST** /v1/capsules/

Create a capsule

`capsule:delete`

Default is_admin:True or project_id:%(project_id)s

Operations

- **DELETE** /v1/capsules/{capsule_ident}

Delete a capsule

`capsule:delete_all_projects`

Default rule:context_is_admin

Operations

- **DELETE** /v1/capsules/{capsule_ident}

Delete a container in any project.

`capsule:get`

Default is_admin:True or project_id:%(project_id)s

Operations

- **GET** /v1/capsules/{capsule_ident}

Retrieve the details of a capsule.

`capsule:get:host`

Default rule:context_is_admin

Operations

- **GET** /v1/capsules/{capsule_ident}
- **GET** /v1/capsules
- **POST** /v1/capsules

Retrieve the host field of a capsule.

`capsule:get_one_all_projects`

Default rule:context_is_admin

Operations

- **GET** /v1/capsules/{capsule_ident}

Retrieve the details of a capsule in any project.

`capsule:get_all`

Default `is_admin:True or project_id:%(project_id)s`

Operations

- **GET** `/v1/capsules/`

List all capsules.

`capsule:get_all_all_projects`

Default `rule:context_is_admin`

Operations

- **GET** `/v1/capsules/`

List all capsules across projects.

`network:attach_external_network`

Default `role:admin`

Operations

- **POST** `/v1/containers`

Attach an unshared external network to a container

`network:create`

Default `role:admin`

Operations

- **POST** `/v1/networks`

Create a network

`network:delete`

Default `role:admin`

Operations

- **DELETE** `/v1/networks`

Delete a network

`container:actions`

Default `is_admin:True or project_id:%(project_id)s`

Operations

- **GET** `/v1/containers/{container_ident}/container_actions/`
- **GET** `/v1/containers/{container_ident}/container_actions/{request_id}`

List actions and show action details for a container

`container:action:events`

Default `rule:context_is_admin`

Operations

- **GET** /v1/containers/{container_ident}/container_actions/{request_id}

Add events details in action details for a container.

`availability_zones:get_all`

Default is_admin:True or project_id:%(project_id)s

Operations

- **GET** /v1/availability_zones

List availability zone

`quota:update`

Default rule:context_is_admin

Operations

- **PUT** /v1/quotas/{project_id}

Update quotas for a project

`quota:delete`

Default rule:context_is_admin

Operations

- **DELETE** /v1/quotas/{project_id}

Delete quotas for a project

`quota:get`

Default is_admin:True or project_id:%(project_id)s

Operations

- **GET** /v1/quotas/{project_id}

Get quotas for a project

`quota:get_default`

Default is_admin:True or project_id:%(project_id)s

Operations

- **GET** /v1/quotas/defaults

Get default quotas for a project

`quota_class:update`

Default rule:context_is_admin

Operations

- **PUT** /v1/quota_classes/{quota_class_name}

Update quotas for specific quota class

`quota_class:get`

Default `rule:context_is_admin`

Operations

- **GET** `/v1/quota_classes/{quota_class_name}`

List quotas for specific quota class

`registry:create`

Default `is_admin:True or project_id:%(project_id)s`

Operations

- **POST** `/v1/registries`

Create a new registry.

`registry:delete`

Default `is_admin:True or project_id:%(project_id)s`

Operations

- **DELETE** `/v1/registries/{registry_ident}`

Delete a registry.

`registry:get_one`

Default `is_admin:True or project_id:%(project_id)s`

Operations

- **GET** `/v1/registries/{registry_ident}`

Retrieve the details of a specific registry.

`registry:get_all`

Default `is_admin:True or project_id:%(project_id)s`

Operations

- **GET** `/v1/registries`

Retrieve the details of all registries.

`registry:get_all_all_projects`

Default `rule:context_is_admin`

Operations

- **GET** `/v1/registries`

Retrieve the details of all registries across projects.

`registry:update`

Default `is_admin:True or project_id:%(project_id)s`

Operations

- **PATCH** `/v1/registries/{registry_ident}`

Update a registry.

5.4 Filter Scheduler

The **Filter Scheduler** supports *filtering* zun compute hosts to make decisions on where a new container should be created.

5.4.1 Filtering

Filter Scheduler iterates over all found compute hosts,evaluating each host against a set of filters. The Scheduler then chooses a host for the requested container. A specific filter can decide whether to pass or filter out a specific host. The decision is made based on the user request specification, the state of the host, and/or some extra information.

If the Scheduler cannot find candidates for the container, it means that there are no appropriate host where that container can be scheduled.

The Filter Scheduler has a set of `filters` that are built-in. If the built-in filters are insufficient, you can implement your own filters with your filtering algorithm.

There are many standard filter classes which may be used (`zun.scheduler.filters`):

- CPUFilter - filters based on CPU core utilization. It passes hosts with sufficient number of CPU cores.
- RamFilter - filters hosts by their RAM. Only hosts with sufficient RAM to host the instance are passed.
- LabelFilter - filters hosts based on whether host has the CLI specified labels.
- ComputeFilter - filters hosts that are operational and enabled. In general, you should always enable this filter.
- RuntimeFilter - filters hosts by their runtime. It passes hosts with the specified runtime.

5.4.2 Configuring Filters

To use filters you specify two settings:

- `filter_scheduler.available_filters` - Defines filter classes made available to the scheduler.
- `filter_scheduler.enabled_filters` - Of the available filters, defines those that the scheduler uses by default.

The default values for these settings in zun.conf are:

```
--filter_scheduler.available_filters=zun.scheduler.filters.all_filters
--filter_scheduler.enabled_filters=RamFilter,CPUFilter,ComputeFilter,
↪RuntimeFilter
```

With this configuration, all filters in `zun.scheduler.filters` would be available, and by default the RamFilter and CPUFilter would be used.

5.4.3 Writing Your Own Filter

To create **your own filter** you must inherit from BaseHostFilter and implement one method: `host_passes`. This method should return `True` if the host passes the filter.

P.S.: you can find more examples of using Filter Scheduler and standard filters in `zun.tests.scheduler`.

5.5 Reference Material

5.5.1 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.

1.1

This is the initial version of the v1.1 API which supports microversions. The v1.1 API is from the REST API userss point of view exactly the same as v1.0 except with strong input validation.

A user can specify a header in the API request:

```
OpenStack-API-Version: <version>
```

where `<version>` is any valid api version for this API.

If no version is specified then the API will behave as if a version request of v1.1 was requested.

1.2

Add a new attribute nets to the request to create a container. Users can use this attribute to specify one or multiple networks for the container. Each network could specify the neutron network, neutron port, or a v4/v6 IP address. For examples:

```
[{'port': '1234567'}]
[{'v4-fixed-ip': '127.0.0.1'}]
[{'network': 'test'}]
[{'network': 'test2'}]
[{'v6-fixed-ip': '2f:33:45'}]
```

1.3

Add auto_remove field for creating a container. With this field, the container will be automatically removed if it exists. The new one will be created instead.

1.4

Add host list api. Users can use this api to list all the zun compute hosts. Add get host api Users can use this api to get details of a zun compute host.

1.5

Add a new attribute runtime to the request to create a container. Users can use this attribute to choose runtime for their containers. The specified runtime should be configured by admin to run with Zun. The default runtime for Zun is runc.

1.6

Add detach a network from a container api. Users can use this api to detach a neutron network from a container.

1.7

Disallow non-admin users to force delete containers Only Admin User can use delete force to force delete a container.

1.8

Add attach a network to a container. Users can use this api to attach a neutron network to a container.

1.9

Add a new attribute hostname to the request to create a container. Users can use this attribute to specify containers hostname.

1.10

Make container delete API async. Delete operation for a container can take long time, so making it async to improve user experience.

1.11

Add a new attribute mounts to the request to create a container. Users can use this attribute to specify one or multiple mounts for the container. Each mount could specify the source and destination. The source is the Cinder volume id or name, and the destination is the path where the file or directory will be mounted in the container. For examples:

[{source: my-vol, destination: /data}]

1.12

Add a new attribute stop to the request to delete containers. Users can use this attribute to stop and delete the container without using the force option.

1.13

Add a new api for a list of networks on a container. Users can use this api to list up neutron network on a container.

1.14

Remove the container rename endpoint (POST /containers/<container>/rename). The equivalent functionality is re-introduced by the patch endpoint (PATCH /containers/<container>). To rename a container, users can send a request to the endpoint with the data in the following form:

{name: <new-name>}

1.15

Remove the APIs for adding/removing security group to/from a container. These APIs are removed because they are proxy APIs to Neutron.

1.16

Modify restart_policy to capsule spec content to align with Kubernetes.

1.17

Add parameter `port` to the network_detach API. This allow users to detach a container from a neutron port.

1.18

Modify the response of network_list (GET /v1/containers/{container_ident}/network_list) API. The normal response will be something like:

```
{
    "networks": [
        {
            "port_id": "5be06e49-70dc-4984-94a2-1b946bb136fb",
            "net_id": "7e6b5e1b-9b44-4f55-b4e3-16a1ead98161",
            "fixed_ips" [
                "ip_address": "30.30.30.10",
                "version": 4,
                "subnet_id": "ae8d7cce-859e-432f-8a33-d7d8834ccd14"
            ]
        }
    ]
}
```

1.19

Introduce an API endpoint for resizing a container, such as changing the CPU or memory of the container.

1.20

Convert type of command from string to list

1.21

Support privileged container

1.22

Add healthcheck to container create

1.23

Add support for file injection when creating a container. The content of the file is sent to Zun server via parameter mounts.

1.24

Add a parameter exposed_ports to the request of creating a container. This parameter is of the following form:

exposed_ports: { <port>/<protocol>: {} }

where port is the containers port and protocol is either tcp or udp. If this parameter is specified, Zun will create a security group and open the exposed port. This parameter cannot be used together with the security_groups parameter because Zun will manage the security groups of the container.

1.25

The get_archive endpoint returns a encoded archived file data by using Base64 algorithm. The put_archive endpoint take a Base64-encoded archived file data as input.

1.26

Introduce Quota support API

1.27

Introduce API for deleting network. By default, this is an admin API.

1.28

Add a new attribute cpu_policy. Users can use this attribute to determine which CPU policy the container uses.

1.29

Add a new attribute enable_cpu_pinning to host resource.

1.30

Introduce API endpoint for create/read/update/delete private registry.

1.31

Add registry_id to container resource. This attribute indicate the registry from which the container pulls images.

1.32

Make capsule deletion asynchronized. API request to delete a capsule will return without waiting for the capsule to be deleted.

1.33

Add finish_time to container action resource. If the action is finished, finish_time shows the finish time. Otherwise, this field will be None.

1.34

Add init_containers to capsule. This field contains a list of init_container information.

1.35

Support processing ports field in capsules container. Users can leverage this field to open ports of a container. For example:

```
spec:
  containers:
  - image: "nginx"
    ports:
    - containerPort: 80
      protocol: TCP
```

1.36

Add tty to container. This field indicate if the container should allocate a TTY for itself.

1.37

Add tty and stdin to capsule. Containers in capsule can specify these two fields.

1.38

Add annotations to capsule. This field stores metadata of the capsule in key-value format.

1.39

Add host parameter on POST /v1/containers. This field is used to request a host to run the container.

1.40

Add entrypoint parameter on POST /v1/containers. This field is used to overwrite the default ENTRYPOINT of the image.