OpenStack-Ansible Documentation

Release 32.1.0.dev51

OpenStack-Ansible Contributors

Feb 06, 2026

CONTENTS

CHAPTER

ONE

INTRODUCTION

OpenStack-Ansible provides Ansible playbooks and roles for deploying and configuring an OpenStack environment.

Documentation for all previous releases is available by substituting `latest` with an expected release (i.e., 2025.2) directly in the address bar (URL) of your browser.

Support status for each OpenStack release can be found on releases.openstack.org.

1.1 2026.1 (Gazpacho): Under Development

1.1.1 Operations Guide

This guide provides information about operating your OpenStack-Ansible deployment.

For information on how to deploy your OpenStack-Ansible cloud, refer to the Deployment Guide for step-by-step instructions on how to deploy the OpenStack packages and dependencies on your cloud using OpenStack-Ansible.

For user guides, see the *User Guide*.

For information on how to contribute, extend or develop OpenStack-Ansible, see the Developer Documentation.

For in-depth technical information, see the *OpenStack-Ansible Reference*.

This guide ranges from first operations to verify your deployment, to the major upgrades procedures.

Verify OpenStack-Ansible cloud

This chapter is intended to document basic OpenStack operations to verify your OpenStack-Ansible deployment.

It explains how CLIs can be used as an admin and a user, to ensure the well-behavior of your cloud.

Check your OpenStack-Ansible cloud

This chapter goes through the verification steps for a basic operation of the OpenStack API and dashboard, as an administrator.

> **Note**
>
> The utility container provides a CLI environment for additional configuration and testing.

1. Access the utility container:

```
$ lxc-attach -n `lxc-ls -1 | grep utility | head -n 1`
```

2. Source the `admin` project credentials:

```
$ . ~/openrc
```

3. Run an OpenStack command that uses one or more APIs. For example:

```
$ openstack user list --domain default
+----------------------------------+--------------------+
| ID                               | Name               |
+----------------------------------+--------------------+
| 04007b990d9442b59009b98a828aa981 | glance             |
| 0ccf5f2020ca4820847e109edd46e324 | keystone           |
| 1dc5f638d4d840c690c23d5ea83c3429 | neutron            |
| 3073d0fa5ced46f098215d3edb235d00 | cinder             |
| 5f3839ee1f044eba921a7e8a23bb212d | admin              |
| 61bc8ee7cc9b4530bb18acb740ee752a | stack_domain_admin |
| 77b604b67b79447eac95969aafc81339 | alt_demo           |
| 85c5bf07393744dbb034fab788d7973f | nova               |
| a86fc12ade404a838e3b08e1c9db376f | swift              |
| bbac48963eff4ac79314c42fc3d7f1df | ceilometer         |
| c3c9858cbaac4db9914e3695b1825e41 | dispersion         |
| cd85ca889c9e480d8ac458f188f16034 | demo               |
| efab6dc30c96480b971b3bd5768107ab | heat               |
+----------------------------------+--------------------+
```

4. With a web browser, access the Dashboard using the external load balancer domain name or IP address. This is defined by the `external_lb_vip_address` option in the `/etc/openstack_deploy/openstack_user_config.yml` file. The dashboard uses HTTPS on port 443.

5. Authenticate using the username `admin` and password defined by the `keystone_auth_admin_password` option in the `/etc/openstack_deploy/user_secrets.yml` file.

6. Run an OpenStack command to reveal all endpoints from your deployment. For example:

```
$ openstack endpoint list
+----------------+-----------+--------------+----------------+---------+--
↪---------+----------------------------------------------------+
| ID             | Region    | Service Name | Service Type   | Enabled |␣
↪Interface | URL                                                |
+----------------+-----------+--------------+----------------+---------+--
↪---------+----------------------------------------------------+
| [ID truncated] | RegionOne | cinderv2     | volumev2       | True    |␣
↪admin     | http://172.29.236.100:8776/v2/%(project_id)s       |
| [ID truncated] | RegionOne | cinderv3     | volumev3       | True    |␣
↪public    | https://10.23.100.127:8776/v3/%(project_id)s       |
| [ID truncated] | RegionOne | aodh         | alarming       | True    |␣
↪internal  | http://172.29.236.100:8042                         |
```

(continues on next page)

(continued from previous page)

```
| [ID truncated] | RegionOne | glance       | image          | True    |␣
↪public    | https://10.23.100.127:9292                          |
| [ID truncated] | RegionOne | cinderv2     | volumev2       | True    |␣
↪internal  | http://172.29.236.100:8776/v2/%(project_id)s        |
| [ID truncated] | RegionOne | heat-cfn     | cloudformation | True    |␣
↪admin     | http://172.29.236.100:8000/v1                       |
| [ID truncated] | RegionOne | neutron      | network        | True    |␣
↪admin     | http://172.29.236.100:9696                          |
| [ID truncated] | RegionOne | aodh         | alarming       | True    |␣
↪public    | https://10.23.100.127:8042                          |
| [ID truncated] | RegionOne | nova         | compute        | True    |␣
↪admin     | http://172.29.236.100:8774/v2.1/%(project_id)s      |
| [ID truncated] | RegionOne | heat-cfn     | cloudformation | True    |␣
↪internal  | http://172.29.236.100:8000/v1                       |
| [ID truncated] | RegionOne | swift        | object-store   | True    |␣
↪public    | https://10.23.100.127:8080/v1/AUTH_%(project_id)s |
| [ID truncated] | RegionOne | designate    | dns            | True    |␣
↪admin     | http://172.29.236.100:9001                          |
| [ID truncated] | RegionOne | cinderv2     | volumev2       | True    |␣
↪public    | https://10.23.100.127:8776/v2/%(project_id)s        |
| [ID truncated] | RegionOne | keystone     | identity       | True    |␣
↪admin     | http://172.29.236.100:5000/v3                       |
| [ID truncated] | RegionOne | nova         | compute        | True    |␣
↪public    | https://10.23.100.127:8774/v2.1/%(project_id)s      |
| [ID truncated] | RegionOne | keystone     | identity       | True    |␣
↪internal  | http://172.29.236.100:5000/v3                       |
| [ID truncated] | RegionOne | nova         | compute        | True    |␣
↪internal  | http://172.29.236.100:8774/v2.1/%(project_id)s      |
| [ID truncated] | RegionOne | gnocchi      | metric         | True    |␣
↪public    | https://10.23.100.127:8041                          |
| [ID truncated] | RegionOne | neutron      | network        | True    |␣
↪internal  | http://172.29.236.100:9696                          |
| [ID truncated] | RegionOne | aodh         | alarming       | True    |␣
↪admin     | http://172.29.236.100:8042                          |
| [ID truncated] | RegionOne | heat         | orchestration  | True    |␣
↪admin     | http://172.29.236.100:8004/v1/%(project_id)s        |
| [ID truncated] | RegionOne | glance       | image          | True    |␣
↪internal  | http://172.29.236.100:9292                          |
| [ID truncated] | RegionOne | designate    | dns            | True    |␣
↪internal  | http://172.29.236.100:9001                          |
| [ID truncated] | RegionOne | cinderv3     | volume         | True    |␣
↪internal  | http://172.29.236.100:8776/v3/%(project_id)s        |
| [ID truncated] | RegionOne | heat-cfn     | cloudformation | True    |␣
↪public    | https://10.23.100.127:8000/v1                       |
| [ID truncated] | RegionOne | designate    | dns            | True    |␣
↪public    | https://10.23.100.127:9001                          |
| [ID truncated] | RegionOne | swift        | object-store   | True    |␣
↪admin     | http://172.29.236.100:8080/v1/AUTH_%(project_id)s |
| [ID truncated] | RegionOne | heat         | orchestration  | True    |␣
```

(continues on next page)

(continued from previous page)

```
↪internal   | http://172.29.236.100:8004/v1/%(project_id)s      |
| [ID truncated] | RegionOne | cinderv3     | volumev3       | True    |␣
↪admin      | http://172.29.236.100:8776/v3/%(project_id)s      |
| [ID truncated] | RegionOne | swift        | object-store   | True    |␣
↪internal   | http://172.29.236.100:8080/v1/AUTH_%(project_id)s |
| [ID truncated] | RegionOne | neutron      | network        | True    |␣
↪public     | https://10.23.100.127:9696                        |
| [ID truncated] | RegionOne | heat         | orchestration  | True    |␣
↪public     | https://10.23.100.127:8004/v1/%(project_id)s      |
| [ID truncated] | RegionOne | gnocchi      | metric         | True    |␣
↪admin      | http://172.29.236.100:8041                        |
| [ID truncated] | RegionOne | gnocchi      | metric         | True    |␣
↪internal   | http://172.29.236.100:8041                        |
| [ID truncated] | RegionOne | keystone     | identity       | True    |␣
↪public     | https://10.23.100.127:5000/v3                     |
| [ID truncated] | RegionOne | glance       | image          | True    |␣
↪admin      | http://172.29.236.100:9292                        |
| [ID truncated] | RegionOne | placement    | placement      | True    |␣
↪internal   | http://172.29.236.100:8780                        |
| [ID truncated] | RegionOne | placement    | placement      | True    |␣
↪admin      | http://172.29.236.100:8780                        |
| [ID truncated] | RegionOne | placement    | placement      | True    |␣
↪public     | https://10.23.100.127:8780                        |
+----------------+-----------+--------------+----------------+---------+--
↪---------+---------------------------------------------------+
```

7. Run an OpenStack command to ensure all the compute services are working (the output depends on your configuration) For example:

```
$ openstack compute service list
+----+------------------+-----------------------------------------+--------
↪--+---------+-------+----------------------------+
| ID | Binary           | Host                                    | Zone  ␣
↪   | Status  | State | Updated At                 |
+----+------------------+-----------------------------------------+--------
↪--+---------+-------+----------------------------+
|  1 | nova-conductor   | aio1-nova-conductor-container-5482ff27 |␣
↪internal | enabled | up    | 2018-02-14T15:34:42.000000 |
|  2 | nova-scheduler   | aio1-nova-scheduler-container-0b594e89 |␣
↪internal | enabled | up    | 2018-02-14T15:34:47.000000 |
|  5 | nova-consoleauth | aio1-nova-console-container-835ca240   |␣
↪internal | enabled | up    | 2018-02-14T15:34:47.000000 |
|  6 | nova-compute     | ubuntu-focal                            | nova  ␣
↪   | enabled | up    | 2018-02-14T15:34:42.000000 |
+----+------------------+-----------------------------------------+--------
↪--+---------+-------+----------------------------+
```

8. Run an OpenStack command to ensure the networking services are working (the output also depends on your configuration) For example:

```
$ openstack network agent list
+--------------------------------------+---------------------+-----------
↪----------------------------+-------------------+-------+-------+------
↪---------------------+
| ID                                   | Agent Type          | Host     ␣
↪                            | Availability Zone | Alive | State |␣
↪Binary                      |
+--------------------------------------+---------------------+-----------
↪----------------------------+-------------------+-------+-------+------
↪---------------------+
| 262b29fe-e60e-44b0-ae3c-065565f8deb7 | Metering agent      | aio1-
↪neutron-agents-container-2b0569d5 | None              | :-)   | UP    |␣
↪neutron-metering-agent    |
| 41135f7f-9e6c-4122-b6b3-d131bfaae53e | Open vSwitch agent  | ubuntu-
↪focal                           | None              | :-)   | UP    |␣
↪neutron-openvswitch-agent |
| 615d12a8-e738-490a-8552-2a03c8544b51 | Metadata agent      | aio1-
↪neutron-agents-container-2b0569d5 | None              | :-)   | UP    |␣
↪neutron-metadata-agent    |
| 99b2abd3-a330-4ca7-b524-ed176c10b31c | DHCP agent          | aio1-
↪neutron-agents-container-2b0569d5 | nova              | :-)   | UP    |␣
↪neutron-dhcp-agent        |
| e0139a26-fbf7-4cee-a37f-90940dc5851f | Open vSwitch agent  | aio1-
↪neutron-agents-container-2b0569d5 | None              | :-)   | UP    |␣
↪neutron-openvswitch-agent |
| feb20ed4-4346-4ad9-b50c-41efd784f2e9 | L3 agent            | aio1-
↪neutron-agents-container-2b0569d5 | nova              | :-)   | UP    |␣
↪neutron-l3-agent          |
+--------------------------------------+---------------------+-----------
↪----------------------------+-------------------+-------+-------+------
↪---------------------+
```

9. Run an OpenStack command to ensure the block storage services are working (depends on your configuration). For example:

```
$ openstack volume service list
+------------------+------------------------------------------+------+----
↪-----+-------+----------------------------+
| Binary           | Host                                     | Zone |␣
↪Status  | State | Updated At                 |
+------------------+------------------------------------------+------+----
↪-----+-------+----------------------------+
| cinder-scheduler | aio1-cinder-scheduler-container-ff4c6c1e | nova |␣
↪enabled | up    | 2018-02-14T15:37:21.000000 |
| cinder-volume    | ubuntu-bionic@lvm                        | nova |␣
↪enabled | up    | 2018-02-14T15:37:25.000000 |
| cinder-backup    | ubuntu-bionic                            | nova |␣
↪enabled | up    | 2018-02-14T15:37:21.000000 |
+------------------+------------------------------------------+------+----
↪-----+-------+----------------------------+
```

10. Run an OpenStack command to ensure the image storage service is working (depends on your uploaded images). For example:

```
$ openstack image list
+--------------------------------------+--------+--------+
| ID                                   | Name   | Status |
+--------------------------------------+--------+--------+
| 6092d7b3-87c1-4d6c-a822-66c0c6171bd3 | cirros | active |
+--------------------------------------+--------+--------+
```

11. Check the backend API health on your load balancer nodes. For example, if using HAProxy, ensure no backend is marked as DOWN:

```
$ hatop -s /var/run/haproxy.stat
```

Configure your first networks

A newly deployed OpenStack-Ansible has no networks by default. If you need to add networks, you can use the OpenStack CLI, or you can use the Ansible modules for it.

An example on how to provision networks is in the OpenStack-Ansible plugins repository, where you can use the openstack_resources role:

1. Define the variable openstack_resources_network according to the structure in the role *defaults <https://opendev.org/openstack/openstack-ansible-plugins/src/branch/master/roles/openstack_resources/defaults/main.yml#L100-L143>*
2. Run the playbook openstack.osa.openstack_resources with the tag network-resources:

```
openstack-ansible openstack.osa.openstack_resources --tags network-
↪resources
```

Use the command line clients

This section describes some of the more common commands to use your OpenStack cloud.

Log in to any utility container or install the OpenStack client on your machine, and run the following commands:

The **openstack flavor list** command lists the *flavors* that are available. These are different disk sizes that can be assigned to images:

```
$ openstack flavor list
+-----+-----------+-------+------+-----------+-------+-----------+
| ID  | Name      |   RAM | Disk | Ephemeral | vCPUs | Is Public |
+-----+-----------+-------+------+-----------+-------+-----------+
| 1   | m1.tiny   |   512 |    1 |         0 |     1 | True      |
| 2   | m1.small  |  2048 |   20 |         0 |     1 | True      |
| 3   | m1.medium |  4096 |   40 |         0 |     2 | True      |
| 4   | m1.large  |  8192 |   80 |         0 |     4 | True      |
| 5   | m1.xlarge | 16384 |  160 |         0 |     8 | True      |
+-----+-----------+-------+------+-----------+-------+-----------+
```

The **openstack floating ip list** command lists the currently available floating IP addresses and the instances they are associated with:

```
$ openstack floating ip list
+------------------+---------------------+------------------+---------+-------
↪------------+---------------+
| ID               | Floating IP Address | Fixed IP Address | Port    |␣
↪Floating Network  | Project       |
+------------------+---------------------+------------------+---------+-------
↪------------+---------------+
| 0a88589a-ffac... | 192.168.12.7        | None             | None    |␣
↪d831dac6-028c...  | 32db2ccf2a... |
+------------------+---------------------+------------------+---------+-------
↪------------+---------------+
```

For more information about OpenStack client utilities, see these links:

- OpenStack API Quick Start
- OpenStackClient commands
- Compute (nova) CLI commands
- Compute (nova) CLI command cheat sheet

Managing your cloud

This chapter is intended to document OpenStack operations tasks that are integral to the operations support in an OpenStack-Ansible deployment.

It explains operations such as managing images, instances, or networks.

Managing images

An image represents the operating system, software, and any settings that instances may need depending on the project goals. Create images first before creating any instances.

Adding images can be done through the Dashboard, or the command line. Another option available is the `python-openstackclient` tool, which can be installed on the controller node, or on a workstation.

Adding an image using the Dashboard

In order to add an image using the Dashboard, prepare an image binary file, which must be accessible over HTTP using a valid and direct URL. Images can be compressed using `.zip` or `.tar.gz`.

Note

Uploading images using the Dashboard will be available to users with administrator privileges. Operators can set user access privileges.

1. Log in to the Dashboard.
2. Select the *Admin* tab in the navigation pane and click *Images*.
3. Click the *Create Image* button. The **Create an Image** dialog box will appear.

4. Enter the details of the image, including the **Image Location**, which is where the URL location of the image is required.
5. Click the *Create Image* button. The newly created image may take some time before it is completely uploaded since the image arrives in an image queue.

Adding an image using the command line

The utility container provides a CLI environment for additional configuration and management.

1. Access the utility container:

```
$ lxc-attach -n `lxc-ls -1 | grep utility | head -n 1`
```

Use the OpenStack client within the utility container to manage all glance images. See the OpenStack client official documentation on managing images.

Managing instances

This chapter describes how to create and access instances.

Creating an instance using the Dashboard

Using an image, create a new instance via the Dashboard options.

1. Log into the Dashboard, and select the *admin* project from the drop down list.
2. On the *Project* tab, open the *Instances* tab and click the *Launch Instance* button.

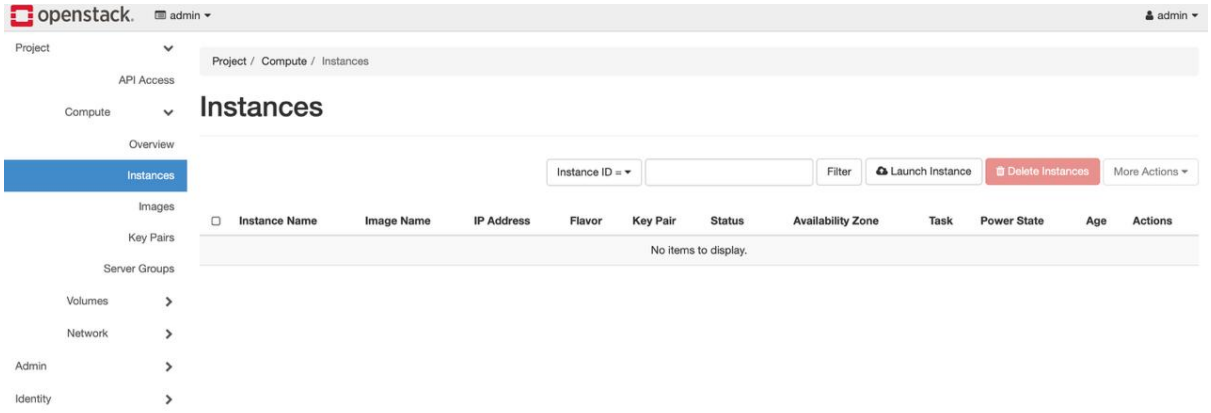


Fig. 1: **Figure Dashboard Instances tab**

3. Check the *Launch Instance* dialog, and find the *Details* tab. Enter the appropriate values for the instance.
4. Click the *Source*. In the Source step, select the boot source: Image, Volume (Volume Snapshot), or Instance Snapshot. If you choose Image, pick the desired OS or custom image from the list to boot your instance. Volume option will only be available if Block Storage service (cinder) is enabled.

   For more information on attaching Block Storage volumes to instances for persistent storage, see the *Managing volumes for persistent storage* section below.
5. In the *Launch Instance* dialog, click the *Flavor* tab and select the prefered flavor for you instance.

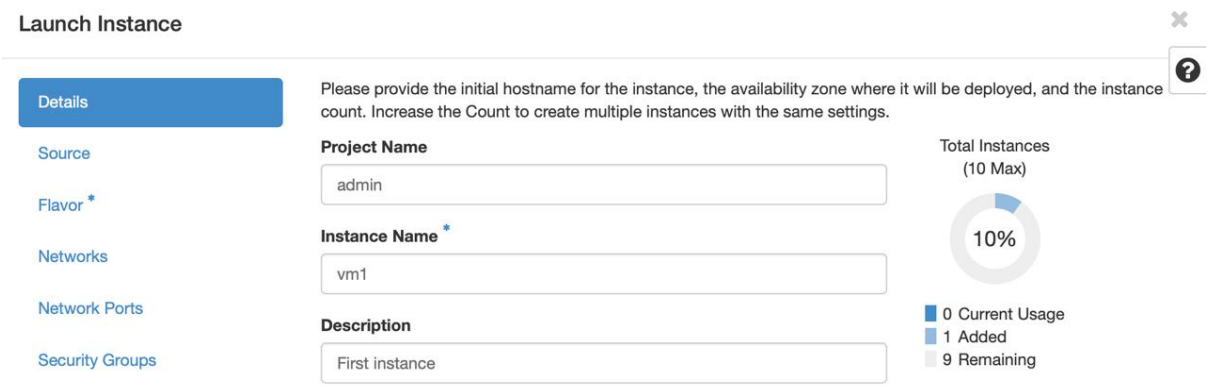


Fig. 2: **Instance Details**

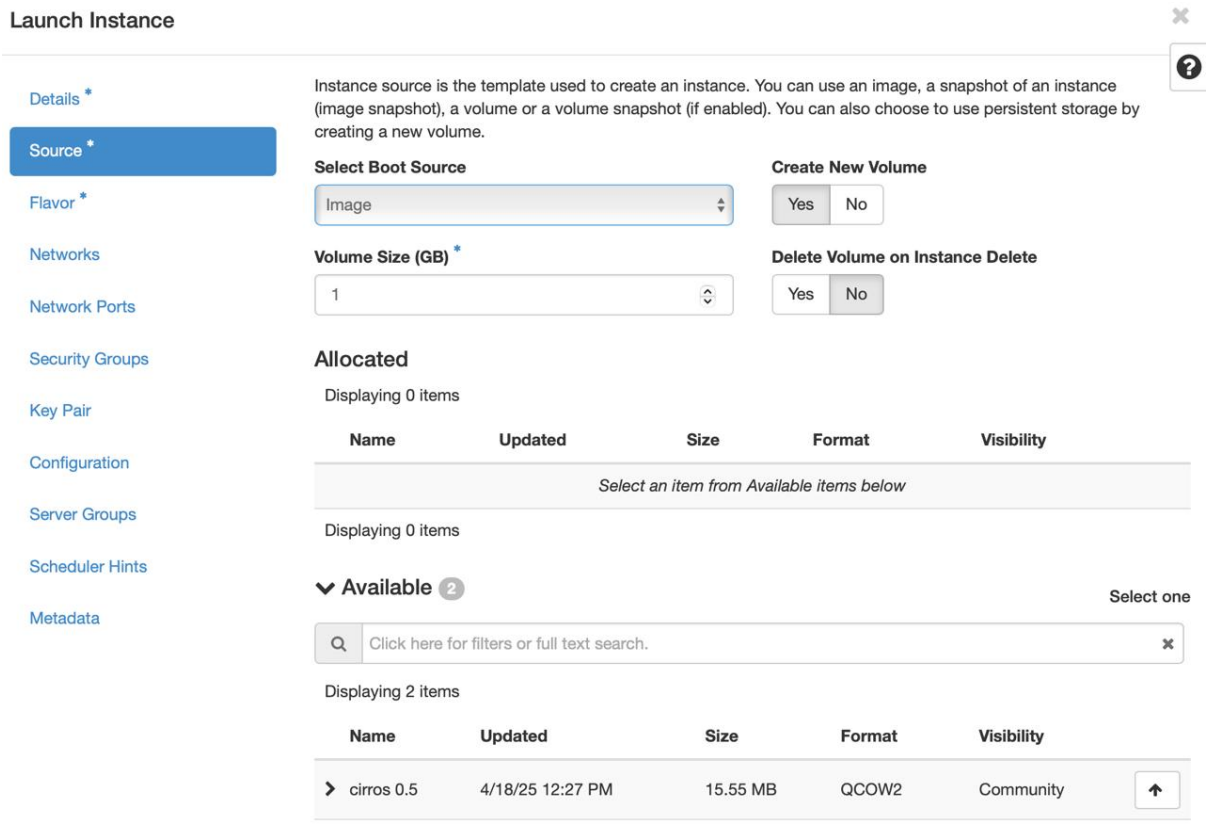


Fig. 3: **Instance Source**

6. Click the *Networks tab*. This tab will be unavailable if Network service (neutron) has not been enabled. If networking is enabled, select the networks on which the instance will reside.

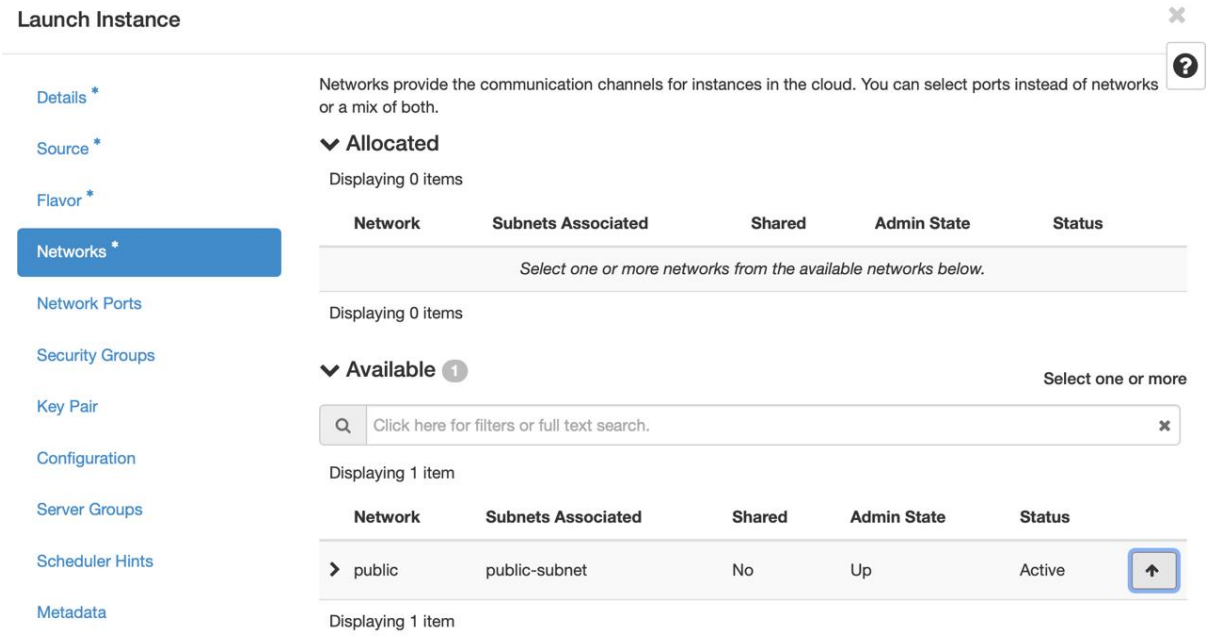


Fig. 4: **Instance Networks**

7. Click the *Keypair* tab and select the keypair or create new one.
8. Click the *Security Groups* tab and set the security group as default.
9. Add customisation scripts, if needed, by clicking the *Configuration*. These run after the instance has been created. Some instances support user data, such as root passwords, or admin users. Enter the information specific to the instance here if required.
10. Click *Launch* to create the instance. The instance will start on a compute node. The **Instances** page will open and start creating a new instance. The **Instances** page that opens will list the instance name, size, status, and task. Power state and public and private IP addresses are also listed here.

    The process will take less than a minute to complete. Instance creation is complete when the status is listed as active. Refresh the page to see the new active instance.

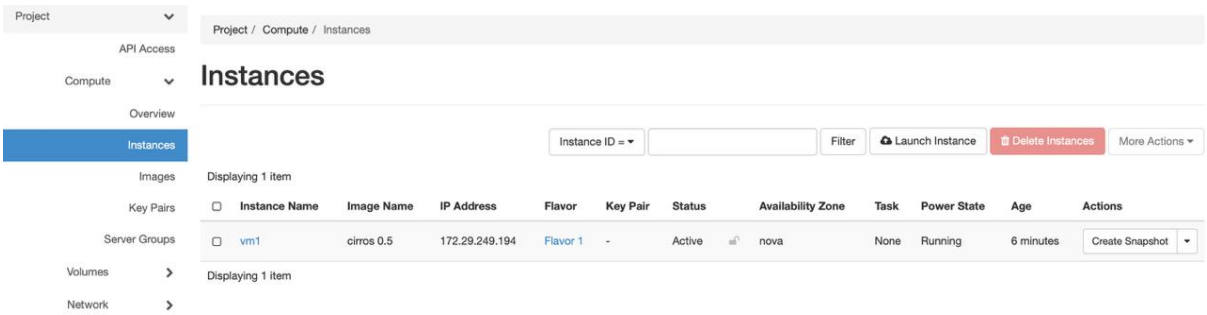


Fig. 5: **Instances Page**

Table 1: **Launching an instance options**

Field Name	Required	Details
Availability Zone	Optional	The availability zone in which the image service creates the instance. If no availability zones is defined, no instances will be found. The cloud provider sets the availability zone to a specific value.
Instance Name	Required	The name of the new instance, which becomes the initial host name of the server. If the server name is changed in the API or directly changed, the Dashboard names remain unchanged
Image	Required	The type of container format, one of `raw`, `qcow2`, `iso`, `vmdk`,"vdi" etc.
Flavor	Required	The vCPU, Memory, and Disk configuration. Note that larger flavors can take a long time to create. If creating an instance for the first time and want something small with which to test, select `m1.small`.
Instance Count	Required	If creating multiple instances with this configuration, enter an integer up to the number permitted by the quota, which is `10` by default.
Instance Boot Source	Required	Specify whether the instance will be based on an image or a snapshot. If it is the first time creating an instance, there will not yet be any snapshots available.
Image	Required	The instance will boot from the selected image. This option will be pre-populated with the instance selected from the table. However, choose `Boot from Snapshot` in **Instance Boot Source**, and it will default to `Snapshot` instead.
Security Groups	Optional	This option assigns security groups to an instance. The default security group activates when no customised group is specified here. Security Groups, similar to a cloud firewall, define which incoming network traffic is

Creating an instance using the command line

On the command line, instance creation is managed with the **openstack server create** command. Before launching an instance, determine what images and flavors are available to create a new instance using the **openstack image list** and **openstack flavor list** commands.

1. Log in to any utility container.
2. Issue the **openstack server create** command with a name for the instance, along with the name of the image and flavor to use:

```
$ openstack server create --image precise-image --flavor 2 --key-name␣
↪example-key example-instance
+--------------------------------------+----------------------------------
↪---+
|                Property              |                Value             ␣
↪    |
+--------------------------------------+----------------------------------
↪---+
|           OS-DCF:diskConfig          |                MANUAL            ␣
↪    |
|         OS-EXT-SRV-ATTR:host         |                 None             ␣
↪    |
| OS-EXT-SRV-ATTR:hypervisor_hostname  |                 None             ␣
↪    |
|    OS-EXT-SRV-ATTR:instance_name     |          instance-0000000d       ␣
↪    |
|        OS-EXT-STS:power_state        |                  0               ␣
↪    |
|        OS-EXT-STS:task_state         |              scheduling          ␣
↪    |
|         OS-EXT-STS:vm_state          |               building           ␣
↪    |
|              accessIPv4              |                                  ␣
↪    |
|              accessIPv6              |                                  ␣
↪    |
|              adminPass               |             ATSEfRY9fZPx         ␣
↪    |
|             config_drive             |                                  ␣
↪    |
|               created                |         2012-08-02T15:43:46Z     ␣
↪    |
|                flavor                |               m1.small           ␣
↪    |
|                hostId                |                                  ␣
↪    |
|                  id                  | 5bf46a3b-084c-4ce1-b06f-
↪e460e875075b |
|                image                 |             precise-image        ␣
↪    |
|               key_name               |              example-key         ␣
```

(continues on next page)

(continued from previous page)

```
↪   |
|               metadata              |                  {}              ␣
↪   |
|                 name                |           example-instance       ␣
↪   |
|               progress              |                  0               ␣
↪   |
|                status               |                BUILD             ␣
↪   |
|              tenant_id              |  ␣
↪b4769145977045e2a9279c842b09be6a   |
|               updated               |         2012-08-02T15:43:46Z     ␣
↪   |
|               user_id               |  ␣
↪5f2f2c28bdc844f9845251290b524e80   |
+-------------------------------------+----------------------------------
↪---+
```

3. To check that the instance was created successfully, issue the **openstack server list** command:

```
$ openstack server list
+------------------+------------------+--------+------------------+------
↪---------+
|        ID        |       Name       | Status |      Networks    |  ␣
↪Image Name  |
+------------------+------------------+--------+------------------+------
↪---------+
| [ID truncated]   | example-instance | ACTIVE |  public=192.0.2.0 |␣
↪precise-image |
+------------------+------------------+--------+------------------+------
↪---------+
```

Managing an instance

1. Log in to the Dashboard. Select one of the projects, and click *Instances*.
2. Select an instance from the list of available instances.
3. Check the **Actions** column, and click on the down arrow. Select the action.

The **Actions** column includes the following options:

- Resize or rebuild any instance
- Attach/Detach Volume
- Attach/Detach Interface
- View the instance console log
- Edit the instance
- Edit security groups
- Pause, resume, rescue or suspend the instance

- Soft or hard reset the instance

Note

Delete the instance under the **Actions** column.

Managing volumes for persistent storage

Volumes attach to instances, enabling persistent storage. Volume storage provides a source of memory for instances. Administrators can attach volumes to a running instance, or move a volume from one instance to another.

Instances live migration

Nova is capable of live migration instances from one host to a different host to support various operational tasks including:

- Host Maintenance
- Host capacity management
- Resizing and moving instances to better hardware

Nova configuration drive implication

Depending on the OpenStack-Ansible version in use, Nova can be configured to force configuration drive attachments to instances. In this case, a ISO9660 CD-ROM image will be made available to the instance via the `/mnt` mount point. This can be used by tools, such as cloud-init, to gain access to instance metadata. This is an alternative way of accessing the Nova EC2-style Metadata.

To allow live migration of Nova instances, this forced provisioning of the config (CD-ROM) drive needs to either be turned off, or the format of the configuration drive needs to be changed to a disk format like vfat, a format which both Linux and Windows instances can access.

This work around is required for all Libvirt versions prior 1.2.17.

To turn off the forced provisioning of and change the format of the configuration drive to a hard disk style format, add the following override to the `/etc/openstack_deploy/user_variables.yml` file:

```
nova_nova_conf_overrides:
  DEFAULT:
    config_drive_format: vfat
    force_config_drive: false
```

Tunneling versus direct transport

In the default configuration, Nova determines the correct transport URL for how to transfer the data from one host to the other. Depending on the `nova_virt_type` override the following configurations are used:

- kvm defaults to `qemu+tcp://%s/system`
- qemu defaults to `qemu+tcp://%s/system`

Libvirt TCP port to transfer the data to migrate.

OpenStack-Ansible changes the default setting and used a encrypted SSH connection to transfer the instance data.

```
live_migration_uri = "qemu+ssh://nova@%s/system?no_verify=1&keyfile={{ nova_
↪system_home_folder }}/.ssh/id_rsa"
```

Other configurations can be configured inside the `/etc/openstack_deploy/user_variables.yml` file:

```
nova_nova_conf_overrides:
  libvirt:
    live_migration_completion_timeout: 0
    live_migration_progress_timeout: 0
    live_migration_uri: "qemu+ssh://nova@%s/system?keyfile=/var/lib/nova/.ssh/
↪id_rsa&no_verify=1"
```

Local versus shared storage

By default, live migration assumes that your instances are stored on shared storage and KVM/Libvirt only need to synchronize the memory and base image of the instance to the new host. Live migrations on local storage will fail as a result of that assumption. Migrations with local storage can be accomplished by allowing instance disk migrations with the `--block-migrate` option.

Additional flavor features like ephemeral storage or swap have an impact on live migration performance and success.

Cinder attached volumes also require a Libvirt version larger or equal to 1.2.17.

Executing the migration

The live migration is accessible via the nova client.

```
nova live-migration [--block-migrate] [--force] <uuid> [<host>]
```

Examplarery live migration on a local storage:

```
nova live-migration --block-migrate <uuid of the instance> <nova host>
```

Monitoring the status

Once the live migration request has been accepted, the status can be monitored with the nova client:

```
nova migration-list

+-----+------------+-----------+----------------+--------------+-----------+--
↪---------+--------------+------------+------------+------------+-----------
↪-+-----------------+
| Id | Source Node | Dest Node | Source Compute | Dest Compute | Dest Host |␣
↪Status    | Instance UUID | Old Flavor | New Flavor | Created At | Updated␣
↪At | Type           |
```

(continues on next page)

(continued from previous page)

```
+----+-------------+-----------+---------------+--------------+-----------+--
↪--------+--------------+------------+------------+------------+-----------
↪-+-----------------+
| 6  | -           | -         | compute01     | compute02    | -         |␣
↪preparing | f95ee17a-d09c | 7          | 7          | date       | date      ␣
↪  | live-migration  |
+----+-------------+-----------+---------------+--------------+-----------+--
↪--------+--------------+------------+------------+------------+-----------
↪-+-----------------+
```

To filter the list, the options `--host` or `--status` can be used:

```
nova migration-list --status error
```

In cases where the live migration fails, both the source and destination compute nodes need to be checked for errors. Usually it is sufficient to search for the instance UUID only to find errors related to the live migration.

Other forms of instance migration

Besides the live migration, Nova offers the option to migrate entire hosts in a online (live) or offline (cold) migration.

The following nova client commands are provided:

- `host-evacuate-live`

  Live migrate all instances of the specified host to other hosts if resource utilzation allows. It is best to use shared storage like Ceph or NFS for host evacuation.

- `host-servers-migrate`

  This command is similar to host evacuation but migrates all instances off the specified host while they are shutdown.

- `resize`

  Changes the flavor of an instance (increase) while rebooting and also migrates (cold) the instance to a new host to accommodate the new resource requirements. This operation can take considerate amount of time, depending disk image sizes.

Managing networks

Operational considerations, like compliance, can make it necessary to manage networks. For example, adding new provider networks to the OpenStack-Ansible managed cloud. The following sections are the most common administrative tasks outlined to complete those tasks.

For more generic information on troubleshooting your network, see the Network Troubleshooting chapter in the Operations Guide.

For more in-depth information on Networking, see the Networking Guide.

Add provider bridges using new network interfaces

Add each provider network to your cloud to be made known to OpenStack-Ansible and the operating system before you can execute the necessary playbooks to complete the configuration.

OpenStack-Ansible configuration

All provider networks need to be added to the OpenStack-Ansible configuration.

Edit the file `/etc/openstack_deploy/openstack_user_config.yml` and add a new block underneath the `provider_networks` section:

The `container_bridge` setting defines the physical network bridge used to connect the veth pair from the physical host to the container. Inside the container, the `container_interface` setting defines the name at which the physical network will be made available. The `container_interface` setting is not required when Neutron agents are deployed on bare metal. Make sure that both settings are uniquely defined across their provider networks and that the network interface is correctly configured inside your operating system. `group_binds` define where this network need to attached to, to either containers or physical hosts and is ultimately dependent on the network stack in use. For example, Linuxbridge versus OVS. The configuration `range` defines Neutron physical segmentation IDs which are automatically used by end users when creating networks via mainly horizon and the Neutron API. Similar is true for the `net_name` configuration which defines the addressable name inside the Neutron configuration. This configuration also need to be unique across other provider networks.

For more information, see Configure the deployment in the OpenStack-Ansible Deployment Guide.

Updating the node with the new configuration

Run the appropriate playbooks depending on the `group_binds` section.

For example, if you update the networks requiring a change in all nodes with a linux bridge agent, assuming you have infra nodes named **infra01**, **infra02**, and **infra03**, run:

```
# openstack-ansible containers-deploy.yml --limit localhost,infra01,infra01-
↪host_containers
# openstack-ansible containers-deploy.yml --limit localhost,infra02,infra02-
↪host_containers
# openstack-ansible containers-deploy.yml --limit localhost,infra03,infra03-
↪host_containers
```

Then update the neutron configuration.

```
# openstack-ansible os-neutron-install.yml --limit localhost,infra01,infra01-
↪host_containers
# openstack-ansible os-neutron-install.yml --limit localhost,infra02,infra02-
↪host_containers
# openstack-ansible os-neutron-install.yml --limit localhost,infra03,infra03-
↪host_containers
```

Then update your compute nodes if necessary.

Remove provider bridges from OpenStack

Similar to adding a provider network, the removal process uses the same procedure but in a reversed order. The Neutron ports will need to be removed, prior to the removal of the OpenStack-Ansible configuration.

1. Unassign all Neutron floating IPs:

   Note

   Export the Neutron network that is about to be removed as single UUID.

```
export NETWORK_UUID=<uuid>
for p in $( neutron port-list -c id --device_owner compute:nova --network_
↪id=${NETWORK_UUID}| awk '/([A-Fa-f0-9]+-){3}/ {print $2}' ); do
  floatid=$( neutron floatingip-list -c id --port_id=$p | awk '/([A-Fa-z0-
↪9]+-){3}/ { print $2 }' )
  if [ -n "$floatid" ]; then
    echo "Disassociating floating IP $floatid from port $p"
    neutron floatingip-disassociate $floatid
  fi
done
```

2. Remove all Neutron ports from the instances:

```
export NETWORK_UUID=<uuid>
for p in $( neutron port-list -c id -c device_id --device_owner␣
↪compute:nova --network_id=${NETWORK_UUID}| awk '/([A-Fa-f0-9]+-){3}/
↪{print $2}' ); do
  echo "Removing Neutron compute port $p"
  neutron port-delete $p
done
```

3. Remove Neutron router ports and DHCP agents:

```
export NETWORK_UUID=<uuid>
for line in $( neutron port-list -c id -c device_id --device_owner␣
↪network:router_interface --network_id=${NETWORK_UUID}| awk '/([A-Fa-f0-
↪9]+-){3}/ {print $2 "+" $4}' ); do
  p=$( echo "$line"| cut -d'+' -f1 ); r=$( echo "$line"| cut -d'+' -f2 )
  echo "Removing Neutron router port $p from $r"
  neutron router-interface-delete $r port=$p
done

for agent in $( neutron agent-list -c id --agent_type='DHCP Agent' --
↪network_id=${NETWORK_UUID}| awk '/([A-Fa-f0-9]+-){3}/ {print $2}' ); do
  echo "Remove network $NETWORK_UUID from Neutron DHCP Agent $agent"
  neutron dhcp-agent-network-remove "${agent}" $NETWORK_UUID
done
```

4. Remove the Neutron network:

```
export NETWORK_UUID=<uuid>
neutron net-delete $NETWORK_UUID
```

5. Remove the provider network from the `provider_networks` configuration of the OpenStack-Ansible configuration `/etc/openstack_deploy/openstack_user_config.yml` and re-run the following playbooks:

```
# openstack-ansible lxc-containers-create.yml --limit infra01:infra01-
↪host_containers
# openstack-ansible lxc-containers-create.yml --limit infra02:infra02-
↪host_containers
# openstack-ansible lxc-containers-create.yml --limit infra03:infra03-
↪host_containers
# openstack-ansible os-neutron-install.yml --tags neutron-config
```

Restart a Networking agent container

Under some circumstances, configuration or temporary issues, one specific or all neutron agents container need to be restarted.

This can be accomplished with multiple commands:

1. Example of rebooting still accessible containers.

   This example will issue a reboot to the container named with `neutron_agents_container_hostname_name` from inside:

```
# ansible -m shell neutron_agents_container_hostname_name -a 'reboot'
```

2. Example of rebooting one container at a time, 60 seconds apart:

```
# ansible -m shell neutron_agents_container -a 'sleep 60; reboot' --forks␣
↪1
```

3. If the container does not respond, it can be restarted from the physical network host:

```
# ansible -m shell network_hosts -a 'for c in $(lxc-ls -1 |grep neutron_
↪agents_container); do lxc-stop -n $c && lxc-start -d -n $c; done' --
↪forks 1
```

Maintenance tasks

This chapter is intended for OpenStack-Ansible specific maintenance tasks.

Galera cluster maintenance

Routine maintenance includes gracefully adding or removing nodes from the cluster without impacting operation and also starting a cluster after gracefully shutting down all nodes.

MariaDB instances are restarted when creating a cluster, when adding a node, when the service is not running, or when changes are made to the `/etc/mysql/my.cnf` configuration file.

Verify cluster status

Compare the output of the following command with the following output. It should give you information about the status of your cluster.

```
# ansible galera_container -m shell -a "mariadb \
-e 'show status like \"%wsrep_cluster_%\";'"
node3_galera_container-3ea2cbd3 | FAILED | rc=1 >>
ERROR 2002 (HY000): Can't connect to local MySQL server
through socket '/var/run/mysqld/mysqld.sock' (2)

node2_galera_container-49a47d25 | FAILED | rc=1 >>
ERROR 2002 (HY000): Can't connect to local MySQL server
through socket '/var/run/mysqld/mysqld.sock' (2)

node4_galera_container-76275635 | success | rc=0 >>
Variable_name             Value
wsrep_cluster_conf_id     7
wsrep_cluster_size        1
wsrep_cluster_state_uuid  338b06b0-2948-11e4-9d06-bef42f6c52f1
wsrep_cluster_status      Primary
```

In this example, only one node responded.

Gracefully shutting down the MariaDB service on all but one node allows the remaining operational node to continue processing SQL requests. When gracefully shutting down multiple nodes, perform the actions sequentially to retain operation.

Start a cluster

Gracefully shutting down all nodes destroys the cluster. Starting or restarting a cluster from zero nodes requires creating a new cluster on one of the nodes.

1. Start a new cluster on the most advanced node. Change to the `playbooks` directory and check the `seqno` value in the `grastate.dat` file on all of the nodes:

```
# ansible galera_container -m shell -a "cat /var/lib/mysql/grastate.dat"
node2_galera_container-49a47d25 | success | rc=0 >>
# GALERA saved state version: 2.1
uuid:    338b06b0-2948-11e4-9d06-bef42f6c52f1
seqno:   31
cert_index:

node3_galera_container-3ea2cbd3 | success | rc=0 >>
# GALERA saved state version: 2.1
uuid:    338b06b0-2948-11e4-9d06-bef42f6c52f1
seqno:   31
cert_index:

node4_galera_container-76275635 | success | rc=0 >>
# GALERA saved state version: 2.1
uuid:    338b06b0-2948-11e4-9d06-bef42f6c52f1
```

(continues on next page)

(continued from previous page)

```
seqno:   31
cert_index:
```

In this example, all nodes in the cluster contain the same positive `seqno` values as they were synchronized just prior to graceful shutdown. If all `seqno` values are equal, any node can start the new cluster.

```
## for init
# /etc/init.d/mariadb start --wsrep-new-cluster
## for systemd
# systemctl set-environment _WSREP_NEW_CLUSTER='--wsrep-new-cluster'
# systemctl start mariadb
# systemctl set-environment _WSREP_NEW_CLUSTER=''
```

Please also have a look at Starting the Cluster.

This can also be done with the help of Ansible using the shell module:

```
# ansible galera_container -m shell -a "/etc/init.d/mariadb start --wsrep-
↪new-cluster" --limit galera_container[0]
```

This command results in a cluster containing a single node. The `wsrep_cluster_size` value shows the number of nodes in the cluster.

```
node2_galera_container-49a47d25 | FAILED | rc=1 >>
ERROR 2002 (HY000): Can't connect to local MySQL server
through socket '/var/run/mysqld/mysqld.sock' (111)

node3_galera_container-3ea2cbd3 | FAILED | rc=1 >>
ERROR 2002 (HY000): Can't connect to local MySQL server
through socket '/var/run/mysqld/mysqld.sock' (2)

node4_galera_container-76275635 | success | rc=0 >>
Variable_name            Value
wsrep_cluster_conf_id    1
wsrep_cluster_size       1
wsrep_cluster_state_uuid 338b06b0-2948-11e4-9d06-bef42f6c52f1
wsrep_cluster_status     Primary
```

2. Restart MariaDB on the other nodes (replace [0] from previous Ansible command with [1:]) and verify that they rejoin the cluster.

```
node2_galera_container-49a47d25 | success | rc=0 >>
Variable_name            Value
wsrep_cluster_conf_id    3
wsrep_cluster_size       3
wsrep_cluster_state_uuid 338b06b0-2948-11e4-9d06-bef42f6c52f1
wsrep_cluster_status     Primary

node3_galera_container-3ea2cbd3 | success | rc=0 >>
Variable_name            Value
```

(continues on next page)

(continued from previous page)

```
wsrep_cluster_conf_id     3
wsrep_cluster_size        3
wsrep_cluster_state_uuid  338b06b0-2948-11e4-9d06-bef42f6c52f1
wsrep_cluster_status      Primary

node4_galera_container-76275635 | success | rc=0 >>
Variable_name             Value
wsrep_cluster_conf_id     3
wsrep_cluster_size        3
wsrep_cluster_state_uuid  338b06b0-2948-11e4-9d06-bef42f6c52f1
wsrep_cluster_status      Primary
```

Galera cluster recovery

Run the `openstack.osa.galera_server` playbook using the `galera_force_bootstrap` variable to automatically recover a node or an entire environment.

1. Run the following Ansible command to show the failed nodes:

```
# openstack-ansible openstack.osa.galera_server -e galera_force_
↪bootstrap=true --tags galera_server-config
```

You can additionally define a different bootstrap node through `galera_server_bootstrap_node` variable, in case current bootstrap node is in desynced/broken state. You can check what node is currently selected for bootstrap using this ad-hoc:

```
root@aio1:/opt/openstack-ansible# ansible -m debug -a var="groups[
↪'galera_all'][0]" localhost
```

The cluster comes back online after completion of this command. If this fails, please review restarting the cluster and recovering the primary component in the Galera documentation as theyre invaluable for a full cluster recovery.

Recover a single-node failure

If a single node fails, the other nodes maintain quorum and continue to process SQL requests.

1. Change to the `playbooks` directory and run the following Ansible command to determine the failed node:

```
# ansible galera_container -m shell -a "mariadb \
-e 'show status like \"%wsrep_cluster_%\";'"
node3_galera_container-3ea2cbd3 | FAILED | rc=1 >>
ERROR 2002 (HY000): Can't connect to local MySQL server through
socket '/var/run/mysqld/mysqld.sock' (111)

node2_galera_container-49a47d25 | success | rc=0 >>
Variable_name             Value
wsrep_cluster_conf_id     17
wsrep_cluster_size        3
wsrep_cluster_state_uuid  338b06b0-2948-11e4-9d06-bef42f6c52f1
```

(continues on next page)

(continued from previous page)

```
wsrep_cluster_status      Primary

node4_galera_container-76275635 | success | rc=0 >>
Variable_name             Value
wsrep_cluster_conf_id     17
wsrep_cluster_size        3
wsrep_cluster_state_uuid  338b06b0-2948-11e4-9d06-bef42f6c52f1
wsrep_cluster_status      Primary
```

In this example, node 3 has failed.

2. Restart MariaDB on the failed node and verify that it rejoins the cluster.
3. If MariaDB fails to start, run the `mariadbd` command and perform further analysis on the output. As a last resort, rebuild the container for the node.

Recover a multi-node failure

When all but one node fails, the remaining node cannot achieve quorum and stops processing SQL requests. In this situation, failed nodes that recover cannot join the cluster because it no longer exists.

1. Run the following Ansible command to show the failed nodes:

```
# ansible galera_container -m shell -a "mariadb \
-e 'show status like \"%wsrep_cluster_%\";'"
node2_galera_container-49a47d25 | FAILED | rc=1 >>
ERROR 2002 (HY000): Can't connect to local MySQL server
through socket '/var/run/mysqld/mysqld.sock' (111)

node3_galera_container-3ea2cbd3 | FAILED | rc=1 >>
ERROR 2002 (HY000): Can't connect to local MySQL server
through socket '/var/run/mysqld/mysqld.sock' (111)

node4_galera_container-76275635 | success | rc=0 >>
Variable_name             Value
wsrep_cluster_conf_id     18446744073709551615
wsrep_cluster_size        1
wsrep_cluster_state_uuid  338b06b0-2948-11e4-9d06-bef42f6c52f1
wsrep_cluster_status      non-Primary
```

In this example, nodes 2 and 3 have failed. The remaining operational server indicates `non-Primary` because it cannot achieve quorum.

2. Run the following command to rebootstrap the operational node into the cluster:

```
# mariadb -e "SET GLOBAL wsrep_provider_options='pc.bootstrap=yes';"
node4_galera_container-76275635 | success | rc=0 >>
Variable_name             Value
wsrep_cluster_conf_id     15
wsrep_cluster_size        1
wsrep_cluster_state_uuid  338b06b0-2948-11e4-9d06-bef42f6c52f1
wsrep_cluster_status      Primary
```

(continues on next page)

(continued from previous page)

```
node3_galera_container-3ea2cbd3 | FAILED | rc=1 >>
ERROR 2002 (HY000): Can't connect to local MySQL server
through socket '/var/run/mysqld/mysqld.sock' (111)

node2_galera_container-49a47d25 | FAILED | rc=1 >>
ERROR 2002 (HY000): Can't connect to local MySQL server
through socket '/var/run/mysqld/mysqld.sock' (111)
```

The remaining operational node becomes the primary node and begins processing SQL requests.

3. Restart MariaDB on the failed nodes and verify that they rejoin the cluster:

```
# ansible galera_container -m shell -a "mariadb \
-e 'show status like \"%wsrep_cluster_%\";'"
node3_galera_container-3ea2cbd3 | success | rc=0 >>
Variable_name             Value
wsrep_cluster_conf_id     17
wsrep_cluster_size        3
wsrep_cluster_state_uuid  338b06b0-2948-11e4-9d06-bef42f6c52f1
wsrep_cluster_status      Primary

node2_galera_container-49a47d25 | success | rc=0 >>
Variable_name             Value
wsrep_cluster_conf_id     17
wsrep_cluster_size        3
wsrep_cluster_state_uuid  338b06b0-2948-11e4-9d06-bef42f6c52f1
wsrep_cluster_status      Primary

node4_galera_container-76275635 | success | rc=0 >>
Variable_name             Value
wsrep_cluster_conf_id     17
wsrep_cluster_size        3
wsrep_cluster_state_uuid  338b06b0-2948-11e4-9d06-bef42f6c52f1
wsrep_cluster_status      Primary
```

4. If MariaDB fails to start on any of the failed nodes, run the `mariadbd` command and perform further analysis on the output. As a last resort, rebuild the container for the node.

Recover a complete environment failure

Restore from backup if all of the nodes in a Galera cluster fail (do not shutdown gracefully). Change to the `playbook` directory and run the following command to determine if all nodes in the cluster have failed:

```
# ansible galera_container -m shell -a "cat /var/lib/mysql/grastate.dat"
node3_galera_container-3ea2cbd3 | success | rc=0 >>
# GALERA saved state
version: 2.1
uuid:    338b06b0-2948-11e4-9d06-bef42f6c52f1
```

(continues on next page)

(continued from previous page)

```
seqno:   -1
cert_index:

node2_galera_container-49a47d25 | success | rc=0 >>
# GALERA saved state
version: 2.1
uuid:    338b06b0-2948-11e4-9d06-bef42f6c52f1
seqno:   -1
cert_index:

node4_galera_container-76275635 | success | rc=0 >>
# GALERA saved state
version: 2.1
uuid:    338b06b0-2948-11e4-9d06-bef42f6c52f1
seqno:   -1
cert_index:
```

All the nodes have failed if `mariadbd` is not running on any of the nodes and all of the nodes contain a `seqno` value of -1.

If any single node has a positive `seqno` value, then that node can be used to restart the cluster. However, because there is no guarantee that each node has an identical copy of the data, we do not recommend to restart the cluster using the `--wsrep-new-cluster` command on one node.

Rebuild a container

Recovering from certain failures require rebuilding one or more containers.

1. Disable the failed node on the load balancer.

   Note

   Do not rely on the load balancer health checks to disable the node. If the node is not disabled, the load balancer sends SQL requests to it before it rejoins the cluster and cause data inconsistencies.

2. Destroy the container and remove MariaDB data stored outside of the container:

   ```
   # openstack-ansible openstack.osa.containers_lxc_destroy \
   -l node3_galera_container-3ea2cbd3
   ```

   In this example, node 3 failed.

3. Run the host setup playbook to rebuild the container on node 3:

   ```
   # openstack-ansible oopenstack.osa.containers_lxc_create -l node3 \
   -l node3_galera_container-3ea2cbd3
   ```

   The playbook restarts all other containers on the node.

4. Run the infrastructure playbook to configure the container specifically on node 3:

```
# openstack-ansible openstack.osa.setup_infrastructure \
--limit node3_galera_container-3ea2cbd3
```

Warning

The new container runs a single-node Galera cluster, which is a dangerous state because the environment contains more than one active database with potentially different data.

```
# ansible galera_container -m shell -a "mariadb \
-e 'show status like \"%wsrep_cluster_%\";'"
node3_galera_container-3ea2cbd3 | success | rc=0 >>
Variable_name             Value
wsrep_cluster_conf_id     1
wsrep_cluster_size        1
wsrep_cluster_state_uuid  da078d01-29e5-11e4-a051-03d896dbdb2d
wsrep_cluster_status      Primary

node2_galera_container-49a47d25 | success | rc=0 >>
Variable_name             Value
wsrep_cluster_conf_id     4
wsrep_cluster_size        2
wsrep_cluster_state_uuid  338b06b0-2948-11e4-9d06-bef42f6c52f1
wsrep_cluster_status      Primary

node4_galera_container-76275635 | success | rc=0 >>
Variable_name             Value
wsrep_cluster_conf_id     4
wsrep_cluster_size        2
wsrep_cluster_state_uuid  338b06b0-2948-11e4-9d06-bef42f6c52f1
wsrep_cluster_status      Primary
```

5. Restart MariaDB in the new container and verify that it rejoins the cluster.

Note

In larger deployments, it may take some time for the MariaDB daemon to start in the new container. It will be synchronizing data from the other MariaDB servers during this time. You can monitor the status during this process by tailing the `journalctl -f -u mariadb` log file.

Lines starting with `WSREP_SST` will appear during the sync process and you should see a line with `WSREP: SST complete, seqno: <NUMBER>` if the sync was successful.

```
# ansible galera_container -m shell -a "mariadb \
-e 'show status like \"%wsrep_cluster_%\";'"
node2_galera_container-49a47d25 | success | rc=0 >>
Variable_name             Value
wsrep_cluster_conf_id     5
```

(continues on next page)

(continued from previous page)

```
wsrep_cluster_size        3
wsrep_cluster_state_uuid  338b06b0-2948-11e4-9d06-bef42f6c52f1
wsrep_cluster_status      Primary

node3_galera_container-3ea2cbd3 | success | rc=0 >>
Variable_name             Value
wsrep_cluster_conf_id     5
wsrep_cluster_size        3
wsrep_cluster_state_uuid  338b06b0-2948-11e4-9d06-bef42f6c52f1
wsrep_cluster_status      Primary

node4_galera_container-76275635 | success | rc=0 >>
Variable_name             Value
wsrep_cluster_conf_id     5
wsrep_cluster_size        3
wsrep_cluster_state_uuid  338b06b0-2948-11e4-9d06-bef42f6c52f1
wsrep_cluster_status      Primary
```

6. Enable the previously failed node on the load balancer.

RabbitMQ cluster maintenance

A RabbitMQ broker is a logical grouping of one or several Erlang nodes with each node running the RabbitMQ application and sharing users, virtual hosts, queues, exchanges, bindings, and runtime parameters. A collection of nodes is often referred to as a *cluster*. For more information on RabbitMQ clustering, see RabbitMQ cluster.

Within OpenStack-Ansible, all data and states required for operation of the RabbitMQ cluster is replicated across all nodes including the message queues providing high availability. RabbitMQ nodes address each other using domain names. The hostnames of all cluster members must be resolvable from all cluster nodes, as well as any machines where CLI tools related to RabbitMQ might be used. There are alternatives that may work in more restrictive environments. For more details on that setup, see Inet Configuration.

Create a RabbitMQ cluster

RabbitMQ clusters can be formed in two ways:

- Manually with `rabbitmqctl`
- Declaratively (list of cluster nodes in a config, with `rabbitmq-autocluster`, or `rabbitmq-clusterer` plugins)

Note

RabbitMQ brokers can tolerate the failure of individual nodes within the cluster. These nodes can start and stop at will as long as they have the ability to reach previously known members at the time of shutdown.

There are two types of nodes you can configure: disk and RAM nodes. Most commonly, you will use your nodes as disk nodes (preferred). Whereas RAM nodes are more of a special configuration used in

performance clusters.

RabbitMQ nodes and the CLI tools use an `erlang cookie` to determine whether or not they have permission to communicate. The cookie is a string of alphanumeric characters and can be as short or as long as you would like.

Note

The cookie value is a shared secret and should be protected and kept private.

The default location of the cookie on `*nix` environments is `/var/lib/rabbitmq/.erlang.cookie` or in `$HOME/.erlang.cookie`.

Tip

While troubleshooting, if you notice one node is refusing to join the cluster, it is definitely worth checking if the erlang cookie matches the other nodes. When the cookie is misconfigured (for example, not identical), RabbitMQ will log errors such as Connection attempt from disallowed node and Could not auto-cluster. See clustering for more information.

To form a RabbitMQ Cluster, you start by taking independent RabbitMQ brokers and re-configuring these nodes into a cluster configuration.

Using a 3 node example, you would be telling nodes 2 and 3 to join the cluster of the first node.

1. Login to the 2nd and 3rd node and stop the RabbitMQ application.
2. Join the cluster, then restart the application:

```
rabbit2$ rabbitmqctl stop_app
Stopping node rabbit@rabbit2 ...done.
rabbit2$ rabbitmqctl join_cluster rabbit@rabbit1
Clustering node rabbit@rabbit2 with [rabbit@rabbit1] ...done.
rabbit2$ rabbitmqctl start_app
Starting node rabbit@rabbit2 ...done.
```

Check the RabbitMQ cluster status

1. Run `rabbitmqctl cluster_status` from either node.

You will see `rabbit1` and `rabbit2` are both running as before.

The difference is that the cluster status section of the output, both nodes are now grouped together:

```
rabbit1$ rabbitmqctl cluster_status
Cluster status of node rabbit@rabbit1 ...
[{nodes,[{disc,[rabbit@rabbit1,rabbit@rabbit2]}]},
{running_nodes,[rabbit@rabbit2,rabbit@rabbit1]}]
...done.
```

To add the third RabbitMQ node to the cluster, repeat the above process by stopping the RabbitMQ application on the third node.

1. Join the cluster, and restart the application on the third node.
2. Execute `rabbitmq cluster_status` to see all 3 nodes:

```
rabbit1$ rabbitmqctl cluster_status
Cluster status of node rabbit@rabbit1 ...
[{nodes,[{disc,[rabbit@rabbit1,rabbit@rabbit2,rabbit@rabbit3]}]},
 {running_nodes,[rabbit@rabbit3,rabbit@rabbit2,rabbit@rabbit1]}]
...done.
```

Stop and restart a RabbitMQ cluster

To stop and start the cluster, keep in mind the order in which you shut the nodes down. The last node you stop, needs to be the first node you start. This node is the *master*.

If you start the nodes out of order, you could run into an issue where it thinks the current *master* should not be the master and drops the messages to ensure that no new messages are queued while the real master is down.

RabbitMQ and Mnesia

Mnesia is a distributed database that RabbitMQ uses to store information about users, exchanges, queues, and bindings. Messages, however are not stored in the database.

For more information about Mnesia, see the Mnesia overview.

To view the locations of important RabbitMQ files, see File Locations.

Repair a partitioned RabbitMQ cluster for a single-node

Invariably due to something in your environment, you are likely to lose a node in your cluster. In this scenario, multiple LXC containers on the same host are running RabbitMQ and are in a single RabbitMQ cluster.

If the host still shows as part of the cluster, but it is not running, execute:

```
# rabbitmqctl start_app
```

However, you may notice some issues with your application as clients may be trying to push messages to the un-responsive node. To remedy this, forget the node from the cluster by executing the following:

1. Ensure RabbitMQ is not running on the node:

```
# rabbitmqctl stop_app
```

2. On the RabbitMQ second node, execute:

```
# rabbitmqctl forget_cluster_node rabbit@rabbit1
```

By doing this, the cluster can continue to run effectively and you can repair the failing node.

Important

Watch out when you restart the node, it will still think it is part of the cluster and will require you to reset the node. After resetting, you should be able to rejoin it to other nodes as needed.

```
rabbit1$ rabbitmqctl start_app
Starting node rabbit@rabbit1 ...

Error: inconsistent_cluster: Node rabbit@rabbit1 thinks it's clustered
       with node rabbit@rabbit2, but rabbit@rabbit2 disagrees

rabbit1$ rabbitmqctl reset
Resetting node rabbit@rabbit1 ...done.
rabbit1$ rabbitmqctl start_app
Starting node rabbit@mcnulty ...
...done.
```

Repair a partitioned RabbitMQ cluster for a multi-node cluster

The same concepts apply to a multi-node cluster that exist in a single-node cluster. The only difference is that the various nodes will actually be running on different hosts. The key things to keep in mind when dealing with a multi-node cluster are:

- When the entire cluster is brought down, the last node to go down must be the first node to be brought online. If this does not happen, the nodes will wait 30 seconds for the last disc node to come back online, and fail afterwards.

  If the last node to go offline cannot be brought back up, it can be removed from the cluster using the **`forget_cluster_node`** command.

- If all cluster nodes stop in a simultaneous and uncontrolled manner, (for example, with a power cut) you can be left with a situation in which all nodes think that some other node stopped after them. In this case you can use the **`force_boot`** command on one node to make it bootable again.

Consult the rabbitmqctl manpage for more information.

Migrate between HA and Quorum queues

In the 2024.1 (Caracal) release OpenStack-Ansible switches to use RabbitMQ Quorum Queues by default, rather than the legacy High Availability classic queues. Migration to Quorum Queues can be performed at upgrade time, but may result in extended control plane downtime as this requires all OpenStack services to be restarted with their new configuration.

In order to speed up the migration, the following playbooks can be run to migrate either to or from Quorum Queues, whilst skipping package install and other configuration tasks. These tasks are available from the 2024.1 release onwards.

```
$ openstack-ansible openstack.osa.rabbitmq_server --tags rabbitmq-
↪config
$ openstack-ansible openstack.osa.setup_openstack --tags common-mq,
↪post-install
```

In order to take advantage of these steps, we suggest setting *oslomsg_rabbit_quorum_queues* to `false` before upgrading to 2024.1. Then, once you have upgraded, set *oslomsg_rabbit_quorum_queues* back to the default of `true` and run the playbooks above.

Running ad-hoc Ansible plays

Being familiar with running ad-hoc Ansible commands is helpful when operating your OpenStack-Ansible deployment. For a review, we can look at the structure of the following Ansible command:

```
$ ansible example_group -m shell -a 'hostname'
```

This command calls on Ansible to run the `example_group` using the `-m` shell module with the `-a` argument which is the hostname command. You can substitute example_group for any groups you may have defined. For example, if you had `compute_hosts` in one group and `infra_hosts` in another, supply either group name and run the command. You can also use the `*` wild card if you only know the first part of the group name, for instance if you know the group name starts with compute you would use `compute_h*`. The `-m` argument is for module.

Modules can be used to control system resources or handle the execution of system commands. For more information about modules, see Module Index and About Modules.

If you need to run a particular command against a subset of a group, you could use the limit flag `-l`. For example, if a `compute_hosts` group contained `compute1`, `compute2`, `compute3`, and `compute4`, and you only needed to execute a command on `compute1` and `compute4` you could limit the command as follows:

```
$ ansible example_group -m shell -a 'hostname' -l compute1,compute4
```

Note

Each host is comma-separated with no spaces.

Note

Run the ad-hoc Ansible commands from the `openstack-ansible/playbooks` directory.

For more information, see Inventory and Patterns.

Running the shell module

The two most common modules used are the `shell` and `copy` modules. The `shell` module takes the command name followed by a list of space delimited arguments. It is almost like the command module, but runs the command through a shell (`/bin/sh`) on the remote node.

For example, you could use the shell module to check the amount of disk space on a set of Compute hosts:

```
$ ansible compute_hosts -m shell -a 'df -h'
```

To check on the status of your Galera cluster:

```
$ ansible galera_container -m shell -a "mariadb \
-e 'show status like \"%wsrep_cluster_%\";'"
```

When a module is being used as an ad-hoc command, there are a few parameters that are not required. For example, for the `chdir` command, there is no need to **`chdir=/home/user ls`** when running Ansible

from the CLI:

```
$ ansible compute_hosts -m shell -a 'ls -la /home/user'
```

For more information, see shell - Execute commands in nodes.

Running the copy module

The copy module copies a file on a local machine to remote locations. To copy files from remote locations to the local machine you would use the fetch module. If you need variable interpolation in copied files, use the template module. For more information, see copy - Copies files to remote locations.

The following example shows how to move a file from your deployment host to the `/tmp` directory on a set of remote machines:

```
$ ansible remote_machines -m copy -a 'src=/root/FILE '\
'dest=/tmp/FILE'
```

The fetch module gathers files from remote machines and stores the files locally in a file tree, organized by the hostname.

Note

This module transfers log files that might not be present, so a missing remote file will not be an error unless `fail_on_missing` is set to `true`.

The following examples shows the `nova-compute.log` file being pulled from a single Compute host:

```
root@libertylab:/opt/rpc-openstack/openstack-ansible/playbooks# ansible␣
↪compute_hosts -m fetch -a 'src=/var/log/nova/nova-compute.log dest=/tmp'
aio1 | success >> {
    "changed": true,
    "checksum": "865211db6285dca06829eb2215ee6a897416fe02",
    "dest": "/tmp/aio1/var/log/nova/nova-compute.log",
    "md5sum": "dbd52b5fd65ea23cb255d2617e36729c",
    "remote_checksum": "865211db6285dca06829eb2215ee6a897416fe02",
    "remote_md5sum": null
}

root@libertylab:/opt/rpc-openstack/openstack-ansible/playbooks# ls -la /tmp/
↪aio1/var/log/nova/nova-compute.log
-rw-r--r-- 1 root root 2428624 Dec 15 01:23 /tmp/aio1/var/log/nova/nova-
↪compute.log
```

Using tags

Tags are similar to the limit flag for groups, except tags are used to only run specific tasks within a playbook. For more information on tags, see Tags.

Ansible forks

The default `MaxSessions` setting for the OpenSSH Daemon is 10. Each Ansible fork makes use of a session. By default, Ansible sets the number of forks to 5. However, you can increase the number of forks used in order to improve deployment performance in large environments.

Note that more than 10 forks will cause issues for any playbooks which use `delegate_to` or `local_action` in the tasks. It is recommended that the number of forks are not raised when executing against the control plane, as this is where delegation is most often used.

When increasing the number of Ansible forks in, particularly beyond 10, SSH connection issues can arise due to the default sshd setting MaxStartups 10:30:100. This setting limits the number of simultaneous unauthenticated SSH connections to 10, after which new connection attempts start getting dropped probabilistically with a 30% chance initially, increasing linearly up to 100% as the number of connections approaches 100.

The number of forks used may be changed on a permanent basis by including the appropriate change to the `ANSIBLE_FORKS` in your `.bashrc` file. Alternatively it can be changed for a particular playbook execution by using the `--forks` CLI parameter. For example, the following executes the nova playbook against the control plane with 10 forks, then against the compute nodes with 50 forks.

```
# openstack-ansible --forks 10 os-nova-install.yml --limit compute_containers
# openstack-ansible --forks 50 os-nova-install.yml --limit compute_hosts
```

For more information about forks, please see the following references:

- Ansible forks entry for ansible.cfg

Container management

With Ansible, the OpenStack installation process is entirely automated using playbooks written in YAML. After installation, the settings configured by the playbooks can be changed and modified. Services and containers can shift to accommodate certain environment requirements. Scaling services are achieved by adjusting services within containers, or adding new deployment groups. It is also possible to destroy containers, if needed, after changes and modifications are complete.

Scale individual services

Individual OpenStack services, and other open source project services, run within containers. It is possible to scale out these services by modifying the `/etc/openstack_deploy/openstack_user_config.yml` file.

1. Navigate into the `/etc/openstack_deploy/openstack_user_config.yml` file.
2. Access the deployment groups section of the configuration file. Underneath the deployment group name, add an affinity value line to container scales OpenStack services:

```
infra_hosts:
  infra1:
    ip: 10.10.236.100
    # Rabbitmq
    affinity:
      galera_container: 1
      rabbit_mq_container: 2
```

In this example, `galera_container` has a container value of one. In practice, any containers that do not need adjustment can remain at the default value of one, and should not be adjusted above or below the value of one.

The affinity value for each container is set at one by default. Adjust the affinity value to zero for situations where the OpenStack services housed within a specific container will not be needed when scaling out other required services.

3. Update the container number listed under the `affinity` configuration to the desired number. The above example has `galera_container` set at one and `rabbit_mq_container` at two, which scales RabbitMQ services, but leaves Galera services fixed.

4. Run the appropriate playbook commands after changing the configuration to create the new containers, and install the appropriate services.

   For example, run the **openstack-ansible lxc-containers-create.yml rabbitmq-install.yml** commands from the `openstack-ansible/playbooks` repository to complete the scaling process described in the example above:

```
$ cd openstack-ansible/playbooks
$ openstack-ansible lxc-containers-create.yml rabbitmq-install.yml
```

Destroy and recreate containers

Resolving some issues may require destroying a container, and rebuilding that container from the beginning. It is possible to destroy and re-create a container with the `lxc-containers-destroy.yml` and `lxc-containers-create.yml` commands. These Ansible scripts reside in the `openstack-ansible/playbooks` repository.

1. Navigate to the `openstack-ansible` directory.
2. Run the **openstack-ansible lxc-containers-destroy.yml** commands, specifying the target containers and the container to be destroyed.

```
$ openstack-ansible lxc-containers-destroy.yml --limit "CONTAINER_NAME"
$ openstack-ansible lxc-containers-create.yml --limit "CONTAINER_NAME"
```

3. Replace `CONTAINER_NAME` with the target container.

Logging Services in OpenStack-Ansible

Since the Train release, OpenStack-Ansible services have been configured to save logs in `systemd-journald` instead of traditional log files. Journald logs from containers are passed through to the physical host, so you can read and manipulate all service logs directly from the metal hosts using tools like `journalctl`.

`systemd-journald` integrates well with a wide range of log collectors and forwarders, including `rsyslog`. However, while `rsyslog` stores data as plain text (making it harder to index and search efficiently), journald uses a structured format that allows logs to be queried and processed much more efficiently by modern log analysis tools.

Log Locations

All container journals are accessible on the host under:

```
/var/log/journal/
```

This allows you to access and filter all service logs directly on the host using tools such as journalctl. This also allows log collectors running on the host to more seamlessly pick up and process journald log streams coming from all service containers.

Note

Due to the adoption of `systemd-journald` as the primary logging backend, the traditional mapping of `/openstack/log/` to `/var/log/$SERVICE` inside the container is no longer present. Logs should be accessed directly through journald tools such as `journalctl` or by examining the `/var/log/journal/` directories on the host.

Configuring journald

The `openstack_hosts` role allows control over the behavior of `systemd-journald` on the host. There are following variable to configure journald settings:

1. **Persistent journal storage**

   By default, systemd journals are kept in memory and discarded after a reboot. OpenStack-Ansible sets the variable `openstack_host_keep_journals: true` by default, which persists journals across reboots. You can explicitly configure it in your `user_variables.yml` if needed:

   ```
   openstack_host_keep_journals: true
   ```

   This ensures that logs remain available for troubleshooting even after host restarts.

2. **Custom journald configuration**

   You can supply arbitrary journald configuration options by defining a mapping in `openstack_hosts_journald_config` in your `user_variables.yml`. For example:

   ```
   openstack_hosts_journald_config:
     SystemMaxUse: 20G
     MaxRetentionSec: 7day
   ```

   This example limits journalds maximum disk usage to 20 GB and retains logs for 7 days.

After adjusting any journald-related variables, you can apply the changes by re-running the `openstack_hosts_setup` role:

```
openstack-ansible openstack.osa.openstack_hosts_setup
```

You can also check out our ELK role from OPS repository for a ready-to-use ELK stack deployment and metrics collection.

Firewalls

OpenStack-Ansible does not configure firewalls for its infrastructure. It is up to the deployer to define the perimeter and its firewall configuration.

By default, OpenStack-Ansible relies on Ansible SSH connections, and needs the TCP port 22 to be opened on all hosts internally.

For more information on generic OpenStack firewall configuration, see the Firewalls and default ports

In each of the roles respective documentatione you can find the default variables for the ports used within the scope of the role. Reviewing the documentation allow you to find the variable names if you want to use a different port.

Note

OpenStack-Ansible group vars conveniently expose the vars outside of the role scope in case you are relying on the OpenStack-Ansible groups to configure your firewall.

Finding ports for your external load balancer

As explained in the previous section, you can find (in each roles documentation) the default variables used for the public interface endpoint ports.

For example, the os_glance documentation lists the variable `glance_service_publicuri`. This contains the port used for the reaching the service externally. In this example, it is equal to `glance_service_port`, whose value is 9292.

As a hint, you could find the list of all public URI defaults by executing the following:

```
cd /etc/ansible/roles
grep -R -e publicuri -e port *
```

Note

HAProxy can be configured with OpenStack-Ansible. The automatically generated `/etc/haproxy/haproxy.cfg` file have enough information on the ports to open for your environment.

Prune Inventory Backup Archive

The inventory backup archive will require maintenance over a long enough period of time.

Bulk pruning

It is possible to do mass pruning of the inventory backup. The following example will prune all but the last 15 inventories from the running archive.

```
ARCHIVE="/etc/openstack_deploy/backup_openstack_inventory.tar"
tar -tvf ${ARCHIVE} | \
  head -n -15 | awk '{print $6}' | \
  xargs -n 1 tar -vf ${ARCHIVE} --delete
```

Selective Pruning

To prune the inventory archive selectively, first identify the files you wish to remove by listing them out.

```
tar -tvf /etc/openstack_deploy/backup_openstack_inventory.tar

-rw-r--r-- root/root    110096 2018-05-03 10:11 openstack_inventory.json-
↪20180503_151147.json
-rw-r--r-- root/root    110090 2018-05-03 10:11 openstack_inventory.json-
↪20180503_151205.json
-rw-r--r-- root/root    110098 2018-05-03 10:12 openstack_inventory.json-
↪20180503_151217.json
```

Now delete the targeted inventory archive.

```
tar -vf /etc/openstack_deploy/backup_openstack_inventory.tar --delete␣
↪openstack_inventory.json-20180503_151205.json
```

Scaling your environment

This chapter is about scaling your environment using OpenStack-Ansible.

Scaling MariaDB and RabbitMQ

Contents

OpenStack is a cloud computing platform that is designed to be highly scalable. However, even though OpenStack is designed to be scalable, there are a few potential bottlenecks that can occur in large deployments. These bottlenecks typically involve the performance and throughput of RabbitMQ and MariaDB clusters.

RabbitMQ is a message broker that is used to decouple different components of OpenStack. MariaDB is

a database that is used to store data for OpenStack. If these two components are not performing well, it can have a negative impact on the performance of the entire OpenStack deployment.

There are a number of different methodologies that can be used to improve the performance of RabbitMQ and MariaDB clusters. These methodologies include scaling up the clusters, using a different message broker or database, or optimizing the configuration of the clusters.

In this series of articles, will be discussed the potential bottlenecks that can occur in large OpenStack deployments and ways to scale up deployments to improve the performance of RabbitMQ and MariaDB clusters.

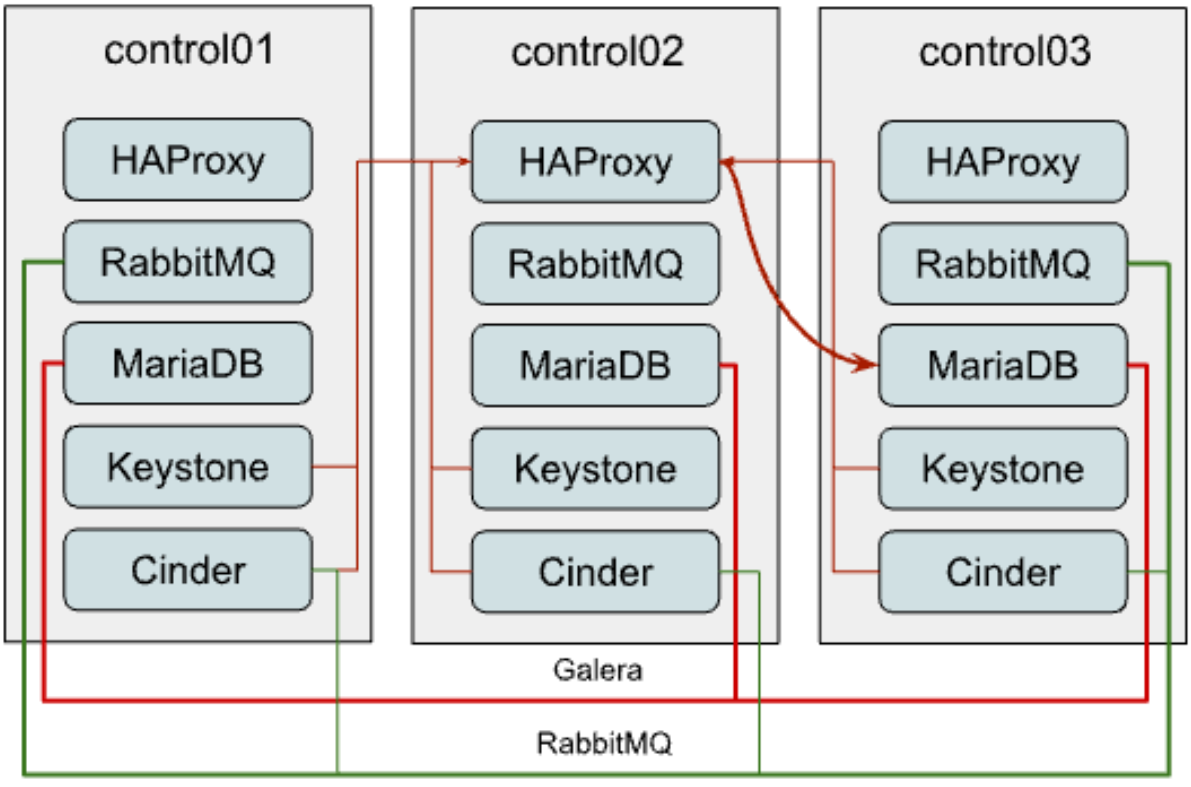
Note

Examples provided in this documentation were made on OpenStack 2023.1 (Antelope). It is possible to achieve the same flows in earlier releases, but some extra steps or slightly different configurations might be required.

Most Common Deployment

Before talking about ways on how to improve things, lets quickly describe starting point, to understand what were dealing with at the starting point.

The most common OpenStack-Ansible deployment design is three control nodes, each one is running all OpenStack API services along with supporting infrastructure, like MariaDB and RabbitMQ clusters. This is a good starting point for small to medium-sized deployments. However, as the deployment grows, you may start to experience performance problems. Typically communication between services and MariaDB/RabbitMQ looks like this:



MariaDB

As you might see on the diagram, all connections to MariaDB come through the HAProxy which has Internal Virtual IP (VIP). OpenStack-Ansible does configure the Galera cluster for MariaDB, which is a multi-master replication system. Although you can issue any request to any member of the cluster, all write requests will be passed to the current primary instance creating more internal traffic and raising the amount of work each instance should do. So it is recommended to pass write requests only to the primary instance.

However HAProxy is not capable of balancing MariaDB queries at an application level (L7 of OSI model), to separate read and write requests, so we have to balance TCP streams (L3) and pass all traffic without any separation to the current primary node in the Galera cluster, which creates a potential bottleneck.

RabbitMQ

RabbitMQ is clustered differently. We supply IP addresses of all cluster members to clients and its up to the client to decide which backend it will use for interaction. Only RabbitMQ management UI is balanced through HAProxy, so the connection of clients to queues does not depend on HAProxy in any way.

Though usage of HA queues and even quorum queues makes all messages and queues to be mirrored to all or several cluster members. While quorum queues show way better performance, they still suffer from clustering traffic which still becomes a problem at a certain scale.

Option 1: Independent clusters per service

With this approach, you might provide the most loaded services, like Nova or Neutron, their standalone MariaDB and RabbitMQ clusters. These new clusters might reside on a separate hardware.

In the example below we assume that only Neutron is being reconfigured to use the new standalone cluster, while other services remain sharing the already existing one. So Neutron connectivity will look like this:

As you might have noticed, we still consume the same HAProxy instance for MariaDB balancing to the new infra cluster.

Next, we will describe how to configure such a stack and execute the service transition to this new layout.

Setup of new MariaDB and RabbitMQ clusters

To configure such a layout and migrate Neutron using it with OpenStack-Ansible you need to follow these steps:

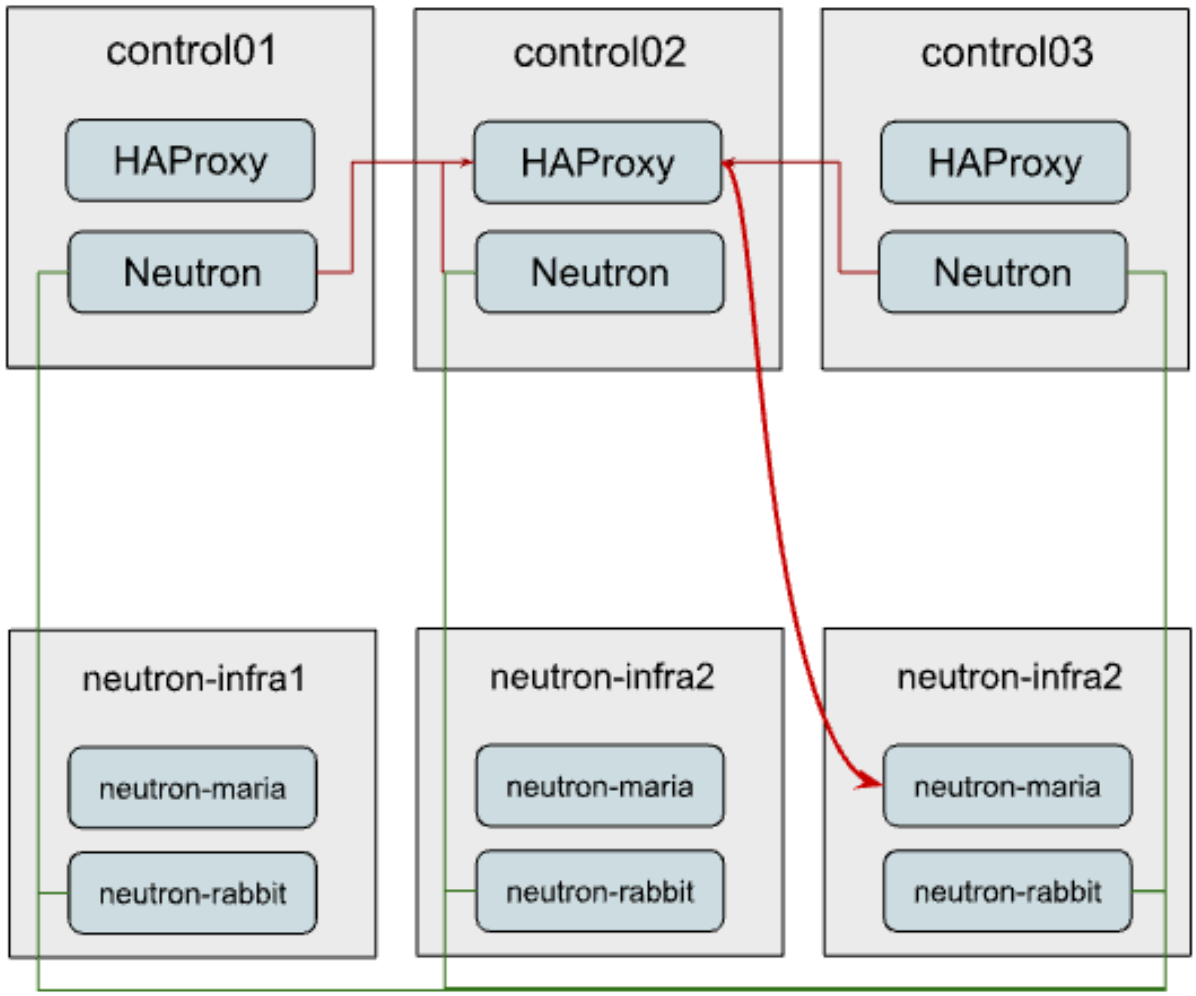
Note

You can reference the following documentation for a deeper understanding of how env.d and conf.d files should be constructed: *Understanding the inventory*

- Define new groups for RabbitMQ and MariaDB. For that, you can create files with the following content: `/etc/openstack_deploy/env.d/galera-neutron.yml`:

```
# env.d file are more clear if you read them bottom-up
# At component skeleton you map component to ansible groups
component_skel:
  # Component itself is an ansible group as well
  neutron_galera:
```

(continues on next page)


control01
HAProxy
Neutron
control02
HAProxy
Neutron
control03
HAProxy
Neutron
neutron-infra1
neutron-maria
neutron-rabbit
neutron-infra2
neutron-maria
neutron-rabbit
neutron-infra2
neutron-maria
neutron-rabbit

(continued from previous page)

```
  # You tell in which ansible groups component will appear
  belongs_to:
    - neutron_galera_all
    - galera_all

# At container skeleton you link components to physical layer
container_skel:
  neutron_galera_container:
    # Here you define on which physical hosts container will reside
    belongs_to:
      - neutron-database_containers
    # Here you define which components will reside on container
   contains:
     - neutron_galera

# At physical skeleton level you map containers to hosts
physical_skel:
  # Here you tell to which global group containers will be added
  # from the host in question.
  # Please note, that <name>_hosts and <name>_containers are
  # interconnected, and <name> can not contain underscores.
  neutron-database_containers:
    belongs_to:
      - all_containers

# You define `<name>_hosts` in your openstack_user_config or conf.d
# files to tell on which physical hosts containers should be spawned
neutron-database_hosts:
  belongs_to:
    - hosts
```

- `/etc/openstack_deploy/env.d/rabbit-neutron.yml`:

```
# On the component level we are creating group `neutron_rabbitmq`
# that is also part of `rabbitmq_all` and `neutron_rabbitmq_all`
component_skel:
  neutron_rabbitmq:
    belongs_to:
      - rabbitmq_all
      - neutron_rabbitmq_all

# On the container level we tell to create neutron_rabbitmq on
# neutron-mq_hosts
container_skel:
  neutron_rabbit_mq_container:
    belongs_to:
      - neutron-mq_containers
    contains:
      - neutron_rabbitmq
```

(continues on next page)

(continued from previous page)

```
# We define the physical level as a base level which can be consumed
# by container and component skeleton.
physical_skel:
  neutron-mq_containers:
    belongs_to:
      - all_containers
  neutron-mq_hosts:
    belongs_to:
      - hosts
```

Map your new neutron-infra hosts to these new groups. To add to your `openstack_user_config.yml` the following content:

```
neutron-mq_hosts: &neutron_infra
  neutron-infra1:
    ip: 172.29.236.200
  neutron-infra2:
    ip: 172.29.236.201
  neutron-infra3:
    ip: 172.29.236.202
neutron-database_hosts: *neutron_infra
```

- Define some specific configurations for newly created groups and balance them:
- MariaDB
  - In file `/etc/openstack_deploy/group_vars/neutron_galera.yml`:

```
galera_cluster_members: "{{ groups['neutron_galera'] }}"
galera_cluster_name: neutron_galera_cluster
galera_root_password: mysecret
```

In file `/etc/openstack_deploy/group_vars/galera.yml`:

```
galera_cluster_members: "{{ groups['galera'] }}"
```

- Move *galera_root_password* from `/etc/openstack_deploy/user_secrets.yml` to `/etc/openstack_deploy/group_vars/galera.yml`
- RabbitMQ
  - In file `/etc/openstack_deploy/group_vars/neutron_rabbitmq.yml`:

```
rabbitmq_host_group: neutron_rabbitmq
rabbitmq_cluster_name: neutron_cluster
```

  - In file `/etc/openstack_deploy/group_vars/rabbitmq.yml`

```
rabbitmq_host_group: rabbitmq
```

- HAProxy
  - In `/etc/openstack_deploy/user_variables.yml` define extra service for MariaDB:

```
haproxy_extra_services:

  - haproxy_service_name: galera_neutron
    haproxy_backend_nodes: "{{ (groups['neutron_galera'] |␣
↪default([]))[:1] }}"
    haproxy_backup_nodes: "{{ (groups['neutron_galera'] |␣
↪default([]))[1:] }}"
    haproxy_bind: "{{ [haproxy_bind_internal_lb_vip_address |␣
↪default(internal_lb_vip_address)] }}"
    haproxy_port: 3307
    haproxy_backend_port: 3306
    haproxy_check_port: 9200
    haproxy_balance_type: tcp
    haproxy_stick_table_enabled: False
    haproxy_timeout_client: 5000s
    haproxy_timeout_server: 5000s
    haproxy_backend_options:
      - "httpchk HEAD / HTTP/1.0\\r\\nUser-agent:\\ osa-haproxy-
↪healthcheck"
    haproxy_backend_server_options:
      - "send-proxy-v2"
    haproxy_allowlist_networks: "{{ haproxy_galera_allowlist_networks }}"
    haproxy_service_enabled: "{{ groups['neutron_galera'] is defined and␣
↪groups['neutron_galera'] | length > 0 }}"

haproxy_galera_service_overrides:
  haproxy_backend_nodes: "{{ groups['galera'][:1] }}"
  haproxy_backup_nodes: "{{ groups['galera'][1:] }}"
```

- Prepare new infra hosts and create containers on them. For that, run the command:

```
# openstack-ansible playbooks/setup-hosts.yml --limit neutron-mq_hosts,
↪neutron-database_hosts,neutron_rabbitmq,neutron_galera
```

- Deploy clusters:
  - MariaDB:

    ```
    openstack-ansible playbooks/galera-install.yml --limit neutron_galera
    ```

  - RabbitMQ:

    ```
    openstack-ansible playbooks/rabbitmq-install.yml --limit neutron_
    ↪rabbitmq
    ```

Migrating the service to use new clusters

While its relatively easy to start using the new RabbitMQ cluster for the service, migration of the database is slightly tricky and will include some downtime.

First, we need to tell Neutron that from now on, the MariaDB database for the service is listening on a different port. So you should add the following override to your `user_variables.yml`:

```
neutron_galera_port: 3307
```

Now lets prepare the destination database: create the database itself along with required users and provide them permissions to interact with the database. For that, we will run the neutron role with a common-db tag and limit execution to the neutron_server group only. You can use the following command for that:

```
# openstack-ansible playbooks/os-neutron-install.yml --limit neutron_server --
↪tags common-db
```

Once we have a database prepared, we need to disable HAProxy backends that proxy traffic to the API of the service in order to prevent any user or service actions with it.

For that, we use a small custom playbook. Lets name it `haproxy_backends.yml`:

```
- hosts: haproxy_all
  tasks:
    - name: Manage backends
      community.general.haproxy:
        socket: /run/haproxy.stat
        backend: "{{ backend_group }}-back"
        drain: "{{ haproxy_drain | default(False) }}"
        host: "{{ item }}"
        state: "{{ haproxy_state | default('disabled') }}"
        shutdown_sessions: "{{ haproxy_shutdown_sessions | default(False) |␣
↪bool }}"
        wait: "{{ haproxy_wait | default(False) | bool }}"
        wait_interval: "{{ haproxy_wait_interval | default(5) }}"
        wait_retries: "{{ haproxy_wait_retries | default(24) }}"
      with_items: "{{ groups[backend_group] }}"
```

We run it as follows:

```
# openstack-ansible haproxy_backends.yml -e backend_group=neutron_server
```

No, we can stop the API service for Neutron:

```
# ansible -m service -a "state=stopped name=neutron-server" neutron_server
```

And run a backup/restore of the MariaDB database for the service. For this purpose, we will use another small playbook, that we name as `mysql_backup_restore.yml` with the following content:

```
- hosts: "{{ groups['galera'][0] }}"
  vars:
    _db: "{{ neutron_galera_database | default('neutron') }}"
  tasks:
    - name: Dump the db
      shell: "mysqldump --single-transaction {{ _db }} > /tmp/{{ _db }}"
    - name: Fetch the backup
      fetch:
        src: "/tmp/{{ _db }}"
        dest: "/tmp/db-backup/"
        flat: yes
```

(continues on next page)

(continued from previous page)

```
- hosts: "{{ groups['neutron_galera'][0] }}"
  vars:
    _db: "{{ neutron_galera_database | default('neutron') }}"
  tasks:
    - name: Copy backups to destination
      copy:
        src: "/tmp/db-backup/"
        dest: "/tmp/db-backup/"
    - name: Restore the DB backup
      shell: "mysql {{ _db }} < /tmp/db-backup/{{ _db }}"
```

Now lets run the playbook weve just created:

```
# openstack-ansible mysql_backup_restore.yml
```

Note

The playbook above is not idempotent as it will override database content on the destination hosts.

Once the database content is in place, we can now re-configure the service using the playbook.

It will not only tell Neutron to use the new database but also will switch it to using the new RabbitMQ cluster as well and re-enable the service in HAProxy.

For that to happen we should run the following command:

```
# openstack-ansible playbooks/os-neutron-install.yml --tags neutron-config,
↪common-mq
```

After the playbook has finished, neutron services will be started and configured to use new clusters.

Option 2: Dedicated hardware for clusters

This option will describe how to move current MariaDB and RabbitMQ clusters to standalone nodes. This approach can be used to offload control-planes and provide dedicated resources for clusters.

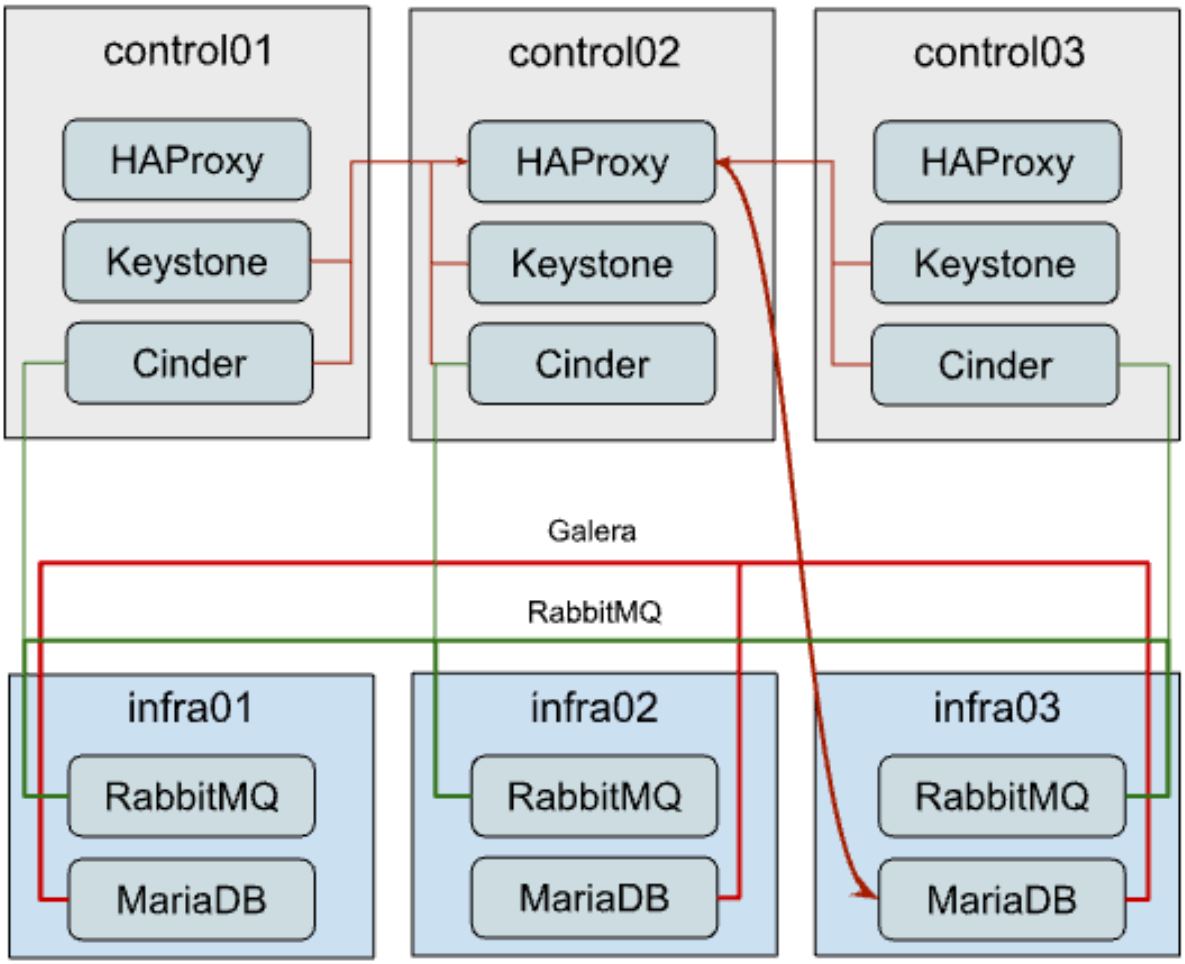
While its quite straightforward to establish the architecture above from the very beginning of the deployment, flawless migration of the existing deployment to such a setup is more tricky, as you need to migrate running clusters to the new hardware. Since we will be performing moves one-by-one, to preserve at least two active cluster members, the steps below should be repeated for the other two members.

Migrating MariaDB to the new hardware

The first thing to do is to list current members of the MariaDB cluster. For that, you can issue the following ad-hoc command:

```
# cd /opt/openstack-ansible/
# ansible -m debug -a "var=groups['galera']" localhost
localhost | SUCCESS => {
```

(continues on next page)


control01
HAProxy
Keystone
Cinder
control02
HAProxy
Keystone
Cinder
control03
HAProxy
Keystone
Cinder
Galera
RabbitMQ
infra01
RabbitMQ
MariaDB
infra02
RabbitMQ
MariaDB
infra03
RabbitMQ
MariaDB

(continued from previous page)

```
    "groups['galera']": [
        "control01_galera_container-68e1fc47",
        "control02_galera_container-59576533",
        "control03_galera_container-f7d1b72b"
    ]
}
```

Unless overridden, the first host in the group is considered as a bootstrap one. This bootstrap host should be migrated last to avoid unnecessary failovers, so it is recommended to start the migration of hosts to the new hardware from the last one to the first one in the output.

Once weve figured out the execution order, its time for a step-by-step guide.

- Remove the last container in the group using the following playbook:

```
# openstack-ansible playbooks/lxc-containers-destroy.yml --limit␣
↪control03_galera_container-f7d1b72b
```

- Clean up the removed container from the inventory:

```
# ./scripts/inventory-manage.py -r control03_galera_container-f7d1b72b
```

Re-configure `openstack_user_config` to create a new container.

Assuming, you currently have a config like the one below in your `openstack_user_config.yml`:

```
_control_hosts: &control_hosts
  control01:
    ip: 172.29.236.11
  control02:
    ip: 172.29.236.12
  control03:
    ip: 172.29.236.13

shared-infra_hosts: *control_hosts
```

Convert it to something like this:

```
_control_hosts: &control_hosts
  control01:
    ip: 172.29.236.11
  control02:
    ip: 172.29.236.12
  control03:
    ip: 172.29.236.13

memcaching_hosts: *control_hosts
mq_hosts: *control_hosts
operator_hosts: *control_hosts

database_hosts:
  control01:
```

(continues on next page)

(continued from previous page)

```
    ip: 172.29.236.11
  control02:
    ip: 172.29.236.12
  infra03:
    ip: 172.29.236.23
```

In the example above we de-couple each service that is part of the *shared-infra_hosts* and define them separately, along with providing MariaDB its new destination host.

- Create the container on the new infra node:

```
# openstack-ansible playbooks/lxc-containers-create.yml --limit infra03,
↪galera
```

> **Note**
>
> New infra hosts should be prepared before this step (i.e., by running `setup-hosts.yml` playbook against them).

- Install MariaDB to this new container and add it to the cluster:

```
# openstack-ansible playbooks/galera-install.yml
```

- Once the playbook is finished, you can ensure that the cluster is in the **Synced** state and has proper cluster_size with the following ad-hoc:

```
# ansible -m command -a "mysql -e \"SHOW STATUS WHERE Variable_name IN (
↪'wsrep_local_state_comment', 'wsrep_cluster_size', 'wsrep_incoming_
↪addresses')\"" neutron_galera
```

- If the cluster is healthy, repeat steps 1-6 for the rest instances, including the bootstrap one.

Migrating RabbitMQ to the new hardware

The process of RabbitMQ migration will be pretty much the same as MariaDB with one exception we need to preserve the same IP addresses for containers when moving them to the new hardware. Otherwise, we would need to re-configure all services (like cinder, nova, neutron, etc.) that rely on RabbitMQ as well, as contrary to MariaDB which is balanced through HAProxy, its a client who decides to which RabbitMQ backend it will connect.

Thus, we also dont care about the order of migration.

Since we need to preserve an IP address, lets collect this data before taking any actions against the current setup:

```
# ./scripts/inventory-manage.py -l | grep rabbitmq
| control01_rabbit_mq_container-a3a802ac  | None     | rabbitmq   | control01␣
↪         | None           | 172.29.239.49  | None                |
| control02_rabbit_mq_container-51f6cf7c  | None     | rabbitmq   | control02␣
↪         | None           | 172.29.236.82  | None                |
```

(continues on next page)

(continued from previous page)

```
| control03_rabbit_mq_container-b30645d9  | None     | rabbitmq    | control03␣
↪          | None           | 172.29.238.23  | None                  |
```

Before dropping the RabbitMQ container, its worth transitioning the RabbitMQ instance to the Maintenance mode, so it could offload its responsibilities to other cluster members and close connections to clients properly. You can use the following ad-hoc for that:

```
root@deploy:/opt/openstack-ansible# ansible -m command -a "rabbitmq-upgrade␣
↪drain" control01_rabbit_mq_container-a3a802ac

control01_rabbit_mq_container-a3a802ac | CHANGED | rc=0 >>
Will put node rabbit@control01-rabbit-mq-container-a3a802ac into maintenance␣
↪mode. The node will no longer serve any client traffic!
```

- Now we can proceed with container removal:

```
# openstack-ansible playbooks/lxc-containers-destroy.yml --limit␣
↪control01_rabbit_mq_container-a3a802ac
```

- And remove it from the inventory:

```
# ./scripts/inventory-manage.py -r control01_rabbit_mq_container-a3a802ac
```

Now you need to re-configure `openstack_user_config` similar to how it was done for MariaDB. The resulting record at this stage for RabbitMQ should look like this:

```
mq_hosts:
  infra01:
    ip: 172.29.236.21
  control02:
    ip: 172.29.236.12
  control03:
    ip: 172.29.236.13
```

Note

Ensure that you dont have more generic shared-infra_hosts defined.

Now we need to manually re-generate the inventory and ensure that a new record was mapped to our infra01:

```
# ./inventory/dynamic_inventory.py

...

# ./scripts/inventory-manage.py -l | grep rabbitmq

| control02_rabbit_mq_container-51f6cf7c  | None     | rabbitmq    | control02␣
↪          | None           | 172.29.236.82  | None                  |
```

(continues on next page)

(continued from previous page)

```
| control03_rabbit_mq_container-b30645d9  | None     | rabbitmq   | control03␣
↪         | None           | 172.29.238.23  | None                |
| infra01_rabbit_mq_container-10ec4732  | None     | rabbitmq   | infra01    ␣
↪     | None           | 172.29.238.248 | None                |
```

As you might see from the output above, a record for the new container has been generated and assigned correctly to the infra01 host. Though this container has a new IP address, we need to preserve it. So we manually replaced the new IP with the old one in the inventory file and ensured its the proper one now:

```
# sed -i 's/172.29.238.248/172.29.239.49/g' /etc/openstack_deploy/openstack_
↪inventory.json
#./scripts/inventory-manage.py -l | grep rabbitmq
| control02_rabbit_mq_container-51f6cf7c  | None     | rabbitmq   | control02␣
↪         | None           | 172.29.236.82  | None                |
| control03_rabbit_mq_container-b30645d9  | None     | rabbitmq   | control03␣
↪         | None           | 172.29.238.23  | None                |
| infra01_rabbit_mq_container-10ec4732  | None     | rabbitmq   | infra01    ␣
↪     | None           | 172.29.239.49 | None                |
```

- Now you can proceed with container creation:

```
# openstack-ansible playbooks/lxc-containers-create.yml --limit infra01,
↪rabbitmq
```

- And install RabbitMQ to the new container and ensure its part of the cluster:

```
# openstack-ansible playbooks/rabbitmq-install.yml
```

- Once the cluster is re-established, its worth to clean-up cluster status with regards to the old container name still being considered as Disk Node, since the container name has changed:

```
# ansible -m command -a "rabbitmqctl forget_cluster_node rabbit@control01-
↪rabbit-mq-container-a3a802ac" rabbitmq[0]
```

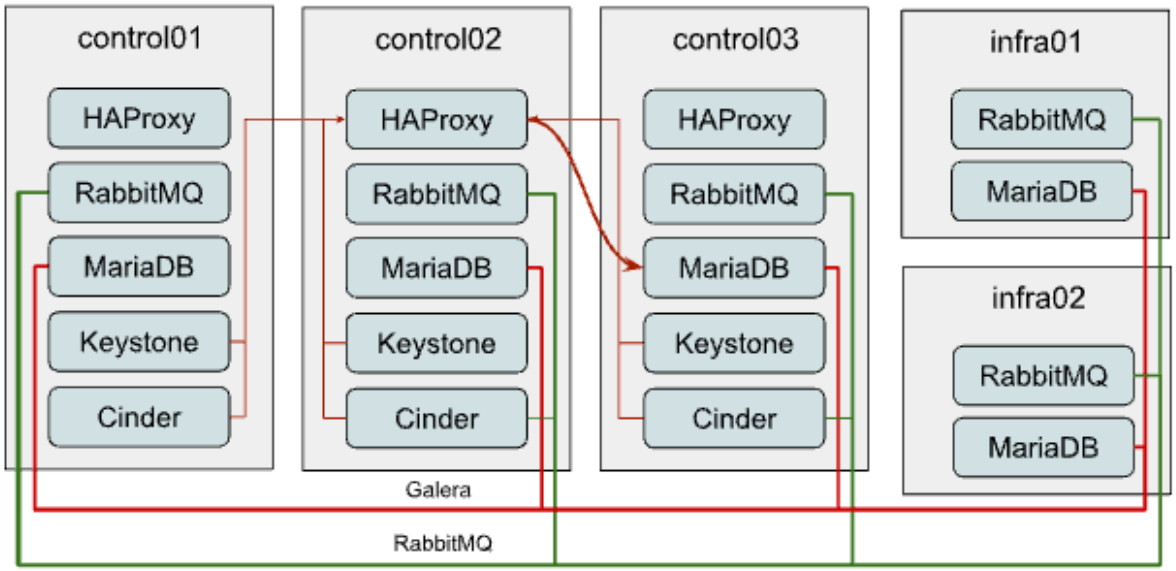
> **Note**
>
> You can take the cluster node name to remove from the output at step two.

- Repeat the steps above for the rest of the instances.

Option 3: Growing Clusters Horizontally

This option is by far the least popular despite being very straightforward, as it has a pretty narrowed use case when it makes sense to scale this way.

Though, to preserve quorum you should always have an odd number of cluster members or be prepared to provide extra configuration if using an even number of members.



Adding new members to the MariaDB Galera cluster

Horizontal scaling of the MariaDB cluster makes sense only when youre using an L7 balancer which can work properly with Galera clusters (like ProxySQL or MaxScale) instead of default HAProxy and the weak point of the current cluster is read performance rather than writes.

Extending the cluster is quite trivial. For that, you need to:

1. Add another destination host in `openstack_user_config` for database_hosts:

```
database_hosts:
  control01:
    ip: 172.29.236.11
  control02:
    ip: 172.29.236.12
  control03:
    ip: 172.29.236.13
  infra01:
    ip: 172.29.236.21
  infra02:
    ip: 172.29.236.22
```

2. Create new containers on the destination host:

```
# openstack-ansible playbooks/lxc-containers-create.yml --limit infra01,
↪infra02,galera
```

3. Deploy MariaDB there and add it to the cluster:

```
# openstack-ansible playbooks/galera-install.yml
```

4. Ensure the cluster is healthy with the following ad-hoc:

```
# ansible -m command -a "mysql -e \"SHOW STATUS WHERE Variable_name IN (
```

(continues on next page)

(continued from previous page)

```
↪'wsrep_local_state_comment', 'wsrep_cluster_size', 'wsrep_incoming_
↪addresses')\"" neutron_galera
```

Adding new members to the RabbitMQ cluster

Growing the RabbitMQ cluster vertically makes sense mostly when you dont have HA queues or Quorum queues enabled.

To add more members to the RabbitMQ cluster execute the following steps:

1. Add another destination host in `openstack_user_config` for mq_hosts:

```
mq_hosts:
  control01:
    ip: 172.29.236.11
  control02:
    ip: 172.29.236.12
  control03:
    ip: 172.29.236.13
  infra01:
    ip: 172.29.236.21
  infra02:
    ip: 172.29.236.22
```

2. Create new containers on the destination host:

```
# openstack-ansible playbooks/lxc-containers-create.yml --limit infra01,
↪infra02,rabbitmq
```

3. Deploy RabbitMQ on the new host and enroll it to the cluster:

```
# openstack-ansible playbooks/rabbitmq-install.yml
```

4. Once a new RabbitMQ container is deployed, you need to make all services aware of its existence by re-configuring them. For that, you can either run individual service playbooks, like this:

```
# openstack-ansible playbooks/os-<service>-install.yml tags <service>-
↪config
```

Where <service> is a service name, like neutron, nova, cinder, etc. Another way around would be to fire up setup-openstack.yml but it will take quite some time to execute.

Conclusion

As you might see, OpenStack-Ansible is flexible enough to let you scale a deployment in many different ways.

But which one is right for you? Well, it all depends on the situation you find yourself in.

In case your deployment has grown to a point where RabbitMQ/MariaDB clusters cant simply deal with the load these clusters create regardless of the hardware beneath them you should use option one (*Option 1: Independent clusters per service*) and make independent clusters per service.

This option can be also recommended to improve deployment resilience in case of cluster failure this will affect just one service rather than each and everyone in a common deployment use case. Another quite popular variation of this option can be having just standalone MariaDB/RabbitMQ instances per service, without any clusterization. The benefit of such a setup is very fast recovery, especially when talking about RabbitMQ.

In case you are the owner of quite modest hardware specs for controllers, you might pay more attention to option two (*Option 1: Independent clusters per service*). This way you can offload your controllers by moving heavy applications, like MariaDB/RabbitMQ, to some other hardware that can also have relatively modest specs.

Option three (*Option 3: Growing Clusters Horizontally*) can be used if your deployment meets the requirements that were written above (ie. not using HA queues or using ProxySQL for balancing) and usually should be considered when youve outgrown option one as well.

Add a new infrastructure host

While three infrastructure hosts are recommended, if further hosts are needed in an environment, it is possible to create additional nodes.

Warning

Make sure you back up your current OpenStack environment before adding any new nodes. See *Backup and restore your cloud* for more information.

1. Add the node to the `infra_hosts` stanza of the `/etc/openstack_deploy/openstack_user_config.yml`:

```
infra_hosts:
[...]
  infra<node-ID>:
    ip: 10.17.136.32
```

2. Change to playbook folder on the deployment host:

```
# cd /opt/openstack-ansible
```

3. To prepare new hosts and deploy containers on them run `setup-hosts.yml`: playbook with the `limit` argument.

```
# openstack-ansible openstack.osa.setup_hosts --limit localhost,infra
↪<node-ID>,infra<node-ID>-host_containers
```

4. In case youre relying on `/etc/hosts` content, you should also update it for all hosts:

```
# openstack-ansible openstack.osa.openstack_hosts_setup -e openstack_
↪hosts_group=all --tags openstack_hosts-file
```

5. Next we need to expand Galera/RabbitMQ clusters, which is done during `setup-infrastructure.yml`. So we will run this playbook without limits:

Warning

Make sure that containers from new infra host *does not* appear in inventory as first one for groups `galera_all`, `rabbitmq_all` and `repo_all`. You can verify that with ad-hoc commands:

```
# ansible -m debug -a "var=groups['galera_all'][0]" localhost
# ansible -m debug -a "var=groups['rabbitmq_all'][0]" localhost
# ansible -m debug -a "var=groups['repo_all'][0]" localhost
```

```
# openstack-ansible openstack.osa.setup_infrastructure -e galera_force_
↪bootstrap=true
```

6. Once infrastructure playboks are done, its turn of OpenStack services to be deployed. Most of the services are fine to be ran with limits, but some, like keystone, are not. So we run keystone playbook separately from all others:

```
# openstack-ansible openstack.osa.keystone
# openstack-ansible openstack.osa.setup_openstack --limit '!keystone_all',
↪localhost,infra<node-ID>,infra<node-ID>-host_containers
```

Test new infra nodes

After creating a new infra node, test that the node runs correctly by launching a new instance. Ensure that the new node can respond to a networking connection test through the **`ping`** command. Log in to your monitoring system, and verify that the monitors return a green signal for the new node.

Add a compute host

Use the following procedure to add a compute host to an operational cluster.

1. Configure the host as a target host. See the target hosts configuration section of the deploy guide for more information.
2. Edit the `/etc/openstack_deploy/openstack_user_config.yml` file and add the host to the `compute_hosts` stanza.

   If necessary, also modify the `used_ips` stanza.
3. If the cluster is utilizing Telemetry/Metering (ceilometer), edit the `/etc/openstack_deploy/conf.d/ceilometer.yml` file and add the host to the `metering-compute_hosts` stanza.
4. Run the following commands to add the host. Replace `NEW_HOST_NAME` with the name of the new host.

```
# cd /opt/openstack-ansible/playbooks
# openstack-ansible openstack.osa.setup_hosts --limit localhost,NEW_HOST_
↪NAME
# openstack-ansible openstack.osa.openstack_hosts_setup -e openstack_
↪hosts_group=nova_compute --tags openstack_hosts-file
# openstack-ansible openstack.osa.setup_openstack --limit localhost,NEW_
↪HOST_NAME
```

Alternatively you can try using new compute nodes deployment script `/opt/openstack-ansible/scripts/add-compute.sh`.

You can provide this script with extra tasks that will be executed before or right after OpenStack-Ansible roles. To do so you should set environment variables `PRE_OSA_TASKS` or `POST_OSA_TASKS` with plays to run devided with semicolon:

```
# export POST_OSA_TASKS="/opt/custom/setup.yml --limit HOST_NAME;/opt/
↪custom/tasks.yml --tags deploy"
# /opt/openstack-ansible/scripts/add-compute.sh HOST_NAME,HOST_NAME_2
```

Test new compute nodes

After creating a new node, test that the node runs correctly by launching an instance on the new node:

```
$ openstack server create --image IMAGE --flavor m1.tiny \
--key-name KEY --availability-zone ZONE:HOST:NODE \
--nic net-id=UUID SERVER
```

Ensure that the new instance can respond to a networking connection test through the **`ping`** command. Log in to your monitoring system, and verify that the monitors return a green signal for the new node.

Using Ansible tags

In Ansible, a tag is a label you can assign to tasks, allowing you to run only the tasks you need instead of the whole playbook. This is especially handy in large playbooks for example, if you have 2030 tasks but just need to restart a service or make some changes in configuration, you can tag those tasks and run them individually.

The following tags are available in OpenStack Ansible:

- `common-mq`
- `common-service`
- `common-db`
- `pki`
- `post-install`
- `haproxy-service-config`
- `ceph`
- `uwsgi`
- `systemd-service`
- `<service>-install`
- `<service>-config`
- `<service>-key`

common-mq

Handles tasks for setting up and configuring RabbitMQ. Use this tag when you need to reconfigure virtual hosts, users, or their privileges without affecting the rest of the deployment.

Example:

```
# openstack-ansible openstack.osa.nova --tags common-mq
```

common-service

Manages service configuration inside Keystone, such as service catalog entries, service user existence, and user privileges.

Example:

```
# openstack-ansible openstack.osa.nova --tags common-service,post-install
```

common-db

Creates and configures databases, including user creation, and permission assignments. Run this tag if database credential or permissions need to be refreshed or corrected.

Example:

```
# openstack-ansible openstack.osa.neutron --tags common-db
```

pki

Manages certificates and public key infrastructure. Use it when renewing, replacing, or troubleshooting SSL/TLS certificates.

Example:

```
# openstack-ansible openstack.osa.setup_infrastructure -e pki_regen_cert=true␣
↪--tags pki
```

post-install

Runs tasks after the main installation and configuration are complete. This tag is used for final adjustments, applying changes in configuration files, and validation checks. Run this tag when youve made changes that require only applying updated configuration.

Example:

```
# openstack-ansible openstack.osa.cinder --tags post-install
```

haproxy-service-config

Configures HAProxy for routing traffic between services. Use this tag if HAProxy settings change or a new service backend is added.

Example:

```
# openstack-ansible haproxy-install.yml --tags haproxy-service-config
```

ceph

Deploys and configures Ceph clients and related components. Use this tag for tasks such as adding new monitors or upgrading Ceph clients to a different version, as well as other Ceph-related configuration updates.

Example:

```
# openstack-ansible ceph-install.yml --tags ceph
```

uwsgi

Sets up and configures uWSGI processes. Useful when adjusting process counts, sockets, or performance tuning.

Example:

```
# openstack-ansible openstack.osa.setup_openstack --tags uwsgi
```

systemd-service

Manages systemd unit components, ensuring they are configured as expected and allowing overrides to be applied. Use this tag when you need to adjust unit files or restart services in a controlled way.

Example:

```
# openstack-ansible openstack.osa.designate --tags systemd-service
```

<service>-install

Installs a specific OpenStack service (replace `<service>` with the service name). A tag including the word `install` handles only software installation tasks it deploys the necessary packages and binaries on the target host. Use this tag when you only need to install or reinstall service software without changing its configuration or running it.

Example:

```
# openstack-ansible openstack.osa.designate --tags designate-install
```

<service>-config

Configures a specific OpenStack service (replace <service> with the service name). This tag applies configuration files, directories, and service-specific settings. It usually covers a broad set of tasks beyond post-install, and may include systemd-service, pki, common-mq or common-db service tags. Run this tag when applying updated configurations to a service that is already installed.

Example:

```
# openstack-ansible openstack.osa.cinder --tags cinder-config
```

<service>-key

This tag is used to generate and distribute SSH certificates, issued through `openstack.osa.ssh_keypairs` role.

This is currently in-use by Keystone, Nova and Swift roles.

Example:

```
# openstack-ansible openstack.osa.nova --tags nova-key
```

Remove a compute host

The OpenStack-Ansible Operator Tooling repository contains a playbook for removing a compute host from an OpenStack-Ansible environment. To remove a compute host, follow the below procedure.

Note

This guide describes how to remove a compute node from an OpenStack-Ansible environment completely. Perform these steps with caution, as the compute node will no longer be in service after the steps have been completed. This guide assumes that all data and instances have been properly migrated.

1. Disable all OpenStack services running on the compute node. This can include, but is not limited to, the `nova-compute` service and the neutron agent service:

   Note

   Ensure this step is performed first.

   ```
   # Run these commands on the compute node to be removed
   # systemctl stop nova-compute
   # systemctl stop neutron-openvswitch-agent
   ```

2. Clone the `openstack-ansible-ops` repository to your deployment host:

   ```
   $ git clone https://opendev.org/openstack/openstack-ansible-ops \
     /opt/openstack-ansible-ops
   ```

3. Run the `remove_compute_node.yml` Ansible playbook with the `host_to_be_removed` user variable set:

   ```
   $ cd /opt/openstack-ansible-ops/ansible_tools/playbooks
   openstack-ansible remove_compute_node.yml \
   -e host_to_be_removed="<name-of-compute-host>"
   ```

4. After the playbook completes, remove the compute node from the OpenStack-Ansible configuration file in `/etc/openstack_deploy/openstack_user_config.yml`.

Recover a compute host failure

The following procedure addresses Compute node failure if shared storage is used.

Note

If shared storage is not used, data can be copied from the `/var/lib/nova/instances` directory on the failed Compute node `${FAILED_NODE}` to another node `${RECEIVING_NODE}`before performing the following procedure. Please note this method is not supported.

1. Re-launch all instances on the failed node.
2. Invoke the MariaDB command line tool.
3. Generate a list of instance UUIDs hosted on the failed node:

```
mysql> select uuid from instances where host = '${FAILED_NODE}' and␣
↪deleted = 0;
```

4. Set instances on the failed node to be hosted on a different node:

```
mysql> update instances set host ='${RECEIVING_NODE}' where host = '$
↪{FAILED_NODE}' \
and deleted = 0;
```

5. Reboot each instance on the failed node listed in the previous query to regenerate the XML files:

```
# nova reboot hard $INSTANCE_UUID
```

6. Find the volumes to check the instance has successfully booted and is at the login:

```
mysql> select nova.instances.uuid as instance_uuid, cinder.volumes.id \
as voume_uuid, cinder.volumes.status, cinder.volumes.attach_status, \
cinder.volumes.mountpoint, cinder.volumes,display_name from \
cinder.volumes inner join nova.instances on cinder.volumes.instance_
↪uuid=nova.instances.uuid \
where nova.instances.host = '${FAILED_NODE}';
```

7. If rows are found, detach and re-attach the volumes using the values listed in the previous query:

```
# nova volume-detach $INSTANCE_UUID $VOLUME_UUID && \
# nova volume-attach $INSTANCE_UUID $VOLUME_UUID $VOLUME_MOUNTPOINT
```

8. Rebuild or replace the failed node as described in *Add a compute host*.

Replacing failed hardware

It is essential to plan and know how to replace failed hardware in your cluster without compromising your cloud environment.

Consider the following to help establish a hardware replacement plan:

- What type of node am I replacing hardware on?

- Can the hardware replacement be done without the host going down? For example, a single disk in a RAID-10.
- If the host DOES have to be brought down for the hardware replacement, how should the resources on that host be handled?

If you have a Compute (nova) host that has a disk failure on a RAID-10, you can swap the failed disk without powering the host down. On the other hand, if the RAM has failed, you would have to power the host down. Having a plan in place for how you will manage these types of events is a vital part of maintaining your OpenStack environment.

For a Compute host, shut down instances on the host before it goes down. For a Block Storage (cinder) host using non-redundant storage, shut down any instances with volumes attached that require that mount point. Unmount the drive within your operating system and re-mount the drive once the Block Storage host is back online.

Shutting down the Compute host

If a Compute host needs to be shut down:

1. Disable the `nova-compute` binary:

```
# nova service-disable --reason "Hardware replacement" HOSTNAME nova-
↪compute
```

2. List all running instances on the Compute host:

```
# nova list --all-t --host <compute_name> | awk '/ACTIVE/ {print $2}' > \
/home/user/running_instances && for i in `cat /home/user/running_
↪instances`; do nova stop $i ; done
```

3. Use SSH to connect to the Compute host.
4. Confirm all instances are down:

```
# virsh list --all
```

5. Shut down the Compute host:

```
# shutdown -h now
```

6. Once the Compute host comes back online, confirm everything is in working order and start the instances on the host. For example:

```
# cat /home/user/running_instances
# do nova start $instance
  done
```

7. Enable the `nova-compute` service in the environment:

```
# nova service-enable HOSTNAME nova-compute
```

Shutting down the Block Storage host

If a LVM backed Block Storage host needs to be shut down:

1. Disable the `cinder-volume` service:

```
# cinder service-list --host CINDER SERVICE NAME INCLUDING @BACKEND
# cinder service-disable CINDER SERVICE NAME INCLUDING @BACKEND \
cinder-volume --reason 'RAM maintenance'
```

2. List all instances with Block Storage volumes attached:

```
# mariadb cinder -BNe 'select instance_uuid from volumes where deleted=0
↪'\
'and host like "%<cinder host>%"' | tee /home/user/running_instances
```

3. Shut down the instances:

```
# cat /home/user/running_instances | xargs -n1 nova stop
```

4. Verify the instances are shutdown:

```
# cat /home/user/running_instances | xargs -n1 nova show | grep -F vm_
↪state
```

5. Shut down the Block Storage host:

```
# shutdown -h now
```

6. Replace the failed hardware and validate the new hardware is functioning.

7. Enable the `cinder-volume` service:

```
# cinder service-enable CINDER SERVICE NAME INCLUDING @BACKEND cinder-
↪volume
```

8. Verify the services on the host are reconnected to the environment:

```
# cinder service-list --host CINDER SERVICE NAME INCLUDING @BACKEND
```

9. Start your instances and confirm all of the instances are started:

```
# cat /home/user/running_instances | xargs -n1 nova start
# cat /home/user/running_instances | xargs -n1 nova show | grep -F vm_
↪state
```

Destroying containers

1. To destroy a container, execute the following:

```
# openstack-ansible openstack.osa.containers_lxc_destroy --limit␣
↪localhost,<container name|container group>
```

Note

You will be asked two questions:

> Are you sure you want to destroy the LXC containers? Are you sure you want to destroy the LXC container data?
>
> The first will just remove the container but leave the data in the bind mounts and logs. The second will remove the data in the bind mounts and logs too.

Warning

If you remove the containers and data for the entire galera_server container group you will lose all your databases! Also, if you destroy the first container in many host groups you will lose other important items like certificates, keys, etc. Be sure that you understand what youre doing when using this tool.

2. To create the containers again, execute the following:

```
# cd /opt/openstack-ansible/playbooks
# openstack-ansible openstack.osa.containers_lxc_create --limit localhost,
↪lxc_hosts,<container name|container
  group>

The lxc_hosts host group must be included as the playbook and roles␣
↪executed require the
use of facts from the hosts.
```

Accessibility for multi-region Object Storage

In multi-region Object Storage utilizing separate database backends, objects are retrievable from an alternate location if the `default_project_id` for a user in the keystone database is the same across each database backend.

Important

It is recommended to perform the following steps before a failure occurs to avoid having to dump and restore the database.

If a failure does occur, follow these steps to restore the database from the Primary (failed) Region:

1. Record the Primary Region output of the `default_project_id` for the specified user from the user table in the keystone database:

Note

The user is `admin` in this example.

```
# mariadb -e "SELECT default_project_id from keystone.user WHERE \
  name='admin';"

+----------------------------------+
| default_project_id               |
+----------------------------------+
| 76ef6df109744a03b64ffaad2a7cf504 |
+----------------+
```

2. Record the Secondary Region output of the `default_project_id` for the specified user from the user table in the keystone database:

```
# mariadb -e "SELECT default_project_id from keystone.user WHERE \
  name='admin';"

+----------------------------------+
| default_project_id               |
+----------------------------------+
| 69c46f8ad1cf4a058aa76640985c     |
+----------------------------------+
```

3. In the Secondary Region, update the references to the `project_id` to match the ID from the Primary Region:

```
# export PRIMARY_REGION_TENANT_ID="76ef6df109744a03b64ffaad2a7cf504"
# export SECONDARY_REGION_TENANT_ID="69c46f8ad1cf4a058aa76640985c"

# mariadb -e "UPDATE keystone.assignment set \
target_id='${PRIMARY_REGION_TENANT_ID}' \
WHERE target_id='${SECONDARY_REGION_TENANT_ID}';"

# mariadb -e "UPDATE keystone.user set \
default_project_id='${PRIMARY_REGION_TENANT_ID}' WHERE \
default_project_id='${SECONDARY_REGION_TENANT_ID}';"

# mariadb -e "UPDATE keystone.project set \
id='${PRIMARY_REGION_TENANT_ID}' WHERE \
id='${SECONDARY_REGION_TENANT_ID}';"
```

The user in the Secondary Region now has access to objects PUT in the Primary Region. The Secondary Region can PUT objects accessible by the user in the Primary Region.

Backup and restore your cloud

For disaster recovery purposes, it is a good practice to perform regular backups of the database, configuration files, network information, and OpenStack service details in your environment. For an OpenStack cloud deployed using OpenStack-Ansible, back up the `/etc/openstack_deploy/` directory.

Backup and restore the `/etc/openstack_deploy/` directory

The `/etc/openstack_deploy/` directory contains a live inventory, host structure, network information, passwords, and options that are applied to the configuration files for each service in your OpenStack deployment. Backup the `/etc/openstack_deploy/` directory to a remote location.

To restore the `/etc/openstack_deploy/` directory, copy the backup of the directory to your cloud environment.

Database backups and recovery

MariaDB data is available on the infrastructure nodes. You can recover databases, and rebuild the Galera cluster. For more information, see *Galera cluster recovery*.

Ansible Logging Guide

OpenStack-Ansible provides flexible options for collecting and analyzing Ansible execution logs. Operators can use the default logging configuration, or integrate with ARA Records Ansible for advanced reporting.

Default Log File

By default, OpenStack-Ansible stores all playbook logs in:

```
/openstack/log/ansible-logging/ansible.log
```

This location is defined by the `ANSIBLE_LOG_PATH` environment variable.

To change the path, override it in the deployment configuration file:

```
/etc/openstack_deploy/user.rc
```

ARA Integration

For richer reporting, OpenStack-Ansible can be integrated with **ARA (Ansible Run Analysis)**.

During the bootstrap process, set the following variable:

```
export SETUP_ARA=true
./bootstrap-ansible.sh
```

This installs the ARA client and configures it as an Ansible callback.

The client requires an ARA server to store data. The server is not included in OpenStack-Ansible and must be deployed by the operator. The recommended method is to use the `recordsansible.ara` collection.

On the deployment host, configure the client with:

```
export ARA_API_CLIENT=http
export ARA_API_SERVER=https://ara.example.com
export ARA_API_INSECURE=False
export ARA_API_USERNAME=ara
export ARA_API_PASSWORD=
```

If you prefer not to run an ARA server, you can still generate local reports:

```
export ARA_REPORT_TYPE=html
```

Each playbook run will then produce an HTML report stored on the deploy host.

Troubleshooting

This chapter is intended to help troubleshoot and resolve operational issues in an OpenStack-Ansible deployment.

Networking

This section focuses on troubleshooting general host-to-host communication required for the OpenStack control plane to function properly.

This does not cover any networking related to instance connectivity.

These instructions assume an OpenStack-Ansible installation using LXC containers, VXLAN overlay for ML2/OVS and Geneve overlay for the ML2/OVN drivers.

Network List

1. `HOST_NET` (Physical Host Management and Access to Internet)
2. `MANAGEMENT_NET` (LXC container network used OpenStack Services)
3. `OVERLAY_NET` (VXLAN overlay network for OVS, Geneve overlay network for OVN)

Useful network utilities and commands:

```
# ip link show [dev INTERFACE_NAME]
# arp -n [-i INTERFACE_NAME]
# ip [-4 | -6] address show [dev INTERFACE_NAME]
# ping <TARGET_IP_ADDRESS>
# tcpdump [-n -nn] < -i INTERFACE_NAME > [host SOURCE_IP_ADDRESS]
# brctl show [BRIDGE_ID]
# iptables -nL
# arping [-c NUMBER] [-d] <TARGET_IP_ADDRESS>
```

Troubleshooting host-to-host traffic on HOST_NET

Perform the following checks:

- Check physical connectivity of hosts to physical network
- Check interface bonding (if applicable)
- Check VLAN configurations and any necessary trunking to edge ports on physical switch
- Check VLAN configurations and any necessary trunking to uplink ports on physical switches (if applicable)
- Check that hosts are in the same IP subnet or have proper routing between them
- Check there are no firewall (firewalld, ufw, etc.) rules applied to the hosts that would deny traffic

IP addresses should be applied to physical interface, bond interface, tagged sub-interface, or in some cases the bridge interface:

```
# ip address show dev bond0
14: bond0: <BROADCAST,MULTICAST,MASTER,UP,LOWER_UP> mtu 1500..UP...
link/ether a0:a0:a0:a0:a0:01 brd ff:ff:ff:ff:ff:ff
inet 10.240.0.44/22 brd 10.240.3.255 scope global bond0
   valid_lft forever preferred_lft forever
...
```

Troubleshooting host-to-host traffic on MANAGEMENT_NET

Perform the following checks:

- Check physical connectivity of hosts to physical network
- Check interface bonding (if applicable)
- Check VLAN configurations and any necessary trunking to edge ports on physical switch
- Check VLAN configurations and any necessary trunking to uplink ports on physical switches (if applicable)
- Check that hosts are in the same subnet or have proper routing between them
- Check there are no firewall (firewalld, ufw, etc.) rules applied to the hosts that would deny traffic
- Check to verify that physical interface is in the bridge
- Check to verify that veth-pair end from container is in `br-mgmt`

IP address should be applied to `br-mgmt`:

```
# ip address show dev br-mgmt
18: br-mgmt: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500...UP...
link/ether a0:a0:a0:a0:a0:01 brd ff:ff:ff:ff:ff:ff
inet 172.29.236.44/22 brd 172.29.239.255 scope global br-mgmt
   valid_lft forever preferred_lft forever
...
```

IP address should be applied to `eth1` inside the LXC container:

```
# ip address show dev eth1
59: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500...UP...
link/ether b1:b1:b1:b1:b1:01 brd ff:ff:ff:ff:ff:ff
inet 172.29.236.55/22 brd 172.29.239.255 scope global eth1
   valid_lft forever preferred_lft forever
   ...
```

`br-mgmt` should contain veth-pair ends from all containers and a physical interface or tagged-subinterface:

```
# brctl show br-mgmt
bridge name bridge id          STP enabled  interfaces
br-mgmt     8000.abcdef12345   no           11111111_eth1
                                            22222222_eth1
                                            ...
                                            bond0.100
```

(continues on next page)

(continued from previous page)

```
                                                99999999_eth1
                                                ...
```

You can also use ip command to display bridges:

```
# ip link show master br-mgmt

12: bond0.100@bond0: ... master br-mgmt state UP mode DEFAULT group default␣
↪qlen 1000
....
51: 11111111_eth1_eth1@if3: ... master br-mgmt state UP mode DEFAULT group␣
↪default qlen 1000
....
```

Troubleshooting host-to-host traffic on OVERLAY_NET

Perform the following checks:

- Check physical connectivity of hosts to physical network
- Check interface bonding (if applicable)
- Check VLAN configurations and any necessary trunking to edge ports on physical switch
- Check VLAN configurations and any necessary trunking to uplink ports on physical switches (if applicable)
- Check that hosts are in the same subnet or have proper routing between them
- Check there are no firewall (firewalld, ufw, etc.) rules applied to the hosts that would deny traffic
- Check to verify that physcial interface is in the bridge
- Check to verify that veth-pair end from container is in `br-vxlan`

IP address should be applied to `br-vxlan`:

```
# ip address show dev br-vxlan
21: br-vxlan: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500...UP...
link/ether a0:a0:a0:a0:a0:02 brd ff:ff:ff:ff:ff:ff
inet 172.29.240.44/22 brd 172.29.243.255 scope global br-vxlan
   valid_lft forever preferred_lft forever
   ...
```

Checking services

You can check the status of an OpenStack service by accessing every controller node and running the **`systemctl status <SERVICE_NAME>`**.

See the following links for additional information to verify OpenStack services:

- Identity service (keystone)
- Image service (glance)
- Compute service (nova)

- Networking service (neutron)
- Block Storage service (cinder)
- Object Storage service (swift)

Some useful commands to manage LXC see *Command Line Reference*.

Restarting services

Restart your OpenStack services by accessing every controller node. Some OpenStack services will require restart from other nodes in your environment.

The following table lists the commands to restart an OpenStack service.

Table 2: Restarting OpenStack services

OpenStack service	Commands
Image service	`# systemctl restart glance-api`

Compute service (controller node)	`# systemctl restart nova-api-os-compute`
	`# systemctl restart nova-scheduler`
	`# systemctl restart nova-conductor`
	`# systemctl restart nova-api-metadata`
	`# systemctl restart nova-novncproxy (if using noVNC)`
	`# systemctl restart nova-spicehtml5proxy (if using SPICE)`

Compute service (compute node)	`# systemctl restart nova-compute`

Networking service (controller node, for OVS)	`# systemctl restart neutron-server`
	`# systemctl restart neutron-dhcp-agent`
	`# systemctl restart neutron-l3-agent`
	`# systemctl restart neutron-metadata-agent`
	`# systemctl restart neutron-openvswitch-agent`

Networking service (compute node, for OVS)	`# systemctl restart neutron-openvswitch-agent`

Networking service (controller node, for OVN)	`# systemctl restart neutron-server`
	`# systemctl restart neutron-ovn-maintenance-worker`
	`# systemctl restart neutron-periodic-workers`

Networking service (compute node, for OVN)	`# systemctl restart neutron-ovn-metadata-agent`

Block Storage service	`# systemctl restart cinder-api`
	`# systemctl restart cinder-backup`
	`# systemctl restart cinder-scheduler`
	`# systemctl restart cinder-volume`

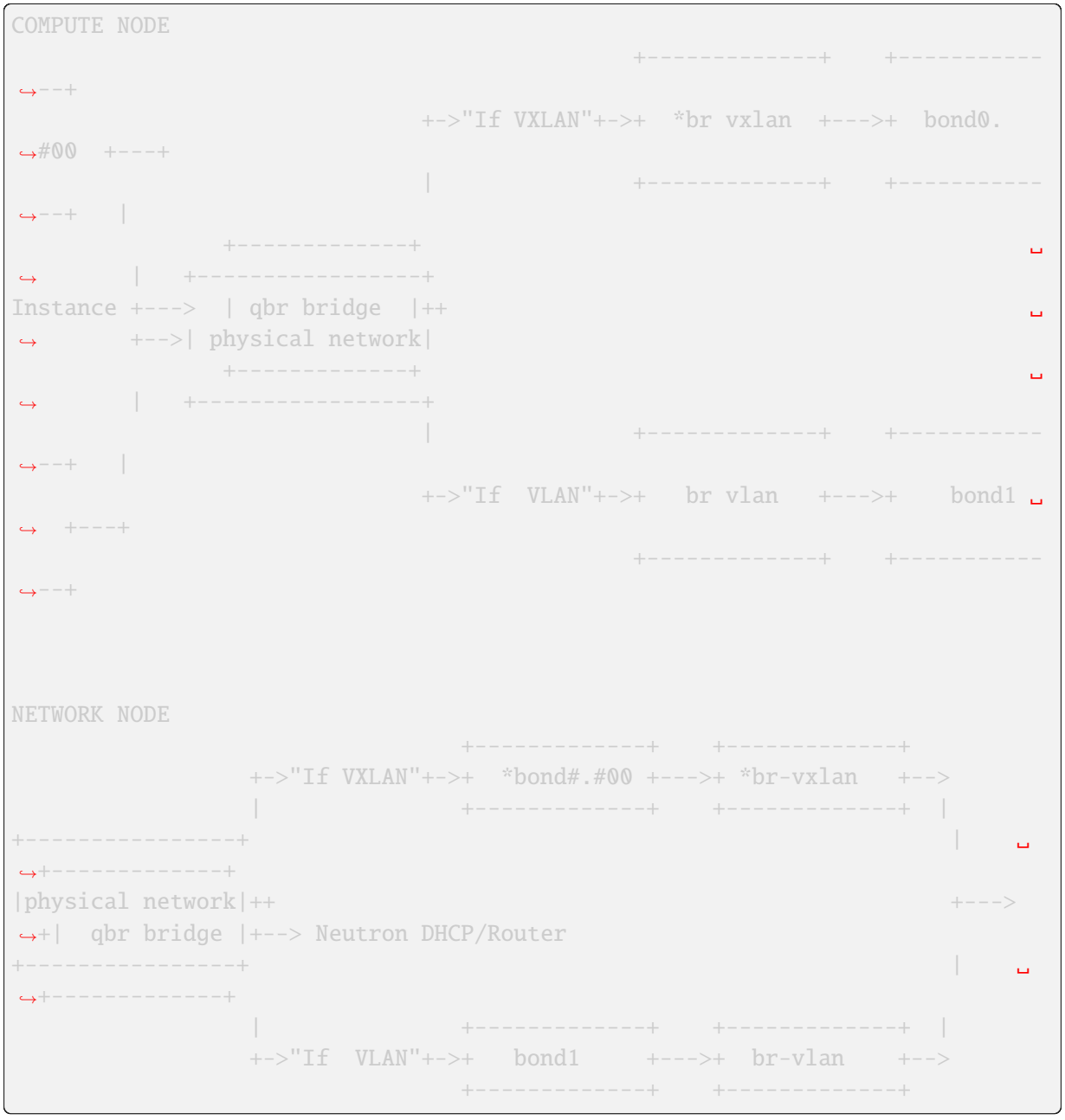
Shared Filesystems service	`# systemctl restart manila-api`
	`# systemctl restart manila-data`
	`# systemctl restart manila-share`
	`# systemctl restart manila-scheduler`

Object Storage service	`# systemctl restart swift-account-auditor`
	`# systemctl restart swift-account-server`
	`# systemctl restart swift-account-reaper`
	`# systemctl restart swift-account-replicator`
	`# systemctl restart swift-container-auditor`
	`# systemctl restart swift-container-server`
	`# systemctl restart swift-container-reconciler`
	`# systemctl restart swift-container-replicator`

Troubleshooting instance connectivity issues

This section will focus on troubleshooting general instances connectivity communication. This does not cover any networking related to instance connectivity. This is assuming a OpenStack-Ansible install using LXC containers, VXLAN overlay for ML2/OVS and Geneve overlay for the ML2/OVN driver.

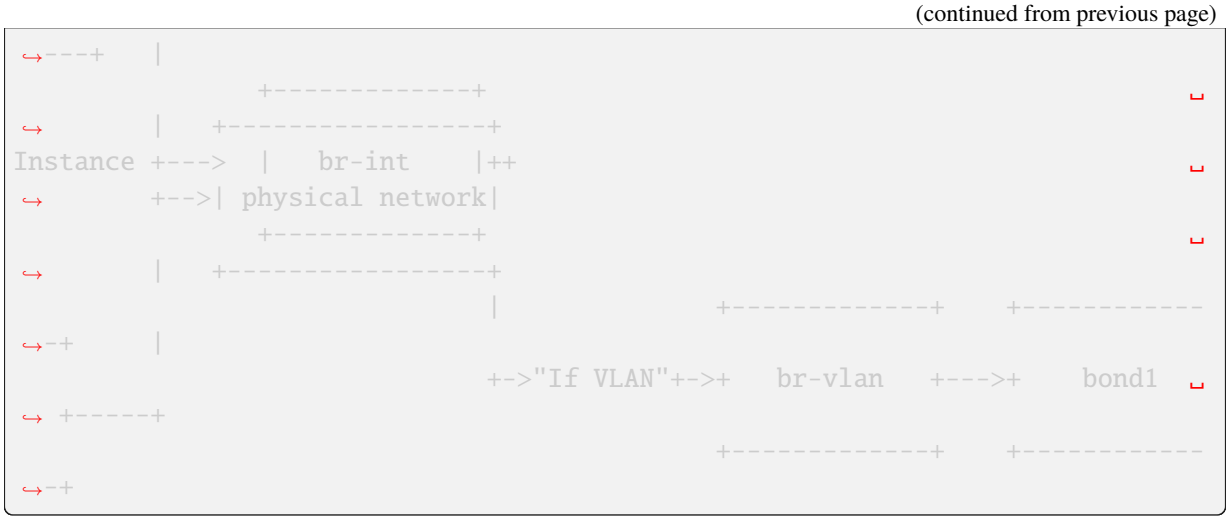
Data flow example (for OVS)



Data flow example (for OVN)

(continues on next page)

(continued from previous page)



Preliminary troubleshooting questions to answer:

- Which compute node is hosting the instance in question?
- Which interface is used for provider network traffic?
- Which interface is used for VXLAN (Geneve) overlay?
- Is there connectivity issue ingress to the instance?
- Is there connectivity issue egress from the instance?
- What is the source address of the traffic?
- What is the destination address of the traffic?
- Is there a Neutron Router in play?
- Which network node (container) is the router hosted?
- What is the project network type?

If VLAN:

Does physical interface show link and all VLANs properly trunked across physical network?

No:

- Check cable, seating, physical switchport configuration, interface/bonding configuration, and general network configuration. See general network troubleshooting documentation.

Yes:

- Good!
- Continue!

Important

Do not continue until physical network is properly configured.

Does the instances IP address ping from networks DHCP namespace or other instances in the same network?

No:

- Check nova console logs to see if the instance ever received its IP address initially.
- Check `security-group-rules`, consider adding allow ICMP rule for testing.
- Check that OVS bridges contain the proper interfaces on compute and network nodes.
- Check Neutron DHCP agent logs.
- Check syslogs.
- Check Neutron Open vSwitch agent logs.

Yes:

- Good! This suggests that the instance received its IP address and can reach local network resources.
- Continue!

Important

Do not continue until instance has an IP address and can reach local network resources like DHCP.

Does the instances IP address ping from the gateway device (Neutron Router namespace or another gateway device)?

No:

- Check Neutron L3 agent logs (if applicable).
- Check Neutron Open vSwitch logs.
- Check physical interface mappings.
- Check Neutron router ports (if applicable).
- Check that OVS bridges contain the proper interfaces on compute and network nodes.
- Check `security-group-rules`, consider adding allow ICMP rule for testing. In case of using OVN check additionally:
- Check ovn-controller on all nodes.
- Verify ovn-northd is running and DBs are healthy.
- Ensure ovn-metadata-agent is active.
- Review logs for ovn-controller, ovn-northd.

Yes:

- Good! The instance can ping its intended gateway. The issue may be north of the gateway or related to the provider network.
- Check gateway or host routes on the Neutron subnet.
- Check `security-group-rules`, consider adding ICMP rule for testing.
- Check Floating IP associations (if applicable).
- Check Neutron Router external gateway information (if applicable).

- Check upstream routes, NATs or access-control-lists.

Important

Do not continue until the instance can reach its gateway.

If VXLAN (Geneve):

Does physical interface show link and all VLANs properly trunked across physical network?

No:

- Check cable, seating, physical switchport configuration, interface/bonding configuration, and general network configuration. See general network troubleshooting documentation.

Yes:

- Good!
- Continue!

Important

Do not continue until physical network is properly configured.

Are VXLAN (Geneve) VTEP addresses able to ping each other?

No:

- Check `br-vxlan` interface on Compute and Network nodes.
- Check veth pairs between containers and Linux bridges on the host.
- Check that OVS bridges contain the proper interfaces on compute and network nodes.

Yes:

- Check ml2 config file for local VXLAN (Geneve) IP and other VXLAN (Geneve) configuration settings.
- **Check VTEP learning method (multicast or l2population):**
  - If multicast, make sure the physical switches are properly allowing and distributing multicast traffic.

Important

Do not continue until VXLAN (Geneve) endpoints have reachability to each other.

Does the instances IP address ping from networks DHCP namespace or other instances in the same network?

No:

- Check Nova console logs to see if the instance ever received its IP address initially.
- Check `security-group-rules`, consider adding allow ICMP rule for testing.

- Check that OVS bridges contain the proper interfaces on compute and network nodes.
- Check Neutron DHCP agent logs.
- Check syslogs.
- Check Neutron Open vSwitch agent logs.
- Check that Bridge Forwarding Database (fdb) contains the proper entries on both the compute and Neutron agent container (`ovs-appctl fdb/show br-int`).

Yes:

- Good! This suggests that the instance received its IP address and can reach local network resources.

Important

Do not continue until instance has an IP address and can reach local network resources.

Does the instances IP address ping from the gateway device (Neutron Router namespace or another gateway device)?

No:

- Check Neutron L3 agent logs (if applicable).
- Check Neutron Open vSwitch agent logs.
- Check physical interface mappings.
- Check Neutron router ports (if applicable).
- Check that OVS bridges contain the proper interfaces on compute and network nodes.
- Check `security-group-rules`, consider adding allow ICMP rule for testing.
- Check that Bridge Forwarding Database (fdb) contains the proper entries on both the compute and Neutron agent container (`ovs-appctl fdb/show br-int`). In case of using OVN check additionally:
- Check ovn-controller on all nodes.
- Verify ovn-northd is running and DBs are healthy.
- Ensure ovn-metadata-agent is active.
- Review logs for ovn-controller, ovn-northd.

Yes:

- Good! The instance can ping its intended gateway.
- Check gateway or host routes on the Neutron subnet.
- Check `security-group-rules`, consider adding ICMP rule for testing.
- Check Neutron Floating IP associations (if applicable).
- Check Neutron Router external gateway information (if applicable).
- Check upstream routes, NATs or `access-control-lists`.

Diagnose Image service issues

The `glance-api` handles the API interactions and image store.

To troubleshoot problems or errors with the Image service, refer to `/var/log/glance-api.log` inside the glance api container.

You can also conduct the following activities which may generate logs to help identity problems:

1. Download an image to ensure that an image can be read from the store.
2. Upload an image to test whether the image is registering and writing to the image store.
3. Run the `openstack image list` command to ensure that the API and registry is working.

For an example and more information, see Verify operation and Manage Images.

Cached Ansible facts issues

At the beginning of a playbook run, information about each host is gathered, such as:

- Linux distribution
- Kernel version
- Network interfaces

To improve performance, particularly in large deployments, you can cache host facts and information.

OpenStack-Ansible enables fact caching by default. The facts are cached in JSON files within `/etc/openstack_deploy/ansible_facts`.

Fact caching can be disabled by running `export ANSIBLE_CACHE_PLUGIN=memory`. To set this permanently, set this variable in `/usr/local/bin/openstack-ansible.rc`. Refer to the Ansible documentation on fact caching for more details.

Forcing regeneration of cached facts

Cached facts may be incorrect if the host receives a kernel upgrade or new network interfaces. Newly created bridges also disrupt cache facts.

This can lead to unexpected errors while running playbooks, and require cached facts to be regenerated.

Run the following command to remove all currently cached facts for all hosts:

```
# rm /etc/openstack_deploy/ansible_facts/*
```

New facts will be gathered and cached during the next playbook run.

To clear facts for a single host, find its file within `/etc/openstack_deploy/ansible_facts/` and remove it. Each host has a JSON file that is named after its hostname. The facts for that host will be regenerated on the next playbook run.

Rebuilding Python Virtual Environments

In certain situations, you may need to forcefully rebuild a services Python virtual environment. This can be required if the `python_venv_build` role fails (for example, due to temporary package conflicts), or if you want to reset the virtual environment after manual modifications.

Two variables control the rebuild process:

- `venv_rebuild` resets the virtual environment to its intended state without rebuilding wheels. This is usually sufficient when the service version has not changed and only the venv state needs to be restored.
- `venv_wheels_rebuild` additionally forces a rebuild of the Python wheels. This may be required if the service version has changed or if its venv requirements were modified.

To trigger a rebuild for a specific service, re-run its playbook with the following environment variables:

```
# reset the venv state
openstack-ansible openstack.osa.nova -e venv_rebuild=true

# reset the venv state and rebuild wheels
openstack-ansible openstack.osa.nova -e venv_rebuild=true -e venv_wheels_
↪rebuild=true
```

Container networking issues

All LXC containers on the host have at least two virtual Ethernet interfaces:

- *eth0* in the container connects to *lxcbr0* on the host
- *eth1* in the container connects to *br-mgmt* on the host

Note

Some containers, such as `cinder`, `glance`, `neutron_agents`, and `swift_proxy` have more than two interfaces to support their functions.

Predictable interface naming

On the host, all virtual Ethernet devices are named based on their container as well as the name of the interface inside the container:

```
${CONTAINER_UNIQUE_ID}_${NETWORK_DEVICE_NAME}
```

As an example, an all-in-one (AIO) build might provide a utility container called *aio1_utility_container-d13b7132*. That container will have two network interfaces: *d13b7132_eth0* and *d13b7132_eth1*.

Another option would be to use the LXC tools to retrieve information about the utility container. For example:

```
# lxc-info -n aio1_utility_container-d13b7132

Name:           aio1_utility_container-d13b7132
State:          RUNNING
PID:            8245
IP:             10.0.3.201
IP:             172.29.237.204
CPU use:        79.18 seconds
BlkIO use:      678.26 MiB
Memory use:     613.33 MiB
KMem use:       0 bytes
```

(continues on next page)

(continued from previous page)

```
Link:           d13b7132_eth0
 TX bytes:      743.48 KiB
 RX bytes:      88.78 MiB
 Total bytes:   89.51 MiB
Link:           d13b7132_eth1
 TX bytes:      412.42 KiB
 RX bytes:      17.32 MiB
 Total bytes:   17.73 MiB
```

The `Link:` lines will show the network interfaces that are attached to the utility container.

Review container networking traffic

To dump traffic on the `br-mgmt` bridge, use `tcpdump` to see all communications between the various containers. To narrow the focus, run `tcpdump` only on the desired network interface of the containers.

Restoring inventory from backup

OpenStack-Ansible maintains a running archive of inventory. If a change has been introduced into the system that has broken inventory or otherwise has caused an unforseen issue, the inventory can be reverted to an early version. The backup file `/etc/openstack_deploy/backup_openstack_inventory.tar` contains a set of timestamped inventories that can be restored as needed.

Example inventory restore process.

```
mkdir /tmp/inventory_restore
cp /etc/openstack_deploy/backup_openstack_inventory.tar /tmp/inventory_
↪restore/backup_openstack_inventory.tar
cd /tmp/inventory_restore
tar xf backup_openstack_inventory.tar
# Identify the inventory you wish to restore as the running inventory
cp openstack_inventory.json-YYYYMMDD_SSSSSS.json /etc/openstack_deploy/
↪openstack_inventory.json
cd -
rm -rf /tmp/inventory_restore
```

At the completion of this operation the inventory will be restored to the earlier version.

Compatibility Matrix

All of the OpenStack-Ansible releases are compatible with specific sets of operating systems and their versions. Operating Systems have their own lifecycles, however we may drop their support before end of their EOL because of various reasons:

- OpenStack requires a higher version of a library (ie. libvirt)
- Python version
- specific dependencies
- etc.

However, we do try to provide `upgrade` releases where we support both new and old Operating System versions, providing deployers the ability to properly upgrade their deployments to the new Operating System release.

In CI we test upgrades between releases only for `source` deployments. This also includes CI testing of upgrade path between SLURP releases.

Below you will find the support matrix of Operating Systems for OpenStack-Ansible releases.

Note

Compatability matrix for legacy releases of OpenStack-Ansible can be found on this page: *Compatibility Matrix of Legacy releases*.

Operating systems with experimental support are marked with `E` in the table.

Operating System Compatibility Matrix

Compatibility Matrix of Legacy releases

This page contains compatability matrix of releases that are either in Extended Maintanence or already reached End of Life. We keep such matrix for historical reasons mainly and for deployments that forgot to get updated in time.

Operating systems with experimental support are marked with `E` in the table.

Operating System Compatibility Matrix

Minor version upgrade

Upgrades between minor versions of OpenStack-Ansible require updating the repository clone to the latest minor release tag, updating the Ansible roles, and then running playbooks against the target hosts. This section provides instructions for those tasks.

Prerequisites

To avoid issues and simplify troubleshooting during the upgrade, disable the security hardening role by setting the `apply_security_hardening` variable to `False` in the `user_variables.yml` file, and backup your OpenStack-Ansible installation.

Execute a minor version upgrade

A minor upgrade typically requires the following steps:

1. Change directory to the cloned repositorys root directory:

```
# cd /opt/openstack-ansible
```

2. Fetch the latest updates from the remote repository to ensure all branches and tags are up to date before proceeding with the minor version upgrade:

```
# git fetch
```

3. Ensure that your OpenStack-Ansible code is on the latest 2026.1 (Gazpacho) tagged release:

```
# git checkout master
```

4. Update all the dependent roles to the latest version:

```
# ./scripts/bootstrap-ansible.sh
```

5. Change to the playbooks directory:

```
# cd playbooks
```

6. Update the hosts:

```
# openstack-ansible openstack.osa.setup_hosts -e package_state=latest
```

7. Update the infrastructure:

```
# openstack-ansible -e rabbitmq_upgrade=true \
openstack.osa.setup_infrastructure
```

8. Update all OpenStack services:

```
# openstack-ansible openstack.osa.setup_openstack -e package_state=latest
```

Note

You can limit upgrades to specific OpenStack components. See the following section for details.

Upgrade specific components

You can limit upgrades to specific OpenStack components by running each of the component playbooks against groups.

For example, you can update only the Compute hosts by running the following command:

```
# openstack-ansible openstack.osa.nova --limit nova_compute
```

To update only a single Compute host, run the following command:

```
# openstack-ansible openstack.osa.nova --limit <node-name>
```

Note

Skipping the `nova-key` tag is necessary so that the keys on all Compute hosts are not gathered.

To see which hosts belong to which groups, use the `openstack-ansible-inventory-manage` script to show all groups and their hosts. For example:

1. Change directory to the repository clone root directory:

```
# cd /opt/openstack-ansible
```

2. Show all groups and which hosts belong to them:

```
# openstack-ansible-inventory-manage -G
```

3. Show all hosts and the groups to which they belong:

```
# openstack-ansible-inventory-manage -g
```

To see which hosts a playbook runs against, and to see which tasks are performed, run the following commands (for example):

1. See the hosts in the `nova_compute` group that a playbook runs against:

```
# openstack-ansible openstack.osa.nova --limit nova_compute \
                                       --list-hosts
```

2. See the tasks that are executed on hosts in the `nova_compute` group:

```
# openstack-ansible openstack.osa.nova --limit nova_compute \
                                       --skip-tags 'nova-key' \
                                       --list-tasks
```

Upgrading a specific component within the same OpenStack-Ansible version

Sometimes you may need to apply the latest security patches or bug fixes for a service while remaining on the same stable branch. This can be done by overriding the Git installation branch for that service, which instructs OpenStack-Ansible to pull the most recent code from the branch you are already tracking. But using branches directly as `<service>_git_install_branch` is highly discouraged. Every time the playbook is re-run, the service may be upgraded to a newer commit, which can lead to inconsistent versions between hosts (for example, when adding a new compute node later).

So the recommended practice is to take the HEAD commit SHA of the desired stable branch and set it explicitly. To find the latest SHA of the `stable/2025.1` branch, you can run (e.g. for Nova):

```
git ls-remote https://opendev.org/openstack/nova refs/heads/stable/2025.1
```

And use that SHA in your configuration to ensure consistent versions across all hosts in your `user_variables.yml`:

```
nova_git_install_branch: {{SHA}}
```

And run:

```
openstack-ansible openstack.osa.nova --tags nova-install
```

The playbook will fetch and install the code from the specified branch or commit SHA, applying the latest patches and fixes as defined. Using a pinned SHA ensures consistent versions across all hosts, while following the branch directly will always pull its current HEAD.

We can verify the version of the service before and after the upgrade (dont forget to load required environment variables):

```
$ ansible -m shell -a '/openstack/venvs/nova-{{ openstack_release }}/bin/pip3␣
↪freeze | grep nova' nova_all
infra1-nova-api-container-e5dbbe38 | CHANGED | rc=0 >>
```

(continues on next page)

(continued from previous page)

```
nova==31.0.1.dev12
infra2-nova-api-container-0c5d0203 | CHANGED | rc=0 >>
nova==31.0.1.dev12
infra3-nova-api-container-d791a43e | CHANGED | rc=0 >>
nova==31.0.1.dev12
compute | CHANGED | rc=0 >>
nova==31.0.1.dev12
```

After the upgrade to the latest patches in the same branch:

```
$ ansible -m shell -a '/openstack/venvs/nova-{{ openstack_release }}/bin/pip3␣
↪freeze | grep nova' nova_all
infra1-nova-api-container-e5dbbe38 | CHANGED | rc=0 >>
nova==31.1.1.dev9
infra2-nova-api-container-0c5d0203 | CHANGED | rc=0 >>
nova==31.1.1.dev9
infra3-nova-api-container-d791a43e | CHANGED | rc=0 >>
nova==31.1.1.dev9
compute | CHANGED | rc=0 >>
nova==31.1.1.dev9
```

Note

This approach is not limited to Nova. You can apply the same method to any other OpenStack service managed by OpenStack-Ansible by overriding its corresponding `<service>_git_install_branch` variable.

Always ensure that the branch is up-to-date and compatible with the rest of your deployment before proceeding.

Major upgrades

This guide provides information about the upgrade process from 2025.2 (Flamingo) or 2025.1 (Epoxy) to 2026.1 (Gazpacho) for OpenStack-Ansible.

Note

You can upgrade between sequential releases or between releases marked as SLURP.

Introduction

For upgrades between major versions, the OpenStack-Ansible repository provides playbooks and scripts to upgrade an environment. The `run-upgrade.sh` script runs each upgrade playbook in the correct order, or playbooks can be run individually if necessary. Alternatively, a deployer can upgrade manually.

For more information about the major upgrade process, see *Upgrading by using a script* and *Upgrading manually*.

Warning

Upgrading to master is not recommended. Master is under heavy development, and is not stable. Test this on a development environment first.

Upgrading by using a script

The 2026.1 (Gazpacho) release series of OpenStack-Ansible contains the code for migrating from 2025.2 (Flamingo) or 2025.1 (Epoxy) to 2026.1 (Gazpacho).

Running the upgrade script

To upgrade from 2025.2 (Flamingo) or 2025.1 (Epoxy) to 2026.1 (Gazpacho) by using the upgrade script, perform the following steps in the `openstack-ansible` directory:

1. Change directory to the repository clone root directory:

   ```
   # cd /opt/openstack-ansible
   ```

2. Run the following commands:

   ```
   # git checkout master
   # ./scripts/run-upgrade.sh
   ```

For more information about the steps performed by the script, see *Upgrading manually*.

Upgrading manually

Manual upgrades are useful for scoping the changes in the upgrade process (for example, in very large deployments with strict SLA requirements), or performing other upgrade automation beyond that provided by OpenStack-Ansible.

The steps detailed here match those performed by the `run-upgrade.sh` script. You can safely run these steps multiple times.

Preflight checks

Before starting with the upgrade, perform preflight health checks to ensure your environment is stable. If any of those checks fail, ensure that the issue is resolved before continuing.

Check out the 2026.1 (Gazpacho) release

Ensure that your OpenStack-Ansible code is on the latest 2026.1 (Gazpacho) tagged release.

```
# git checkout master
```

Backup the existing OpenStack-Ansible configuration

Make a backup of the configuration of the environment:

```
# source_series_backup_file="/openstack/backup-openstack-ansible-2025.2.tar.
↪gz"
# tar zcf ${source_series_backup_file} /etc/openstack_deploy /etc/ansible/ /
↪usr/local/bin/openstack-ansible.rc
```

Bootstrap the new Ansible and OpenStack-Ansible roles

To ensure that there is no currently set ANSIBLE_INVENTORY to override the default inventory location, we unset the environment variable.

```
# unset ANSIBLE_INVENTORY
```

Bootstrap Ansible again to ensure that all OpenStack-Ansible role dependencies are in place before you run playbooks from the 2026.1 (Gazpacho) release.

```
# cd /opt/openstack-ansible
# ./scripts/bootstrap-ansible.sh
```

Implement changes to OpenStack-Ansible configuration

If there have been any OpenStack-Ansible variable name changes or environment/inventory changes, there is a playbook to handle those changes to ensure service continuity in the environment when the new playbooks run. The playbook is tagged to ensure that any part of it can be executed on its own or skipped. Please review the contents of the playbook for more information.

```
# openstack-ansible openstack.osa.upgrade.deploy_config_changes
```

Note

With upgrade to 2024.2 (Dalmatian) release and beyond, usage of RabbitMQ Quorum Queues is mandatory to ensure high availability of queues. If you had previously set `oslomsg_rabbit_quorum_queues: false`, please consider migrating before continuing with this upgrade which uses RabbitMQ 4.x.

Please, check RabbitMQ maintenance for more information about switching between Quourum and HA Queues.

Upgrade hosts

Before installing the infrastructure and OpenStack, update the host machines.

Warning

Usage of non-trusted certificates for RabbitMQ is not possible due to requirements of newer `amqp` versions.

After that you can proceed with standard OpenStack upgrade steps:

```
# openstack-ansible openstack.osa.setup_hosts --limit '!galera_all:!rabbitmq_
↪all' -e package_state=latest
```

This command is the same setting up hosts on a new installation. The `galera_all` and `rabbitmq_all` host groups are excluded to prevent reconfiguration and restarting of any of those containers as they need to be updated, but not restarted.

Once that is complete, upgrade the final host groups with the flag to prevent container restarts.

```
# openstack-ansible openstack.osa.setup_hosts -e 'lxc_container_allow_
↪restarts=false' --limit 'galera_all:rabbitmq_all'
```

Upgrade infrastructure

We can now go ahead with the upgrade of all the infrastructure components. To ensure that RabbitMQ and MariaDB are upgraded, we pass the appropriate flags.

Warning

Please make sure you are running RabbitMQ version 3.13 or later before proceeding to this step. Upgrade of RabbitMQ to version 4.0 (default for 2024.2) from prior version will result in playbook failure.

At this point you can minorly upgrade RabbitMQ with the following command:

`openstack-ansible openstack.osa.rabbitmq_server -e rabbitmq_upgrade=true -e rabbitmq_package_version=3.13.7-1`

Also ensure that you have migrated from mirrored queues (HA queues) to Quorum queues before the upgrade, as mirrored queues are no longer supported after upgrade.

```
# openstack-ansible openstack.osa.setup_infrastructure -e 'galera_upgrade=true
↪' -e 'rabbitmq_upgrade=true' -e package_state=latest
```

With this complete, we can now restart the MariaDB containers one at a time, ensuring that each is started, responding, and synchronized with the other nodes in the cluster before moving on to the next steps. This step allows the LXC container configuration that you applied earlier to take effect, ensuring that the containers are restarted in a controlled fashion.

```
# openstack-ansible openstack.osa.tools.galera_cluster_rolling_restart
```

Upgrade OpenStack

We can now go ahead with the upgrade of all the OpenStack components.

```
# openstack-ansible openstack.osa.setup_openstack -e package_state=latest
```

Upgrade Ceph

With each OpenStack-Ansible version we define default Ceph client version that will be installed on Glance/Cinder/Nova hosts and used by these services. If you want to preserve the previous version of the Ceph client during an OpenStack-Ansible upgrade, you will need to override a variable `ceph_stable_release` in your user_variables.yml

If Ceph has been deployed as part of an OpenStack-Ansible deployment using the roles maintained by the Ceph-Ansible project you will also need to upgrade the Ceph version. Each OpenStack-Ansible release is tested only with specific Ceph-Ansible release and Ceph upgrades are not checked in any OpenStack-Ansible integration tests. So we do not test or guarantee an upgrade path for such deployments. In this case tests should be done in a lab environment before upgrading.

Warning

Ceph related playbooks are included as part of `openstack.osa.setup_infrastructure` and `openstack.osa.setup_openstack` playbooks, so you should be cautious when running them during OpenStack upgrades. If you have `upgrade_ceph_packages: true` in your user variables or provided `-e upgrade_ceph_packages=true` as argument and run `setup-infrastructure.yml` this will result in Ceph package being upgraded as well.

In order to upgrade Ceph in the deployment you will need to run:

```
# openstack-ansible /etc/ansible/roles/ceph-ansible/infrastructure-playbooks/
↪rolling_update.yml
```

Distribution upgrades

This guide provides information about upgrading from one distribution release to the next.

Note

This guide was last updated when upgrading from Ubuntu 20.04 (Focal Fossa) to Ubuntu 22.04 (Jammy Jellyfish) during the Antelope (2023.1) release. For earlier releases please see other versions of the guide.

Introduction

OpenStack-Ansible supports operating system distribution upgrades during specific release cycles. These can be observed by consulting the operating system compatibility matrix, and identifying where two versions of the same operating system are supported.

Upgrades should be performed in the order specified in this guide to minimise the risk of service interruptions. Upgrades must also be carried out by performing a fresh installation of the target systems operating system, before running OpenStack-Ansible to install services on this host.

Ordering

This guide includes a suggested order for carrying out upgrades. This may need to be adapted dependent on the extent to which you have customised your OpenStack-Ansible deployment.

Critically, it is important to consider when you upgrade repo hosts/containers. At least one repo host should be upgraded before you upgrade any API hosts/containers. The last repo host to be upgraded should be the primary, and should not be carried out until after the final service which does not support limit is upgraded.

If you have a multi-architecture deployment, then at least one repo host of each architecture will need to be upgraded before upgrading any other hosts which use that architecture.

If this order is adapted, it will be necessary to restore some files to the repo host from a backup part-way through the process. This will be necessary if no repo hosts remain which run the older operating system version, which prevents older packages from being built.

Beyond these requirements, a suggested order for upgrades is a follows:

1. Infrastructure services (Galera, RabbitMQ, APIs, HAProxy)

   In all cases, secondary or backup instances should be upgraded first

2. Compute nodes
3. Network nodes

Pre-Requisites

- Ensure that all hosts in your target deployment have been installed and configured using a matching version of OpenStack-Ansible. Ideally perform a minor upgrade to the latest version of the OpenStack release cycle which you are currently running first in order to reduce the risk of encountering bugs.
- Check any OpenStack-Ansible variables which you customise to ensure that they take into account the new and old operating system version (for example custom package repositories and version pinning).
- Perform backups of critical data, in particular the Galera database in case of any failures. It is also recommended to back up the /var/www/repo directory on the primary repo host in case it needs to be restored mid-upgrade.
- Identify your primary HAProxy/Galera/RabbitMQ/repo infrastructure host

  In a simple 3 infrastructure hosts setup, these services/containers usually end up being all on the the same host.

  The primary will be the LAST box youll want to reinstall.

  - HAProxy/Keepalived

    Finding your HAProxy/Keepalived primary is as easy as

    ```
    ssh {{ external_lb_vip_address }}
    ```

    Or preferably if youve installed HAProxy with stats, like so:

    ```
    haproxy_stats_enabled: true
    haproxy_stats_bind_address: "{{ external_lb_vip_address }}"
    ```

    and can visit https://admin:password@external_lb_vip_address:1936/ and read Statistics Report for pid # on infrastructure_host

- Ensure RabbitMQ is running with all feature flags enabled to avoid conflicts when re-installing nodes. If any are listed as disabled then enable them via the console on one of the nodes:

```
rabbitmqctl list_feature_flags
rabbitmqctl enable_feature_flag all
```

Warnings

- During the upgrade process, some OpenStack services cannot be deployed by using Ansibles limit. As such, it will be necessary to deploy some services to mixed operating system versions at the same time.

  The following services are known to lack support for limit:

  - RabbitMQ
  - Repo Server
  - Keystone

- In the same way as OpenStack-Ansible major (and some minor) upgrades, there will be brief interruptions to the entire Galera and RabbitMQ clusters during the upgrade which will result in brief service interruptions.
- When taking down memcached instances for upgrades you may encounter performance issues with the APIs.

Deploying Infrastructure Hosts

1. Define redeployed host as environment variable

   This will serve as a shortcut for future operations and will make following the instruction more error-prone. For example:

   ```
   export REINSTALLED_HOST="infra3"
   ```

2. Disable HAProxy back ends (optional)

   If you wish to minimise error states in HAProxy, services on hosts which are being reinstalled can be set in maintenance mode (MAINT).

   Log into your primary HAProxy/Keepalived and run something similar to

   ```
   echo "disable server repo_all-back/<infrahost>_repo_container-<hash>" |␣
   ↪socat /var/run/haproxy.stat stdio
   ```

   for each API or service instance you wish to disable.

   You can also use a playbook for this:

   ```
   openstack-ansible openstack.osa.tools.set_haproxy_backends_state -e␣
   ↪hostname=${REINSTALLED_HOST} -e backend_state=disabled
   ```

   Or if youve enabled haproxy_stats as described above, you can visit https://admin:password@external_lb_vip_address:1936/ and select them and set state to `MAINT`.

3. Reinstall an infrastructure hosts operating system

   As noted above, this should be carried out for non-primaries first, ideally starting with a repo host.

4. Clearing out stale information

   1. Removing stale ansible-facts

      ```
      rm /etc/openstack_deploy/ansible-facts/${REINSTALLED_HOST}*
      ```

      (* because were deleting all container facts for the host as well)

   2. If RabbitMQ was running on this host

      We forget it by running these commands on another RabbitMQ host.

```
rabbitmqctl cluster_status
rabbitmqctl forget_cluster_node rabbit@removed_host_rabbitmq_
↪container
```

3. If GlusterFS was running on this host (repo nodes)

   We forget it by running these commands on another repo host. Note that we have to tell Gluster we are intentionally reducing the number of replicas. N should be set to the number of repo servers minus 1. Existing gluster peer names can be found using the gluster peer status command.

```
gluster volume remove-brick gfs-repo replica N removed_host_gluster_
↪peer:/gluster/bricks/1 force
gluster peer detach removed_host_gluster_peer
```

5. Do generic preparation of reinstalled host

```
openstack-ansible openstack.osa.setup_hosts --limit localhost,$
↪{REINSTALLED_HOST}*
```

6. This step should be executed when you are re-configuring one of HAProxy hosts

   Since configuration of HAProxy backends happens during individual service provisioning, we need to ensure that all backends are configured before enabling Keepalived to select this host.

   Commands below will configure all required backends on HAProxy nodes:

```
openstack-ansible openstack.osa.haproxy --limit localhost,${REINSTALLED_
↪HOST} --skip-tags keepalived
openstack-ansible openstack.osa.repo --tags haproxy-service-config
openstack-ansible openstack.osa.galera_server --tags haproxy-service-
↪config
openstack-ansible openstack.osa.rabbitmq_server --tags haproxy-service-
↪config
openstack-ansible openstack.osa.setup_openstack --tags haproxy-service-
↪config
```

   Once this is done, you can deploy Keepalived again:

```
openstack-ansible openstack.osa.haproxy --tags keepalived --limit␣
↪localhost,${REINSTALLED_HOST}
```

   After that you might want to ensure that local backends remain disabled. You can also use the playbook for this:

```
openstack-ansible openstack.osa.tools.set_haproxy_backends_state -e␣
↪hostname=${REINSTALLED_HOST} -e backend_state=disabled --limit $
↪{REINSTALLED_HOST}
```

7. If it is NOT a primary, install everything on the new host

```
openstack-ansible openstack.osa.setup_infrastructure --limit localhost,
↪repo_all,rabbitmq_all,${REINSTALLED_HOST}*
```

(continues on next page)

(continued from previous page)

```
openstack-ansible openstack.osa.setup_openstack --limit localhost,
↪keystone_all,${REINSTALLED_HOST}*
```

(* because we need to include containers in the limit)

8. If it IS a primary, do these steps

    1. Temporarily set your primary Galera in MAINT in HAProxy.

        In order to prevent role from making your primary Galera as UP in HAProxy, create an empty file `/var/tmp/clustercheck.disabled` . You can do this with ad-hoc:

        ```
        cd /opt/openstack-ansible
        ansible -m file -a "path=/var/tmp/clustercheck.disabled state=touch"
        ↪"${REINSTALLED_HOST}*:&galera_all"
        ```

        Once its done you can run playbook to install MariaDB to the destination

        ```
        openstack-ansible openstack.osa.galera_server --limit localhost,$
        ↪{REINSTALLED_HOST}* -e galera_server_bootstrap_node="{{ groups[
        ↪'galera_all'][-1] }}"
        ```

        Youll now have MariaDB running, and it should be synced with non-primaries.

        To check that verify MariaDB cluster status by executing from host running primary MariaDB following command:

        ```
        mariadb -e 'SHOW STATUS LIKE "wsrep_cluster_%";'
        ```

        In case node is not getting synced you might need to restart the mariadb.service and verify everything is in order.

        ```
        systemctl restart mariadb.service
        mariadb
        MariaDB> SHOW STATUS LIKE "wsrep_cluster_%";
        MariaDB> SHOW DATABASES;
        ```

        Once MariaDB cluster is healthy you can remove the file that disables backend from being used by HAProxy.

        ```
        ansible -m file -a "path=/var/tmp/clustercheck.disabled state=absent
        ↪" "${REINSTALLED_HOST}*:&galera_all"
        ```

    2. We can move on to RabbitMQ primary

        ```
        openstack-ansible openstack.osa.rabbitmq_server -e rabbitmq_primary_
        ↪cluster_node="{{ hostvars[groups['rabbitmq_all'][-1]]['ansible_
        ↪facts']['hostname'] }}"
        ```

    3. Now the repo host primary

        ```
        openstack-ansible openstack.osa.repo -e glusterfs_bootstrap_node="{{␣
        ↪groups['repo_all'][-1] }}"
        ```

4. Everything should now be in a working state and we can finish it off with

```
openstack-ansible openstack.osa.setup_infrastructure --limit␣
↪localhost,repo_all,rabbitmq_all,${REINSTALLED_HOST}*
openstack-ansible openstack.osa.setup_openstack --limit localhost,
↪keystone_all,${REINSTALLED_HOST}*
```

9. Adjust HAProxy status

   If HAProxy was set into MAINT mode, this can now be removed for services which have been restored.

   For the repo host, it is important that the freshly installed hosts are set to READY in HAProxy, and any which remain on the old operating system are set to MAINT.

   You can use the playbook to re-enable all backends from the host:

```
openstack-ansible openstack.osa.tools.set_haproxy_backends_state -e␣
↪hostname=${REINSTALLED_HOST} -e backend_state=enabled
```

Deploying Compute and Network Hosts

1. Disable the hypervisor service on compute hosts and migrate any instances to another available hypervisor.
2. Reinstall a hosts operating system
3. Clear out stale ansible-facts

```
rm /etc/openstack_deploy/ansible-facts/${REINSTALLED_HOST}*
```

   (* because were deleting all container facts for the host as well)

4. Execute the following:

```
openstack-ansible openstack.osa.setup_hosts --limit localhost,$
↪{REINSTALLED_HOST}*
openstack-ansible openstack.osa.setup_infrastructure --limit localhost,$
↪{REINSTALLED_HOST}*
openstack-ansible openstack.osa.setup_openstack --limit localhost,$
↪{REINSTALLED_HOST}*
```

   (* because we need to include containers in the limit)

5. Re-instate compute node hypervisor UUIDs

   Compute nodes should have their UUID stored in the file /var/lib/nova/compute_id and the nova-compute service restarted. UUIDs can be found from the command lineopenstack hypervisor list.

   Alternatively, the following Ansible can be used to automate these actions:

```
openstack-ansible openstack.osa.tools.nova_restore_compute_id --limit $
↪{REINSTALLED_HOST}
```

1.1.2 User Guide

In this section, you will find user stories and examples relevant to deploying OpenStack-Ansible.

For information on how to deploy your OpenStack-Ansible cloud, refer to the Deployment Guide for step-by-step instructions on how to deploy the OpenStack packages and dependencies on your cloud using OpenStack-Ansible.

For information on how to manage and operate OpenStack-Ansible, see the *Operations Guide*.

For information on how to contribute, extend or develop OpenStack-Ansible, see the Developer Documentation.

For in-depth technical information, see the *OpenStack-Ansible Reference*.

Quickstart: AIO

All-in-one (AIO) builds are a great way to perform an OpenStack-Ansible build for:

- a development environment
- an overview of how all of the OpenStack services fit together
- a simple lab deployment

Although AIO builds arent recommended for large production deployments, theyre great for smaller proof-of-concept deployments.

Absolute minimum server resources (currently used for gate checks):

- 8 vCPUs
- 50GB free disk space on the root partition
- 8GB RAM

Recommended server resources:

- CPU/motherboard that supports hardware-assisted virtualization
- 8 CPU Cores
- 80GB free disk space on the root partition, or 60GB+ on a blank secondary disk. Using a secondary disk requires the use of the `bootstrap_host_data_disk_device` parameter. Please see *Building an AIO* for more details.
- 16GB RAM

It is *possible* to perform AIO builds within a virtual machine for demonstration and evaluation, but your virtual machines will perform poorly unless nested virtualization is available. For production workloads, multiple nodes for specific roles are recommended.

Building an AIO

Overview

There are three steps to running an AIO build, with an optional first step should you need to customize your build:

- Prepare the host
- Bootstrap Ansible and the required roles

- Bootstrap the AIO configuration
- Run playbooks

Prepare the host

When building an AIO on a new server, it is recommended that all system packages are upgraded and then reboot into the new kernel:

Note

Execute the following commands and scripts as the root user.

```
## Ubuntu / Debian
# apt-get update
# apt-get dist-upgrade
# reboot
```

```
## CentOS Stream / Rocky Linux
# dnf upgrade
# dnf install git-core
# systemctl stop firewalld
# systemctl mask firewalld
# reboot
```

Note

Before rebooting, in `/etc/sysconfig/selinux`, make sure that `SELINUX=enforcing` is changed to `SELINUX=disabled`. SELinux enabled is not currently supported in OpenStack-Ansible for CentOS Stream or Rocky Linux due to a lack of maintainers for the feature.

Note

If you are installing with limited connectivity, please review *Installing with limited connectivity* before proceeding.

Bootstrap Ansible and the required roles

Start by cloning the OpenStack-Ansible repository and changing into the repository root directory:

```
# git clone https://opendev.org/openstack/openstack-ansible \
    /opt/openstack-ansible
# cd /opt/openstack-ansible
```

Next switch the applicable branch/tag to be deployed from. Note that deploying from the head of a branch may result in an unstable build due to changes in flight and upstream OpenStack changes. For a test (for example, not a development) build it is usually best to checkout the latest tagged version.

```
# # List all existing tags.
```

```
# git tag -l

# # Checkout the stable branch and find just the latest tag
# git checkout master
# git describe --abbrev=0 --tags

# # Checkout the latest tag from either method of retrieving the tag.
# git checkout master
```

Note

The 2026.1 (Gazpacho) release is only compatible with Debian 12 (bookworm), Debian 13 (trixie), Ubuntu 24.04 (Noble Numbat), CentOS 10 Stream and derivatives of RHEL 10 such as Rocky Linux.

The next step is to bootstrap Ansible and the Ansible roles for the development environment.

Run the following to bootstrap Ansible and the required roles:

```
# scripts/bootstrap-ansible.sh
```

Note

You might encounter an error while running the Ansible bootstrap script when building some of the Python extensions (like pycrypto) which says:

```
configure: error: cannot run C compiled programs.
```

The reason of this failure might be resulting from a noexec mount flag used for the filesystem associated with /tmp which you can check by running the following command:

```
# mount | grep $(df /tmp | tail -n +2 | awk '{print $1}') | grep noexec
```

If this is the case you can specify an alternate path which does not have this mount option set:

```
# TMPDIR=/var/tmp scripts/bootstrap-ansible.sh
```

Bootstrap the AIO configuration

In order for all the services to run, the host must be prepared with the appropriate disks partitioning, packages, network configuration and configurations for the OpenStack Deployment.

By default the AIO bootstrap scripts deploy a base set of OpenStack services with sensible defaults for the purpose of a gate check, development or testing system.

Review the bootstrap-host role defaults file to see various configuration options. Deployers have the option to change how the host is bootstrapped. This is useful when you wish the AIO to make use of a secondary data disk, or when using this role to bootstrap a multi-node development environment.

The bootstrap script is pre-set to pass the environment variable `BOOTSTRAP_OPTS` as an additional option to the bootstrap process. For example, if you wish to set the bootstrap to re-partition a specific secondary storage device (`/dev/sdb`), which will erase all of the data on the device, then execute:

```
# export BOOTSTRAP_OPTS="bootstrap_host_data_disk_device=sdb"
```

Additional options may be implemented by simply concatenating them with a space between each set of options, for example:

```
# export BOOTSTRAP_OPTS="bootstrap_host_data_disk_device=sdb"
# export BOOTSTRAP_OPTS="${BOOTSTRAP_OPTS} bootstrap_host_data_disk_fs_
↪type=xfs"
```

If you are installing with limited connectivity, or you dont have default route set, you will need to define interface for outgoing connections manually

```
# export BOOTSTRAP_OPTS="bootstrap_host_public_interface=eth1"
```

For the default AIO scenario, the AIO configuration preparation is completed by executing:

```
# scripts/bootstrap-aio.sh
```

To add OpenStack Services over and above the bootstrap-aio default services for the applicable scenario, copy the `conf.d` files with the `.aio` file extension into `/etc/openstack_deploy` and rename then to `.yml` files. For example, in order to enable the OpenStack Telemetry services, execute the following:

```
# cd /opt/openstack-ansible/
# cp etc/openstack_deploy/conf.d/{aodh,gnocchi,ceilometer}.yml.aio /etc/
↪openstack_deploy/conf.d/
# for f in $(ls -1 /etc/openstack_deploy/conf.d/*.aio); do mv -v ${f} ${f%.*};
↪ done
```

It is possible to also do this (and change other defaults) during the bootstrap script initial execution by changing the SCENARIO environment variable before running the script. The key word aio will ensure that a basic set of OpenStack services (cinder, glance, horizon, neutron, nova) will be deployed. The key words lxc can be used to set the container back-end, while the key word metal will deploy all services without containers. In order to implement any other services, add the name of the conf.d file name without the *.yml.aio* extension into the SCENARIO environment variable. Each key word should be delimited by an underscore. For example, the following will implement an AIO with barbican, cinder, glance, horizon, neutron, and nova. It will set the cinder storage back-end to ceph and will make use of LXC as the container back-end.

```
# export SCENARIO='aio_lxc_barbican_ceph_ovs'
# scripts/bootstrap-aio.sh
```

To add any global overrides, over and above the defaults for the applicable scenario, edit `/etc/openstack_deploy/user_variables.yml`. In order to understand the various ways that you can override the default behaviour set out in the roles, playbook and group variables, see *Overriding default configuration*.

See the Deployment Guide for a more detailed break down of how to implement your own configuration rather than to use the AIO bootstrap.

Run playbooks

Finally, run the playbooks by executing:

```
# openstack-ansible openstack.osa.setup_hosts
# openstack-ansible openstack.osa.setup_infrastructure
# openstack-ansible openstack.osa.setup_openstack
```

The installation process will take a while to complete, but here are some general estimates:

- Bare metal systems with SSD storage: ~ 30-50 minutes
- Virtual machines with SSD storage: ~ 45-60 minutes
- Systems with traditional hard disks: ~ 90-120 minutes

Once the playbooks have fully executed, it is possible to experiment with various settings changes in `/etc/openstack_deploy/user_variables.yml` and only run individual playbooks. For example, to run the playbook for the Keystone service, execute:

```
# cd /opt/openstack-ansible/playbooks
# openstack-ansible os-keystone-install.yml
```

Interacting with an AIO

Once an AIO has been deployed, you most likely want to interact with it. You can do this via the web interface or one of the many clients or libraries that exist for OpenStack.

Using a GUI

The Horizon web interface provides a graphical interface for interacting with the AIO deployment. By default, the Horizon API is available on port 443 of the host (or port 80, if SSL certificate configuration was disabled). As such, to interact with Horizon, simply browse to the IP of the host.

Note

If the AIO was deployed in a cloud VM, you may need to configure security groups or firewall rules to allow access to the HTTP(S) ports. For example, if the AIO was deployed in an OpenStack VM, you can create and apply a suitable security group for interacting with Horizon like so:

```
$ openstack security group create http \
    --description 'Allow HTTP and HTTPS access'
$ openstack security group rule create http \
    --protocol tcp --dst-port 80 --remote-ip 0.0.0.0/0
$ openstack security group rule create http \
    --protocol tcp --dst-port 443 --remote-ip 0.0.0.0/0
$ openstack server add security group $SERVER http
```

A list of service ports can be found in the OpenStack Install Guide.

This will present a login page. By default, OpenStack-Ansible create a user called `admin`. The password will be the value of the `keystone_auth_admin_password` variable. If you did not configure this variable, OpenStack-Ansible auto-generates one. You can view the configured password in the `/etc/openstack_deploy/user_secrets.yml` file.

```
# grep admin_pass /etc/openstack_deploy/user_secrets.yml
heat_stack_domain_admin_password: <redacted>
keystone_auth_admin_password: <redacted>
radosgw_admin_password: <redacted>
```

Using this username and password combination, log in to Horizon.

Using a client or library

There are a variety of clients and libraries available for interacting with an OpenStack deployment, including as openstackclient, openstacksdk, or gophercloud. These are typically configured using either environment variables sourced from an `openrc` file or the newer `clouds.yaml` file.

OpenStack-Ansible provides the `openstack_openrc` role for creating these configuration files as well as a number of utilities such as *openstackclient*. If the AIO deployment using the `lxc` scenario (the default), these will be availably in the utility container.

```
$ lxc-attach -n `lxc-ls -1 | grep utility`

# ls /root/openrc
/root/openrc

# ls /root/.config/openstack/clouds.yaml
/root/.config/openstack/clouds.yaml

# export OS_CLOUD=default
# openstack project list -c Name -f value
service
admin
```

Alternatively, if the AIO was deployed using the `metal` scenario, these files will be available on the host itself.

```
# ls /root/openrc
/root/openrc

# ls /root/.config/openstack/clouds.yaml
/root/.config/openstack/clouds.yaml
```

If you wish to access the AIO deployment from another host - perhaps your local workstation - you will need either an `openrc` file or `clouds.yaml` file. You can download an `openrc` file from Horizon: simply click the User dropdown in the top-right corner and select `OpenStack RC File`.

Important

You may be tempted to copy the `openrc` or `clouds.yaml` files created by the `openstack_openrc` role. However, these files use the `internal` interface by default. This interface use the management network (`172.29.236.0/22`) , which is not routable from outside the host. If you wish to use these files, you will need to change the interface to `public`.

Note

If the AIO was deployed in a cloud VM, you may need to configure security groups or firewall rules to allow access to the various sevice ports. For example, if the AIO was deployed in an OpenStack VM, you can create and apply a suitable security group for interacting the core services like so:

```
$ openstack security group create openstack-apis \
    --description 'Allow access to various OpenStack services'
$ for port in 8774 8776 9292 9696 5000 8780; do
    openstack security group rule create openstack-apis \
      --protocol tcp --dst-port ${port}:${port} --remote-ip 0.0.0.0/0
  done
$ openstack server add security group $SERVER openstack-apis
```

A list of service ports can be found in the OpenStack Install Guide.

Note

If you have enabled SSL certificate configuration (default), all services will use self-signed certificates. While the host is configured to trust these certificates, this is not the case for other hosts. This will result in HTTPS errors when attempting to interact with the cloud. To resolve this issue, you will need to manually configure certificates on other hosts or ignore SSL issues. To use the self-signed certificate, first copy it to the other hosts. The name and location of the generated certificate are configured by the `pki_authorities` and `pki_trust_store_location` variables respectively, which are used by the `pki` role provided by ansible-role-pki. On an Ubuntu 24.04 host, these will default to `ExampleCorpRoot` and `/usr/local/share/ca-certificates`, respectively. For example:

```
$ scp aio:/usr/local/share/ca-certificates/ExampleCorpRoot.crt ~/.config/
↪openstack/aio.crt
```

Once this is done, configure the `cacert` value in the the definition for your cloud in `clouds.yaml`. For example:

```
clouds:
  aio:
    # ...
    cacert: /home/<username>/.config/openstack/aio.crt
```

Alternatively, you can simply ignore SSL issues by setting `verify: false` in the definition for your cloud in `clouds.yaml`. This will disable SSL verification entirely for this cloud. For example:

```
clouds:
  aio:
    # ...
    verify: false
```

Finally, you can also opt to disable SSL certificate configuration during initial deployment or opt to use an external certificate authority for signing, such as Lets Encrypt. Both topics are outside the scope of this document.

More information about SSL certificate configuration can be found in the *Securing services with SSL certificates*.

Once one of these files have been created, you can use it to interact with your deployment using most standard clients and libraries. For example, to list available projects using *openstackclient*:

```
$ export OS_CLOUD=aio
$ openstack project list -c Name -f value
service
admin
```

Rebooting an AIO

As the AIO includes all three cluster members of MariaDB/Galera, the cluster has to be re-initialized after the host is rebooted.

This is done by executing the following:

```
# cd /opt/openstack-ansible/playbooks
# openstack-ansible -e galera_ignore_cluster_state=true galera-install.yml
```

If this fails to get the database cluster back into a running state, then please make use of the *Galera cluster recovery* section in the *Operations Guide*.

Rebuilding an AIO

Sometimes it may be useful to destroy all the containers and rebuild the AIO. While it is preferred that the AIO is entirely destroyed and rebuilt, this isnt always practical. As such the following may be executed instead:

```
# # Move to the playbooks directory.
# cd /opt/openstack-ansible/playbooks

# # Destroy all of the running containers.
# openstack-ansible lxc-containers-destroy.yml

# # On the host stop all of the services that run locally and not
# #  within a container.
# for i in \
      $(ls /etc/init \
        | grep -e "nova\|swift\|neutron\|cinder" \
        | awk -F'.' '{print $1}'); do \
    service $i stop; \
  done

# # Uninstall the core services that were installed.
# for i in $(pip freeze | grep -e "nova\|neutron\|keystone\|swift\|cinder"); ␣
↪do \
    pip uninstall -y $i; done

# # Remove crusty directories.
# rm -rf /openstack /etc/{neutron,nova,swift,cinder} \
        /var/log/{neutron,nova,swift,cinder}
```

(continues on next page)

(continued from previous page)

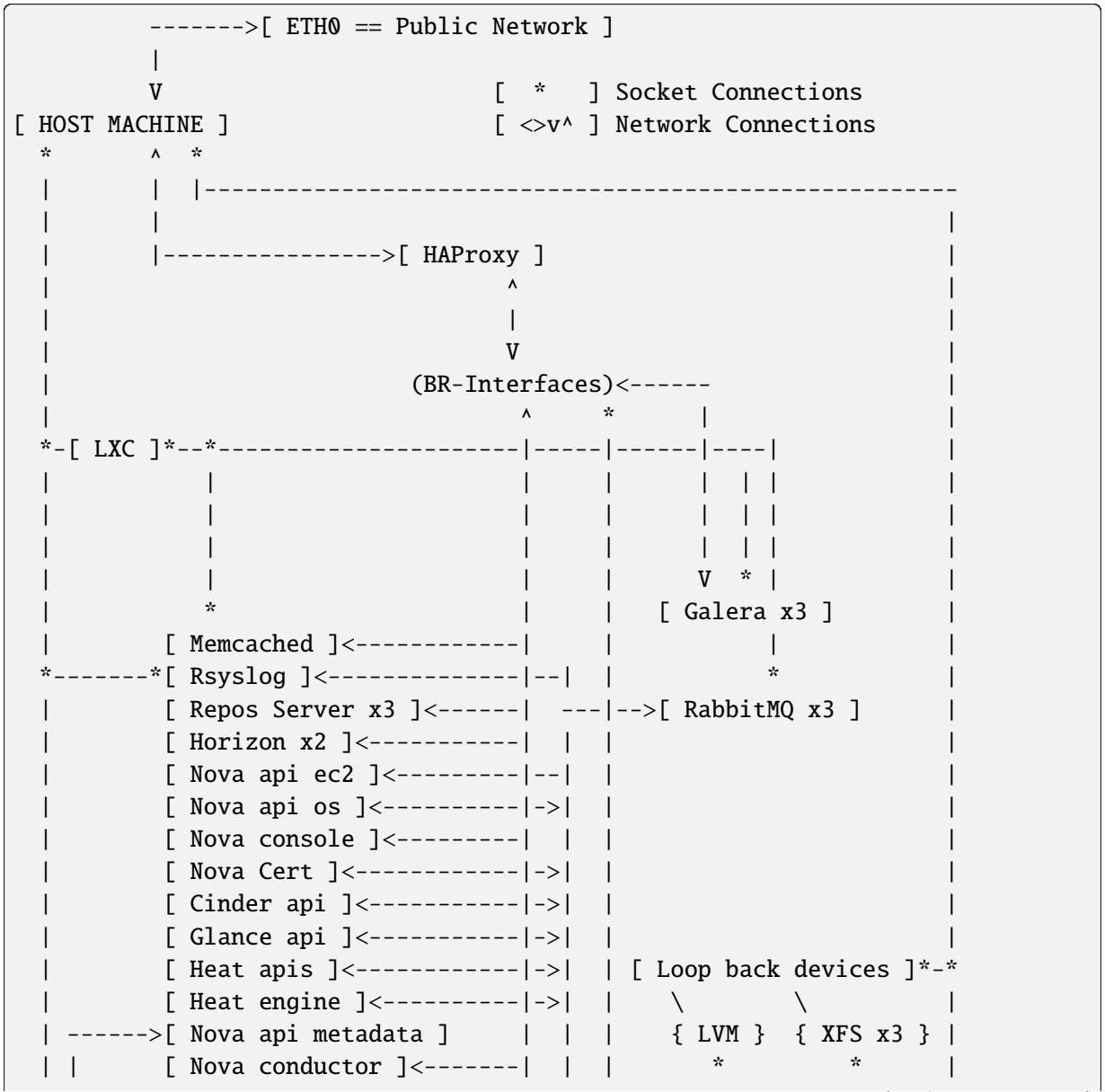
```
# # Remove the pip configuration files on the host
# rm -rf /root/.pip

# # Remove the apt package manager proxy
# rm /etc/apt/apt.conf.d/00apt-cacher-proxy
```

Should an existing AIO environment need to be reinstalled, the most efficient method is to destroy the host operating system and start over. For this reason, AIOs are best run inside of some form of virtual machine or cloud guest.

Reference Diagram for an AIO Build

Here is a basic diagram that attempts to illustrate what the resulting AIO deployment looks like.

This diagram is not to scale and is not even 100% accurate, this diagram was built for informational purposes only and should **ONLY** be used as such.



(continues on next page)

(continued from previous page)

```
 | |----->[ Nova scheduler ]--------|->|  |       |         |      |
 | |      [ Keystone x3 ]<----------|->|  |       |         |      |
 | | |--->[ Neutron agents ]*-------|--|---------------------------*
 | | |    [ Neutron server ]<-------|->|          |         |      |
 | | | |->[ Swift proxy ]<-----------  |          |         |      |
 *-|-|-|-*[ Cinder volume ]*----------------------*         |      |
 | | | |                                |                   |      |
 | | | ---------------------------------------            |      |
 | | ---------------------------------------- |           |      |
 | |          -------------------------|     | |           |      |
 | |          |                              | |           |      |
 | |          V                              | |           *      |
 ---->[ Compute ]*[ Neutron OpenvSwitch ]<---| |->[ Swift storage ]-
```

Network architectures

OpenStack-Ansible supports a number of different network architectures, and can be deployed using a single network interface for non-production workloads or using multiple network interfaces or bonded interfaces for production workloads.

The OpenStack-Ansible reference architecture segments traffic using VLANs across multiple network interfaces or bonds. Common networks used in an OpenStack-Ansible deployment can be observed in the following table:

Network	CIDR	VLAN
Management Network	172.29.236.0/22	10
Overlay Network	172.29.240.0/22	30
Storage Network	172.29.244.0/22	20

The `Management Network`, also referred to as the `container network`, provides management of and communication between the infrastructure and OpenStack services running in containers or on metal. The `management network` uses a dedicated VLAN typically connected to the `br-mgmt` bridge, and may also be used as the primary interface used to interact with the server via SSH.

The `Overlay Network`, also referred to as the `tunnel network`, provides connectivity between hosts for the purpose of tunnelling encapsulated traffic using VXLAN, Geneve, or other protocols. The `overlay network` uses a dedicated VLAN typically connected to the `br-vxlan` bridge.

The `Storage Network` provides segregated access to Block Storage from OpenStack services such as Cinder and Glance. The `storage network` uses a dedicated VLAN typically connected to the `br-storage` bridge.

Note

The CIDRs and VLANs listed for each network are examples and may be different in your environment.

Additional VLANs may be required for the following purposes:

- External provider networks for Floating IPs and instances

- Self-service project networks for instances
- Other OpenStack services

Network interfaces

Configuring network interfaces

OpenStack-Ansible does not mandate any specific method of configuring network interfaces on the host. You may choose any tool, such as ifupdown, netplan, systemd-networkd, networkmanager or another operating-system specific tool. The only requirement is that a set of functioning network bridges and interfaces are created which match those expected by OpenStack-Ansible, plus any that you choose to specify for neutron physical interfaces.

A selection of network configuration example files are given in the etc/network and etc/netplan for Ubuntu systems, and in etc/NetworkManager for RHEL-based (or others) systems. It is expected that these will need adjustment for the specific requirements of each deployment.

If you want to delegate management of network bridges and interfaces to OpenStack-Ansible, you can define variables `openstack_hosts_systemd_networkd_devices` and `openstack_hosts_systemd_networkd_networks` in *group_vars/lxc_hosts*, for example:

```
openstack_hosts_systemd_networkd_devices:
  - NetDev:
      Name: vlan-mgmt
      Kind: vlan
    VLAN:
      Id: 10
  - NetDev:
      Name: "{{ management_bridge }}"
      Kind: bridge
    Bridge:
      ForwardDelaySec: 0
      HelloTimeSec: 2
      MaxAgeSec: 12
      STP: off

openstack_hosts_systemd_networkd_networks:
  - interface: "vlan-mgmt"
    bridge: "{{ management_bridge }}"
  - interface: "{{ management_bridge }}"
    address: "{{ management_address }}"
    netmask: "255.255.252.0"
    gateway: "172.29.236.1"
  - interface: "eth0"
    vlan:
      - "vlan-mgmt"
    # NOTE: `05` is prefixed to filename to have precedence over netplan
    filename: 05-lxc-net-eth0
    address: "{{ ansible_facts['eth0']['ipv4']['address'] }}"
    netmask: "{{ ansible_facts['eth0']['ipv4']['netmask'] }}"
```

If you need to run some pre/post hooks for interfaces, you will need to configure a systemd service for

that. It can be done using variable `openstack_hosts_systemd_services`, like that:

```
openstack_hosts_systemd_services:
  - service_name: "{{ management_bridge }}-hook"
    state: started
    enabled: yes
    service_type: oneshot
    execstarts:
      - /bin/bash -c "/bin/echo 'management bridge is available'"
    config_overrides:
      Unit:
        Wants: network-online.target
        After: "{{ sys-subsystem-net-devices-{{ management_bridge }}.device }}
↪"
        BindsTo: "{{ sys-subsystem-net-devices-{{ management_bridge }}.device␣
↪}}"
```

Setting an MTU on a network interface

Larger MTUs can be useful on certain networks, especially storage networks. Add a container_mtu attribute within the `provider_networks` dictionary to set a custom MTU on the container network interfaces that attach to a particular network:

```
provider_networks:
  - network:
      group_binds:
        - glance_api
        - cinder_api
        - cinder_volume
        - nova_compute
      type: "raw"
      container_bridge: "br-storage"
      container_interface: "eth2"
      container_type: "veth"
      container_mtu: "9000"
      ip_from_q: "storage"
      static_routes:
        - cidr: 10.176.0.0/12
          gateway: 172.29.248.1
```

The example above enables jumbo frames by setting the MTU on the storage network to 9000.

Note

Its important to ensure that the MTU is consistently set across the entire network path. This includes not only the container interfaces but also the underlying bridge, physical NICs, and any connected network equipment like switches, routers, and storage devices. Inconsistent MTU settings can lead to fragmentation or dropped packets, which can severely impact performance.

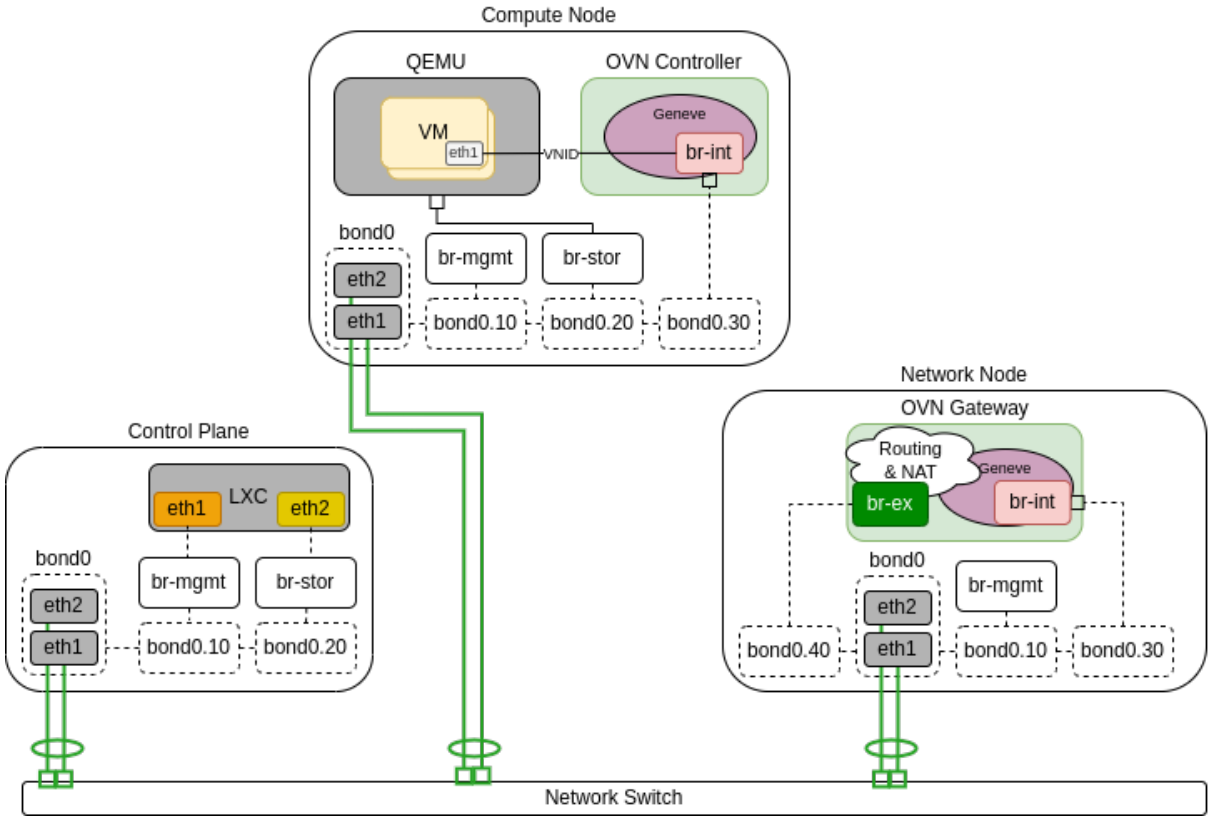
Single interface or bond

OpenStack-Ansible supports the use of a single interface or set of bonded interfaces that carry traffic for OpenStack services as well as instances.

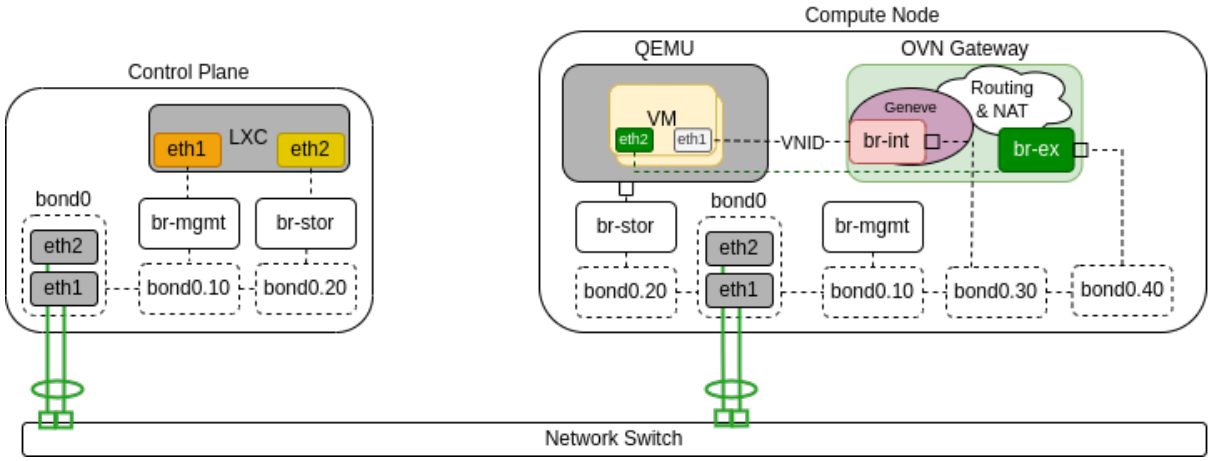
Open Virtual Network (OVN)

The following diagrams demonstrate hosts using a single bond with OVN.

In the scenario below only Network node is connected to external network and computes do not have external connectivity, so routers are needed for external connectivity:

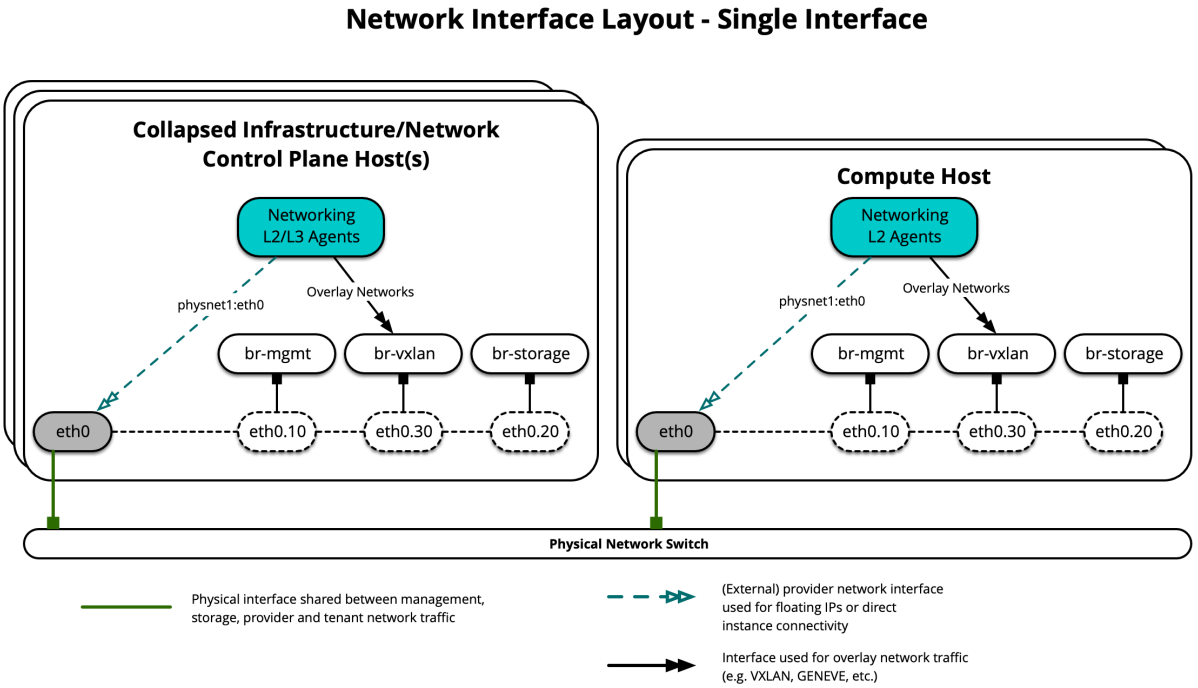


The following diagram demonstrates a compute node serving as an OVN gatway. It is connected to the public network, which enables to connect VMs to public networks not only through routers, but also directly:

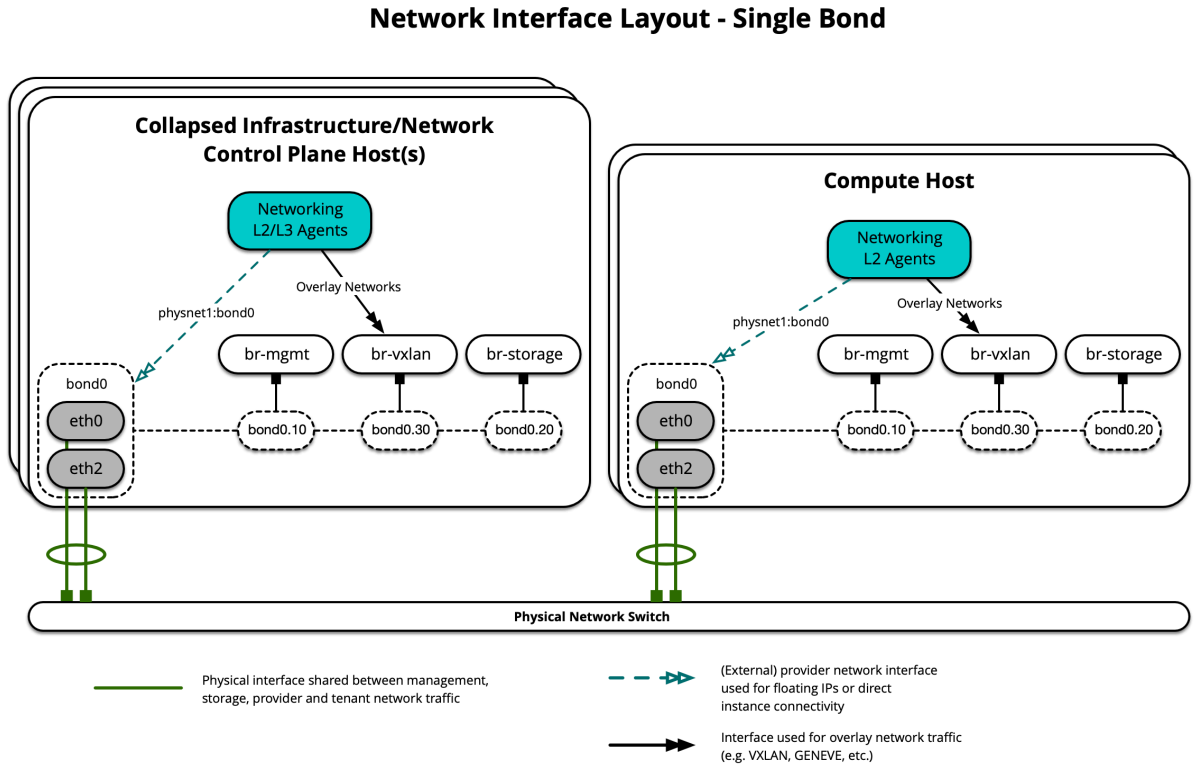


Open vSwitch

The following diagram demonstrates hosts using a single interface for OVS Scenario:



The following diagram demonstrates hosts using a single bond:



Each host will require the correct network bridges to be implemented. The following is the `/etc/network/interfaces` file for `infra1` using a single bond.

Note

If your environment does not have `eth0`, but instead has `p1p1` or some other interface name, ensure that all references to `eth0` in all configuration files are replaced with the appropriate name. The same applies to additional network interfaces.

```
# This is a multi-NIC bonded configuration to implement the required bridges
# for OpenStack-Ansible. This illustrates the configuration of the first
# Infrastructure host and the IP addresses assigned should be adapted
# for implementation on the other hosts.
#
# After implementing this configuration, the host will need to be
# rebooted.

# Assuming that eth0/1 and eth2/3 are dual port NIC's we pair
# eth0 with eth2 for increased resiliency in the case of one interface card
# failing.
auto eth0
iface eth0 inet manual
    bond-master bond0
    bond-primary eth0

auto eth1
iface eth1 inet manual

auto eth2
iface eth2 inet manual
    bond-master bond0

auto eth3
iface eth3 inet manual

# Create a bonded interface. Note that the "bond-slaves" is set to none. This
# is because the bond-master has already been set in the raw interfaces for
# the new bond0.
auto bond0
iface bond0 inet manual
    bond-slaves none
    bond-mode active-backup
    bond-miimon 100
    bond-downdelay 200
    bond-updelay 200

# Container/Host management VLAN interface
auto bond0.10
iface bond0.10 inet manual
    vlan-raw-device bond0

# OpenStack Networking VXLAN (tunnel/overlay) VLAN interface
```

(continues on next page)

(continued from previous page)

```
auto bond0.30
iface bond0.30 inet manual
    vlan-raw-device bond0

# Storage network VLAN interface (optional)
auto bond0.20
iface bond0.20 inet manual
    vlan-raw-device bond0

# Container/Host management bridge
auto br-mgmt
iface br-mgmt inet static
    bridge_stp off
    bridge_waitport 0
    bridge_fd 0
    bridge_ports bond0.10
    address 172.29.236.11
    netmask 255.255.252.0
    gateway 172.29.236.1
    dns-nameservers 8.8.8.8 8.8.4.4

# OpenStack Networking VXLAN (tunnel/overlay) bridge
#
# Nodes hosting Neutron agents must have an IP address on this interface,
# including COMPUTE, NETWORK, and collapsed INFRA/NETWORK nodes.
#

auto br-vxlan
iface br-vxlan inet static
    bridge_stp off
    bridge_waitport 0
    bridge_fd 0
    bridge_ports bond0.30
    address 172.29.240.16
    netmask 255.255.252.0

# OpenStack Networking VLAN bridge
#
# The "br-vlan" bridge is no longer necessary for deployments unless Neutron
# agents are deployed in a container. Instead, a direct interface such as
# bond0 can be specified via the "host_bind_override" override when defining
# provider networks.
#
#auto br-vlan
#iface br-vlan inet manual
#    bridge_stp off
#    bridge_waitport 0
#    bridge_fd 0
#    bridge_ports bond0
```

(continues on next page)

(continued from previous page)

```
# compute1 Network VLAN bridge
#auto br-vlan
#iface br-vlan inet manual
#    bridge_stp off
#    bridge_waitport 0
#    bridge_fd 0
#

# Storage bridge (optional)
#
# Only the COMPUTE and STORAGE nodes must have an IP address
# on this bridge. When used by infrastructure nodes, the
# IP addresses are assigned to containers which use this
# bridge.
#
auto br-storage
iface br-storage inet manual
    bridge_stp off
    bridge_waitport 0
    bridge_fd 0
    bridge_ports bond0.20

# compute1 Storage bridge
#auto br-storage
#iface br-storage inet static
#    bridge_stp off
#    bridge_waitport 0
#    bridge_fd 0
#    bridge_ports bond0.20
#    address 172.29.244.16
#    netmask 255.255.252.0
```

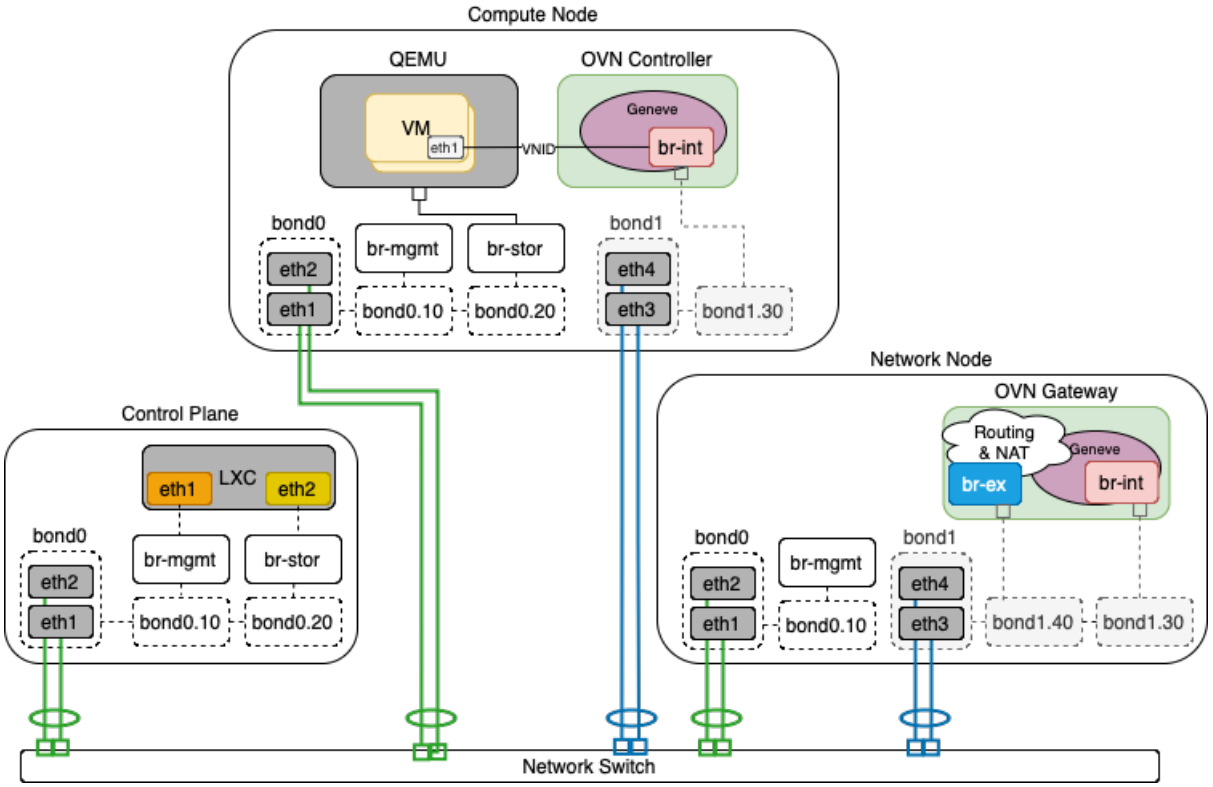
Multiple interfaces or bonds

OpenStack-Ansible supports the use of a multiple interfaces or sets of bonded interfaces that carry traffic for OpenStack services and instances.

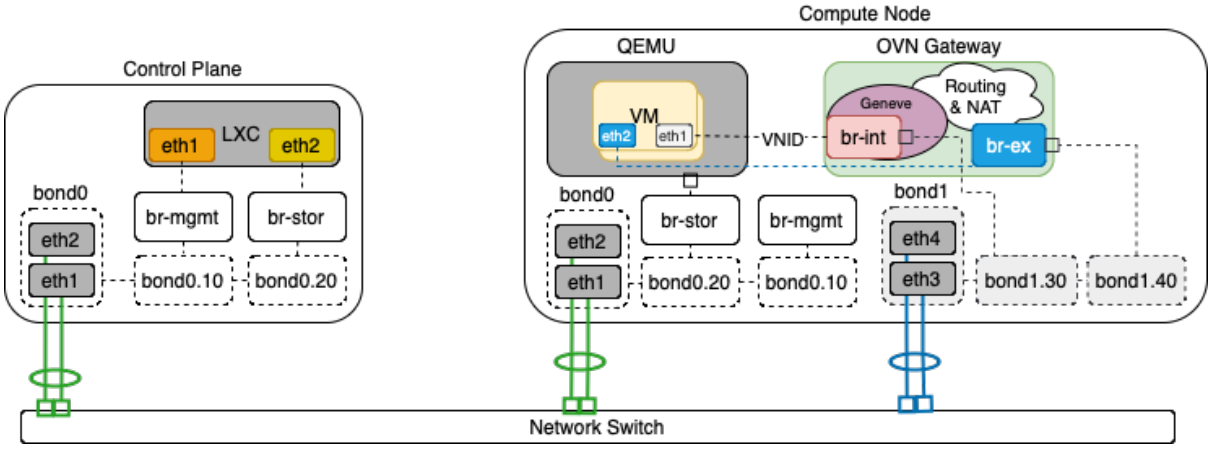
Open Virtual Network (OVN)

The following diagrams demonstrate hosts using multiple bonds with OVN.

In the scenario below only Network node is connected to external network and computes do not have external connectivity, so routers are needed for external connectivity:



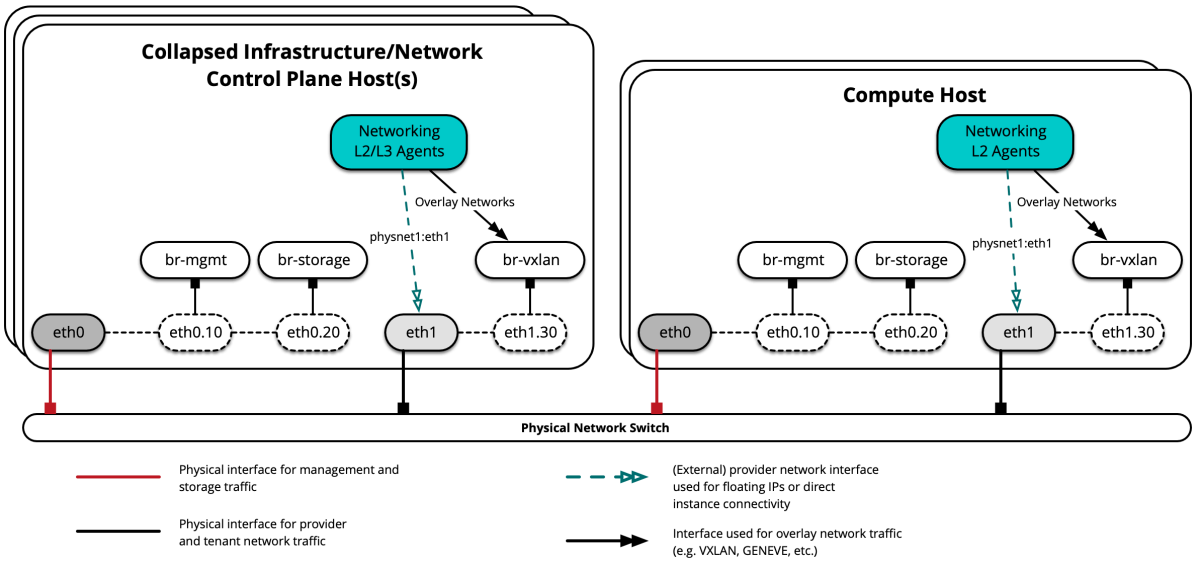
The following diagram demonstrates a compute node serving as an OVN gatway. It is connected to the public network, which enables to connect VMs to public networks not only through routers, but also directly:



Open vSwitch

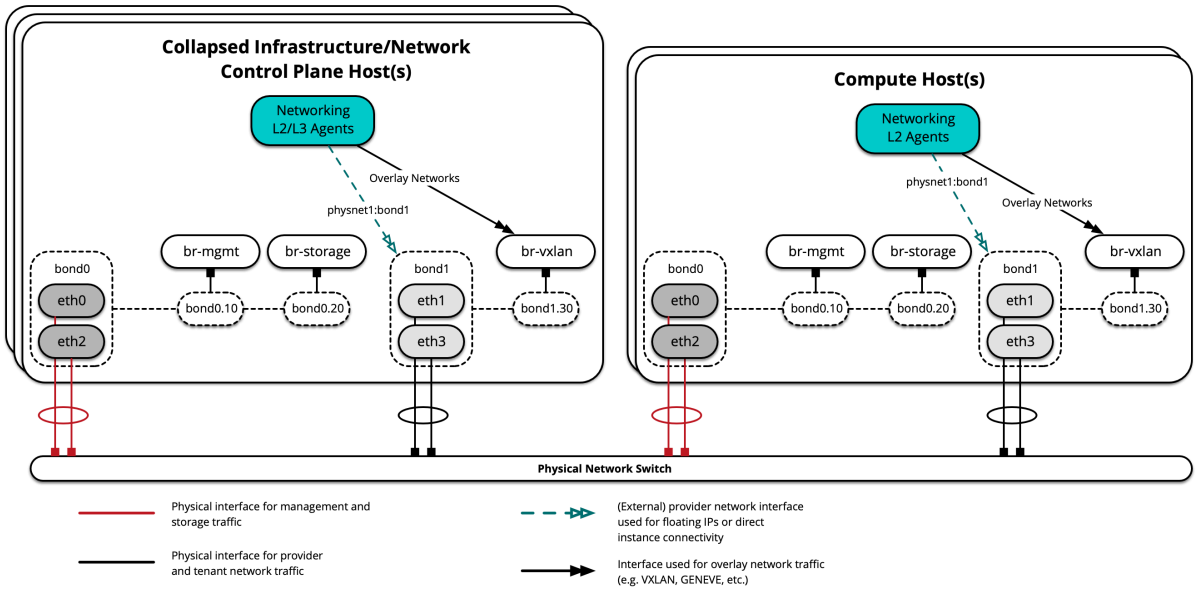
The following diagram demonstrates hosts using multiple interfaces for OVS Scenario:

Network Interface Layout - Multiple Interfaces



The following diagram demonstrates hosts using multiple bonds:

Network Interface Layout - Multiple Bonded Interfaces



Each host will require the correct network bridges to be implemented. The following is the `/etc/network/interfaces` file for `infra1` using multiple bonded interfaces.

Note

If your environment does not have `eth0`, but instead has `p1p1` or some other interface name, ensure that all references to `eth0` in all configuration files are replaced with the appropriate name. The same applies to additional network interfaces.

```
# This is a multi-NIC bonded configuration to implement the required bridges
```

(continues on next page)

(continued from previous page)

```
# for OpenStack-Ansible. This illustrates the configuration of the first
# Infrastructure host and the IP addresses assigned should be adapted
# for implementation on the other hosts.
#
# After implementing this configuration, the host will need to be
# rebooted.

# Assuming that eth0/1 and eth2/3 are dual port NIC's we pair
# eth0 with eth2 and eth1 with eth3 for increased resiliency
# in the case of one interface card failing.
auto eth0
iface eth0 inet manual
    bond-master bond0
    bond-primary eth0

auto eth1
iface eth1 inet manual
    bond-master bond1
    bond-primary eth1

auto eth2
iface eth2 inet manual
    bond-master bond0

auto eth3
iface eth3 inet manual
    bond-master bond1

# Create a bonded interface. Note that the "bond-slaves" is set to none. This
# is because the bond-master has already been set in the raw interfaces for
# the new bond0.
auto bond0
iface bond0 inet manual
    bond-slaves none
    bond-mode active-backup
    bond-miimon 100
    bond-downdelay 200
    bond-updelay 200

# This bond will carry VLAN and VXLAN traffic to ensure isolation from
# control plane traffic on bond0.
auto bond1
iface bond1 inet manual
    bond-slaves none
    bond-mode active-backup
    bond-miimon 100
    bond-downdelay 250
    bond-updelay 250
```

(continues on next page)

(continued from previous page)

```
# Container/Host management VLAN interface
auto bond0.10
iface bond0.10 inet manual
    vlan-raw-device bond0

# OpenStack Networking VXLAN (tunnel/overlay) VLAN interface
auto bond1.30
iface bond1.30 inet manual
    vlan-raw-device bond1

# Storage network VLAN interface (optional)
auto bond0.20
iface bond0.20 inet manual
    vlan-raw-device bond0

# Container/Host management bridge
auto br-mgmt
iface br-mgmt inet static
    bridge_stp off
    bridge_waitport 0
    bridge_fd 0
    bridge_ports bond0.10
    address 172.29.236.11
    netmask 255.255.252.0
    gateway 172.29.236.1
    dns-nameservers 8.8.8.8 8.8.4.4

# OpenStack Networking VXLAN (tunnel/overlay) bridge
#
# Nodes hosting Neutron agents must have an IP address on this interface,
# including COMPUTE, NETWORK, and collapsed INFRA/NETWORK nodes.
#

auto br-vxlan
iface br-vxlan inet static
    bridge_stp off
    bridge_waitport 0
    bridge_fd 0
    bridge_ports bond1.30
    address 172.29.240.16
    netmask 255.255.252.0

# OpenStack Networking VLAN bridge
#
# The "br-vlan" bridge is no longer necessary for deployments unless Neutron
# agents are deployed in a container. Instead, a direct interface such as
# bond1 can be specified via the "host_bind_override" override when defining
# provider networks.
#
```

(continues on next page)

(continued from previous page)

```
#auto br-vlan
#iface br-vlan inet manual
#    bridge_stp off
#    bridge_waitport 0
#    bridge_fd 0
#    bridge_ports bond1

# compute1 Network VLAN bridge
#auto br-vlan
#iface br-vlan inet manual
#    bridge_stp off
#    bridge_waitport 0
#    bridge_fd 0
#

# Storage bridge (optional)
#
# Only the COMPUTE and STORAGE nodes must have an IP address
# on this bridge. When used by infrastructure nodes, the
# IP addresses are assigned to containers which use this
# bridge.
#
auto br-storage
iface br-storage inet manual
    bridge_stp off
    bridge_waitport 0
    bridge_fd 0
    bridge_ports bond0.20

# compute1 Storage bridge
#auto br-storage
#iface br-storage inet static
#    bridge_stp off
#    bridge_waitport 0
#    bridge_fd 0
#    bridge_ports bond0.20
#    address 172.29.244.16
#    netmask 255.255.252.0
```

Additional resources

For more information on how to properly configure network interface files and OpenStack-Ansible configuration files for different deployment scenarios, please refer to the following:

- *Test environment example*
- *Production environment*
- *Provider network groups*

For network agent and container networking toplogies, please refer to the following:

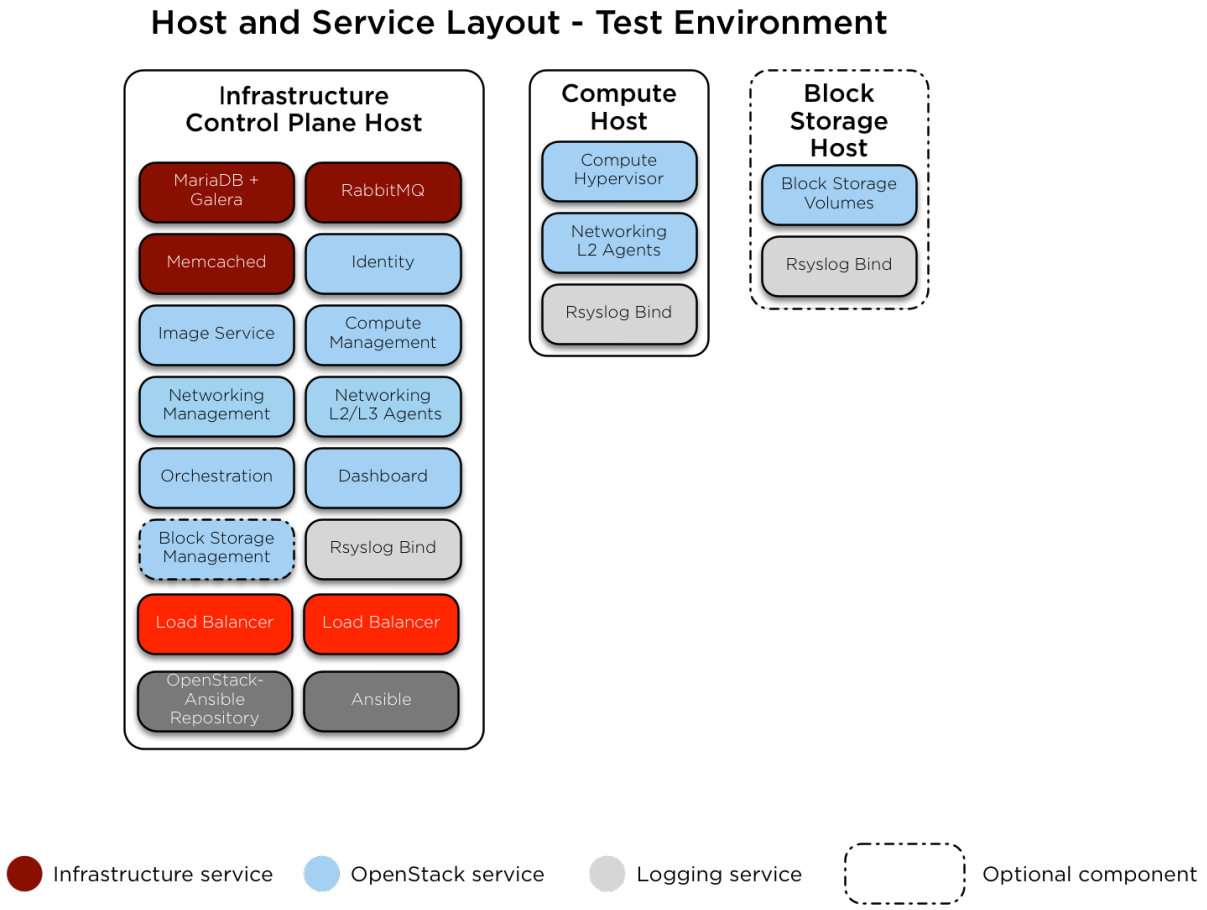
- *Container networking*

Test environment example

Here is an example test environment for a working OpenStack-Ansible (OSA) deployment with a small number of servers.

This example environment has the following characteristics:

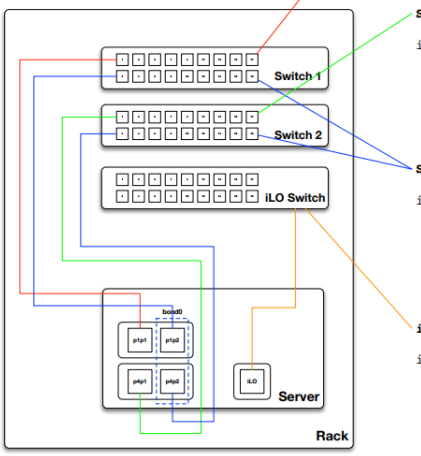
- One infrastructure (control plane) host (8 vCPU, 8 GB RAM, 60 GB HDD)
- One compute host (8 vCPU, 8 GB RAM, 60 GB HDD)
- One Network Interface Card (NIC) for each host
- A basic compute kit environment, with the Image (glance) and Compute (nova) services set to use file-backed storage.
- Internet access via the router address 172.29.236.1 on the Management Network



Network configuration

Switch port configuration

The following example provides a good reference for switch configuration and cab layout. This example may be more that what is required for basic setups however it can be adjusted to just about any configuration. Additionally you will need to adjust the VLANS noted within this example to match your environment.




```
Switch 1 Low Port Configuration

interfaces for p1p1
  switchport mode trunk
  switchport trunk native vlan 500
  switchport trunk allowed vlan 500-502,530-549
  spanning-tree port type edge trunk
  spanning-tree bpduguard enable
  no snmp trap link-status

Switch 2 Low Port Configuration

interfaces for p4p1
  switchport mode trunk
  switchport trunk native vlan 500
  switchport trunk allowed vlan 500-502,530-549
  spanning-tree port type edge trunk
  spanning-tree bpduguard enable
  no snmp trap link-status

Switch 1 and 2 High Port Configuration

interfaces for bond0 (p1p2 and p4p2)
  switchport mode trunk
  switchport trunk allowed vlan 500-502,530-549
  spanning-tree port type edge trunk
  spanning-tree bpduguard enable
  no snmp trap link-status
  channel-group # passive

iLO Switch Configuration

interfaces for ilo
  description ilo
  switchport mode access
  switchport access vlan 501
  spanning-tree port type edge
```

VLAND ID	Name	Subnet	Gateway	Reserved IP Addresses	Internet Access
VLAN 500	pxe	172.24.0.0/22	172.24.0.1	First 20 IP addresses	Yes
VLAN 501	ilo	10.15.240.0/21	10.15.240.1	First 20 IP addresses	No
VLAN 502	mgmt	172.24.8.0/22	172.24.8.1	First 20 IP addresses	Yes
VLAN 530	General	172.24.100.0/22	172.24.100.1	First 20 IP addresses	Yes
VLAN 531	General	172.24.104.0/22	172.24.104.1	First 20 IP addresses	Yes
VLAN 532	General	172.24.108.0/22	172.24.108.1	First 20 IP addresses	Yes
VLAN 533	General	172.24.112.0/22	172.24.112.1	First 20 IP addresses	Yes
VLAN 534	General	172.24.116.0/22	172.24.116.1	First 20 IP addresses	Yes
VLAN 535	General	172.24.120.0/22	172.24.120.1	First 20 IP addresses	Yes
VLAN 536	General	172.24.124.0/22	172.24.124.1	First 20 IP addresses	Yes
VLAN 537	General	172.24.128.0/22	172.24.128.1	First 20 IP addresses	Yes
VLAN 538	General	172.24.132.0/22	172.24.132.1	First 20 IP addresses	Yes
VLAN 539	General	172.24.136.0/22	172.24.136.1	First 20 IP addresses	Yes
VLAN 540	General	172.24.140.0/22	172.24.140.1	First 20 IP addresses	Yes
VLAN 541	General	172.24.144.0/22	172.24.144.1	First 20 IP addresses	Yes
VLAN 542	General	172.24.148.0/22	172.24.148.1	First 20 IP addresses	Yes
VLAN 543	General	172.24.152.0/22	172.24.152.1	First 20 IP addresses	Yes
VLAN 544	General	172.24.156.0/22	172.24.156.1	First 20 IP addresses	Yes
VLAN 545	General	172.24.160.0/22	172.24.160.1	First 20 IP addresses	Yes
VLAN 546	General	172.24.164.0/22	172.24.164.1	First 20 IP addresses	Yes
VLAN 547	General	172.24.168.0/22	172.24.168.1	First 20 IP addresses	Yes
VLAN 548	General	172.24.172.0/22	172.24.172.1	First 20 IP addresses	Yes
VLAN 549	General	172.24.176.0/22	172.24.176.1	First 20 IP addresses	Yes

Network CIDR/VLAN assignments

The following CIDR and VLAN assignments are used for this environment.

Network	CIDR	VLAN
Management Network	172.29.236.0/22	10
Tunnel (VXLAN) Network	172.29.240.0/22	30
Storage Network	172.29.244.0/22	20

IP assignments

The following host name and IP address assignments are used for this environment.

Host name	Management IP	Tunnel (VXLAN) IP	Storage IP
infra1	172.29.236.11	172.29.240.11	
compute1	172.29.236.12	172.29.240.12	172.29.244.12
storage1	172.29.236.13		172.29.244.13

Host network configuration

Each host will require the correct network bridges to be implemented. The following is the `/etc/network/interfaces` file for `infra1`.

> **Note**
>
> If your environment does not have `eth0`, but instead has `p1p1` or some other interface name, ensure that all references to `eth0` in all configuration files are replaced with the appropriate name. The same applies to additional network interfaces.

```
# This is a single-NIC configuration to implement the required bridges
# for OpenStack-Ansible. This illustrates the configuration of the first
# Infrastructure host and the IP addresses assigned should be adapted
# for implementation on the other hosts.
#
# After implementing this configuration, the host will need to be
# rebooted.

# Physical interface
auto eth0
iface eth0 inet manual

# Container/Host management VLAN interface
auto eth0.10
iface eth0.10 inet manual
    vlan-raw-device eth0

# OpenStack Networking VXLAN (tunnel/overlay) VLAN interface
auto eth0.30
iface eth0.30 inet manual
    vlan-raw-device eth0

# Storage network VLAN interface (optional)
auto eth0.20
iface eth0.20 inet manual
    vlan-raw-device eth0

# Container/Host management bridge
auto br-mgmt
iface br-mgmt inet static
    bridge_stp off
    bridge_waitport 0
    bridge_fd 0
    bridge_ports eth0.10
    address 172.29.236.11
    netmask 255.255.252.0
    gateway 172.29.236.1
    dns-nameservers 8.8.8.8 8.8.4.4
```

(continues on next page)

(continued from previous page)

```
# Bind the External VIP
auto br-mgmt:0
iface br-mgmt:0 inet static
    address 172.29.236.10
    netmask 255.255.252.0

# OpenStack Networking VXLAN (tunnel/overlay) bridge
#
# The COMPUTE, NETWORK and INFRA nodes must have an IP address
# on this bridge.
#
auto br-vxlan
iface br-vxlan inet static
    bridge_stp off
    bridge_waitport 0
    bridge_fd 0
    bridge_ports eth0.30
    address 172.29.240.11
    netmask 255.255.252.0

# OpenStack Networking VLAN bridge
auto br-vlan
iface br-vlan inet manual
    bridge_stp off
    bridge_waitport 0
    bridge_fd 0
    bridge_ports eth0

# compute1 Network VLAN bridge
#auto br-vlan
#iface br-vlan inet manual
#    bridge_stp off
#    bridge_waitport 0
#    bridge_fd 0
#
# For tenant vlan support, create a veth pair to be used when the neutron
# agent is not containerized on the compute hosts. 'eth12' is the value used on
# the host_bind_override parameter of the br-vlan network section of the
# openstack_user_config example file. The veth peer name must match the value
# specified on the host_bind_override parameter.
#
# When the neutron agent is containerized it will use the container_interface
# value of the br-vlan network, which is also the same 'eth12' value.
#
# Create veth pair, do not abort if already exists
#    pre-up ip link add br-vlan-veth type veth peer name eth12 || true
# Set both ends UP
#    pre-up ip link set br-vlan-veth up
#    pre-up ip link set eth12 up
```

(continues on next page)

(continued from previous page)

```
# Delete veth pair on DOWN
#    post-down ip link del br-vlan-veth || true
#    bridge_ports eth0 br-vlan-veth

# Storage bridge (optional)
#
# Only the COMPUTE and STORAGE nodes must have an IP address
# on this bridge. When used by infrastructure nodes, the
# IP addresses are assigned to containers which use this
# bridge.
#
auto br-storage
iface br-storage inet manual
    bridge_stp off
    bridge_waitport 0
    bridge_fd 0
    bridge_ports eth0.20

# compute1 Storage bridge
#auto br-storage
#iface br-storage inet static
#    bridge_stp off
#    bridge_waitport 0
#    bridge_fd 0
#    bridge_ports eth0.20
#    address 172.29.244.12
#    netmask 255.255.252.0
```

Deployment configuration

Environment layout

The `/etc/openstack_deploy/openstack_user_config.yml` file defines the environment layout.

The following configuration describes the layout for this environment.

```
---
cidr_networks:
  management: 172.29.236.0/22
  tunnel: 172.29.240.0/22
  storage: 172.29.244.0/22

used_ips:
  - "172.29.236.1,172.29.236.50"
  - "172.29.240.1,172.29.240.50"
  - "172.29.244.1,172.29.244.50"
  - "172.29.248.1,172.29.248.50"

global_overrides:
  # The internal and external VIP should be different IPs, however they
```

(continues on next page)

(continued from previous page)

```
# do not need to be on separate networks.
external_lb_vip_address: 172.29.236.10
internal_lb_vip_address: 172.29.236.11
management_bridge: "br-mgmt"
provider_networks:
  - network:
      container_bridge: "br-mgmt"
      container_type: "veth"
      container_interface: "eth1"
      ip_from_q: "management"
      type: "raw"
      group_binds:
        - all_containers
        - hosts
      is_management_address: true
  - network:
      container_bridge: "br-vxlan"
      container_type: "veth"
      container_interface: "eth10"
      ip_from_q: "tunnel"
      type: "vxlan"
      range: "1:1000"
      net_name: "vxlan"
      group_binds:
        - neutron_openvswitch_agent
  - network:
      container_bridge: "br-vlan"
      container_type: "veth"
      container_interface: "eth12"
      host_bind_override: "eth12"
      type: "flat"
      net_name: "physnet1"
      group_binds:
        - neutron_openvswitch_agent
  - network:
      container_bridge: "br-vlan"
      container_type: "veth"
      container_interface: "eth11"
      type: "vlan"
      range: "101:200,301:400"
      net_name: "physnet2"
      group_binds:
        - neutron_openvswitch_agent
  - network:
      container_bridge: "br-storage"
      container_type: "veth"
      container_interface: "eth2"
      ip_from_q: "storage"
      type: "raw"
```

(continues on next page)

(continued from previous page)

```
        group_binds:
          - glance_api
          - cinder_api
          - cinder_volume
          - nova_compute

###
### Infrastructure
###

# galera, memcache, rabbitmq, utility
shared-infra_hosts:
  infra1:
    ip: 172.29.236.11

# repository (apt cache, python packages, etc)
repo-infra_hosts:
  infra1:
    ip: 172.29.236.11

# load balancer
load_balancer_hosts:
  infra1:
    ip: 172.29.236.11

###
### OpenStack
###

# keystone
identity_hosts:
  infra1:
    ip: 172.29.236.11

# cinder api services
storage-infra_hosts:
  infra1:
    ip: 172.29.236.11

# glance
image_hosts:
  infra1:
    ip: 172.29.236.11

# placement
placement-infra_hosts:
  infra1:
    ip: 172.29.236.11
```

(continues on next page)

(continued from previous page)

```
# nova api, conductor, etc services
compute-infra_hosts:
  infra1:
    ip: 172.29.236.11

# heat
orchestration_hosts:
  infra1:
    ip: 172.29.236.11

# horizon
dashboard_hosts:
  infra1:
    ip: 172.29.236.11

# neutron server, agents (L3, etc)
network_hosts:
  infra1:
    ip: 172.29.236.11

# nova hypervisors
compute_hosts:
  compute1:
    ip: 172.29.236.12

# cinder storage host (LVM-backed)
storage_hosts:
  storage1:
    ip: 172.29.236.13
    container_vars:
      cinder_backends:
        limit_container_types: cinder_volume
        lvm:
          volume_group: cinder-volumes
          volume_driver: cinder.volume.drivers.lvm.LVMVolumeDriver
          volume_backend_name: LVM_iSCSI
          iscsi_ip_address: "172.29.244.13"
```

Environment customizations

The optionally deployed files in `/etc/openstack_deploy/env.d` allow the customization of Ansible groups. This allows the deployer to set whether the services will run in a container (the default), or on the host (on metal).

For this environment you do not need the `/etc/openstack_deploy/env.d` folder as the defaults set by OpenStack-Ansible are suitable.

User variables

The `/etc/openstack_deploy/user_variables.yml` file defines the global overrides for the default variables.

For this environment, if you want to use the same IP address for the internal and external endpoints, you will need to ensure that the internal and public OpenStack endpoints are served with the same protocol. This is done with the following content:

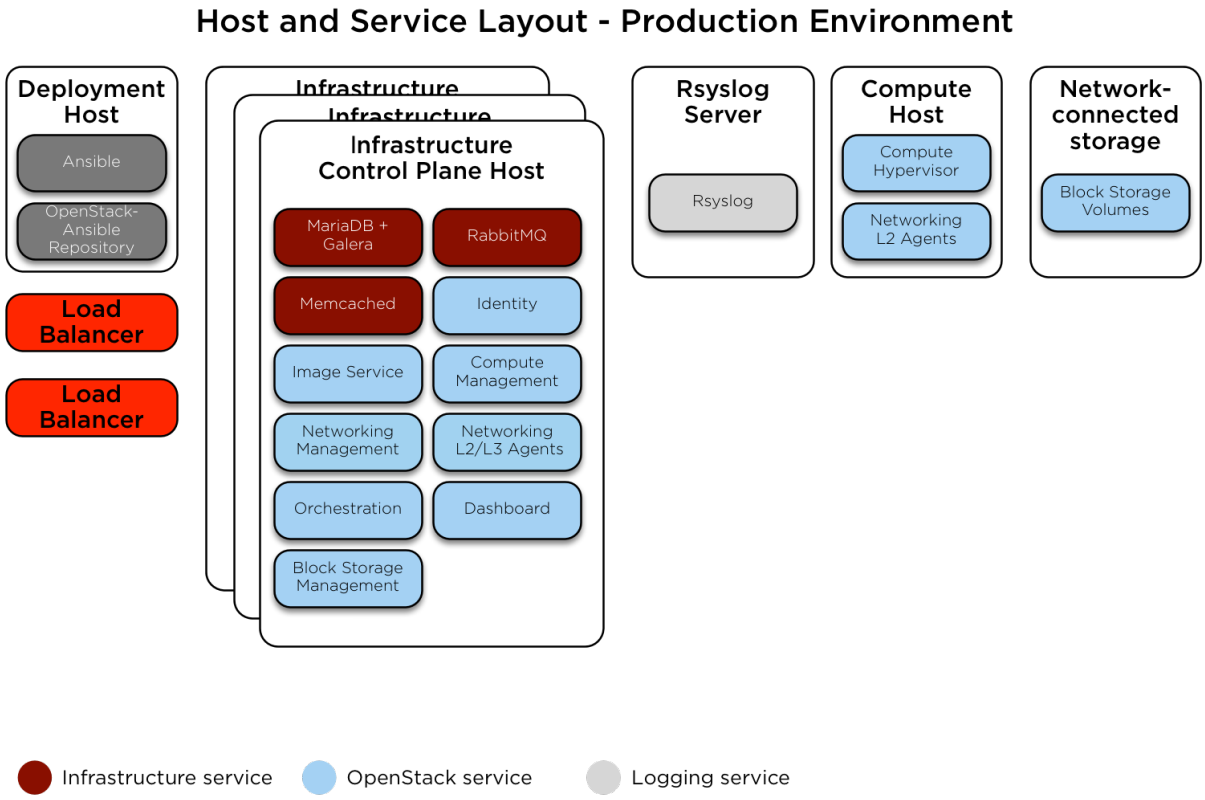
```
---
# This file contains an example of the global variable overrides
# which may need to be set for a production environment.

## OpenStack public endpoint protocol
openstack_service_publicuri_proto: http
```

Production environment

This is an example production environment for a working OpenStack-Ansible (OSA) deployment with high availability services.

This example environment has the following characteristics:

- Three infrastructure (control plane) hosts
- Two compute hosts
- One NFS storage device
- One log aggregation host
- Multiple Network Interface Cards (NIC) configured as bonded pairs for each host
- Full compute kit with the Telemetry service (ceilometer) included, with NFS configured as a storage back end for the Image (glance), and Block Storage (cinder) services
- Internet access via the router address 172.29.236.1 on the Management Network



Network configuration

Network CIDR/VLAN assignments

The following CIDR and VLAN assignments are used for this environment.

Network	CIDR	VLAN
Management Network	172.29.236.0/22	10
Tunnel (VXLAN) Network	172.29.240.0/22	30
Storage Network	172.29.244.0/22	20

IP assignments

The following host name and IP address assignments are used for this environment.

Host name	Management IP	Tunnel (VXLAN) IP	Storage IP
lb_vip_address	172.29.236.9		
infra1	172.29.236.11	172.29.240.11	
infra2	172.29.236.12	172.29.240.12	
infra3	172.29.236.13	172.29.240.13	
log1	172.29.236.14		
NFS Storage			172.29.244.15
compute1	172.29.236.16	172.29.240.16	172.29.244.16
compute2	172.29.236.17	172.29.240.17	172.29.244.17

Host network configuration

Each host will require the correct network bridges to be implemented. The following is the `/etc/network/interfaces` file for `infra1`.

Note

If your environment does not have `eth0`, but instead has `p1p1` or some other interface name, ensure that all references to `eth0` in all configuration files are replaced with the appropriate name. The same applies to additional network interfaces.

```
# This is a multi-NIC bonded configuration to implement the required bridges
# for OpenStack-Ansible. This illustrates the configuration of the first
# Infrastructure host and the IP addresses assigned should be adapted
# for implementation on the other hosts.
#
# After implementing this configuration, the host will need to be
# rebooted.

# Assuming that eth0/1 and eth2/3 are dual port NIC's we pair
# eth0 with eth2 and eth1 with eth3 for increased resiliency
# in the case of one interface card failing.
auto eth0
iface eth0 inet manual
    bond-master bond0
    bond-primary eth0

auto eth1
iface eth1 inet manual
    bond-master bond1
    bond-primary eth1

auto eth2
iface eth2 inet manual
    bond-master bond0

auto eth3
iface eth3 inet manual
    bond-master bond1

# Create a bonded interface. Note that the "bond-slaves" is set to none. This
# is because the bond-master has already been set in the raw interfaces for
# the new bond0.
auto bond0
iface bond0 inet manual
    bond-slaves none
    bond-mode active-backup
    bond-miimon 100
    bond-downdelay 200
    bond-updelay 200
```

(continues on next page)

(continued from previous page)

```
# This bond will carry VLAN and VXLAN traffic to ensure isolation from
# control plane traffic on bond0.
auto bond1
iface bond1 inet manual
    bond-slaves none
    bond-mode active-backup
    bond-miimon 100
    bond-downdelay 250
    bond-updelay 250

# Container/Host management VLAN interface
auto bond0.10
iface bond0.10 inet manual
    vlan-raw-device bond0

# OpenStack Networking VXLAN (tunnel/overlay) VLAN interface
auto bond1.30
iface bond1.30 inet manual
    vlan-raw-device bond1

# Storage network VLAN interface (optional)
auto bond0.20
iface bond0.20 inet manual
    vlan-raw-device bond0

# Container/Host management bridge
auto br-mgmt
iface br-mgmt inet static
    bridge_stp off
    bridge_waitport 0
    bridge_fd 0
    bridge_ports bond0.10
    address 172.29.236.11
    netmask 255.255.252.0
    gateway 172.29.236.1
    dns-nameservers 8.8.8.8 8.8.4.4

# OpenStack Networking VXLAN (tunnel/overlay) bridge
#
# The COMPUTE, NETWORK and INFRA nodes must have an IP address
# on this bridge.
#

auto br-vxlan
iface br-vxlan inet static
    bridge_stp off
    bridge_waitport 0
    bridge_fd 0
```

(continues on next page)

(continued from previous page)

```
    bridge_ports bond1.30
    address 172.29.240.16
    netmask 255.255.252.0

# OpenStack Networking VLAN bridge
auto br-vlan
iface br-vlan inet manual
    bridge_stp off
    bridge_waitport 0
    bridge_fd 0
    bridge_ports bond1

# compute1 Network VLAN bridge
#auto br-vlan
#iface br-vlan inet manual
#    bridge_stp off
#    bridge_waitport 0
#    bridge_fd 0
#
# For tenant vlan support, create a veth pair to be used when the neutron
# agent is not containerized on the compute hosts. 'eth12' is the value used on
# the host_bind_override parameter of the br-vlan network section of the
# openstack_user_config example file. The veth peer name must match the value
# specified on the host_bind_override parameter.
#
# When the neutron agent is containerized it will use the container_interface
# value of the br-vlan network, which is also the same 'eth12' value.
#
# Create veth pair, do not abort if already exists
#    pre-up ip link add br-vlan-veth type veth peer name eth12 || true
# Set both ends UP
#    pre-up ip link set br-vlan-veth up
#    pre-up ip link set eth12 up
# Delete veth pair on DOWN
#    post-down ip link del br-vlan-veth || true
#    bridge_ports bond1 br-vlan-veth

# Storage bridge (optional)
#
# Only the COMPUTE and STORAGE nodes must have an IP address
# on this bridge. When used by infrastructure nodes, the
# IP addresses are assigned to containers which use this
# bridge.
#
auto br-storage
iface br-storage inet manual
    bridge_stp off
    bridge_waitport 0
    bridge_fd 0
```

(continues on next page)

(continued from previous page)

```
    bridge_ports bond0.20

# compute1 Storage bridge
#auto br-storage
#iface br-storage inet static
#    bridge_stp off
#    bridge_waitport 0
#    bridge_fd 0
#    bridge_ports bond0.20
#    address 172.29.244.16
#    netmask 255.255.252.0
```

Deployment configuration

Environment layout

The `/etc/openstack_deploy/openstack_user_config.yml` file defines the environment layout.

The following configuration describes the layout for this environment.

```
---
cidr_networks:
  management: 172.29.236.0/22
  tunnel: 172.29.240.0/22
  storage: 172.29.244.0/22

used_ips:
  - "172.29.236.1,172.29.236.50"
  - "172.29.240.1,172.29.240.50"
  - "172.29.244.1,172.29.244.50"
  - "172.29.248.1,172.29.248.50"

global_overrides:
  internal_lb_vip_address: 172.29.236.9
  #
  # The below domain name must resolve to an IP address
  # in the CIDR specified in haproxy_keepalived_external_vip_cidr.
  # If using different protocols (https/http) for the public/internal
  # endpoints the two addresses must be different.
  #
  external_lb_vip_address: openstack.example.com
  management_bridge: "br-mgmt"
  provider_networks:
    - network:
        container_bridge: "br-mgmt"
        container_type: "veth"
        container_interface: "eth1"
        ip_from_q: "management"
        type: "raw"
        group_binds:
```

(continues on next page)

(continued from previous page)

```
        - all_containers
        - hosts
      is_management_address: true
  - network:
      container_bridge: "br-vxlan"
      container_type: "veth"
      container_interface: "eth10"
      ip_from_q: "tunnel"
      type: "geneve"
      range: "1:1000"
      net_name: "geneve"
      group_binds:
        - neutron_ovn_controller
  - network:
      container_bridge: "br-vlan"
      container_type: "veth"
      container_interface: "eth12"
      host_bind_override: "eth12"
      type: "flat"
      net_name: "physnet1"
      group_binds:
        - neutron_ovn_controller
  - network:
      container_bridge: "br-vlan"
      container_type: "veth"
      container_interface: "eth11"
      type: "vlan"
      range: "101:200,301:400"
      net_name: "physnet2"
      group_binds:
        - neutron_ovn_controller
  - network:
      container_bridge: "br-storage"
      container_type: "veth"
      container_interface: "eth2"
      ip_from_q: "storage"
      type: "raw"
      group_binds:
        - glance_api
        - cinder_api
        - cinder_volume
        - nova_compute

###
### Infrastructure
###

# galera, memcache, rabbitmq, utility
shared-infra_hosts:
```

(continues on next page)

(continued from previous page)

```
  infra1:
    ip: 172.29.236.11
  infra2:
    ip: 172.29.236.12
  infra3:
    ip: 172.29.236.13

# zookeeper
coordination_hosts:
  infra1:
    ip: 172.29.236.11
  infra2:
    ip: 172.29.236.12
  infra3:
    ip: 172.29.236.13

# repository (apt cache, python packages, etc)
repo-infra_hosts:
  infra1:
    ip: 172.29.236.11
  infra2:
    ip: 172.29.236.12
  infra3:
    ip: 172.29.236.13

# load balancer
# Ideally the load balancer should not use the Infrastructure hosts.
# Dedicated hardware is best for improved performance and security.
load_balancer_hosts:
  infra1:
    ip: 172.29.236.11
  infra2:
    ip: 172.29.236.12
  infra3:
    ip: 172.29.236.13

###
### OpenStack
###

# keystone
identity_hosts:
  infra1:
    ip: 172.29.236.11
  infra2:
    ip: 172.29.236.12
  infra3:
    ip: 172.29.236.13
```

(continues on next page)

(continued from previous page)

```
# cinder api services
storage-infra_hosts:
  infra1:
    ip: 172.29.236.11
  infra2:
    ip: 172.29.236.12
  infra3:
    ip: 172.29.236.13

# glance
# The settings here are repeated for each infra host.
# They could instead be applied as global settings in
# user_variables, but are left here to illustrate that
# each container could have different storage targets.
image_hosts:
  infra1:
    ip: 172.29.236.11
    container_vars:
      limit_container_types: glance
      glance_remote_client:
        - what: "172.29.244.15:/images"
          where: "/var/lib/glance/images"
          type: "nfs"
          options: "_netdev,auto"
  infra2:
    ip: 172.29.236.12
    container_vars:
      limit_container_types: glance
      glance_remote_client:
        - what: "172.29.244.15:/images"
          where: "/var/lib/glance/images"
          type: "nfs"
          options: "_netdev,auto"
  infra3:
    ip: 172.29.236.13
    container_vars:
      limit_container_types: glance
      glance_remote_client:
        - what: "172.29.244.15:/images"
          where: "/var/lib/glance/images"
          type: "nfs"
          options: "_netdev,auto"

# placement
placement-infra_hosts:
  infra1:
    ip: 172.29.236.11
  infra2:
    ip: 172.29.236.12
```

(continues on next page)

(continued from previous page)

```
  infra3:
    ip: 172.29.236.13

# nova api, conductor, etc services
compute-infra_hosts:
  infra1:
    ip: 172.29.236.11
  infra2:
    ip: 172.29.236.12
  infra3:
    ip: 172.29.236.13

# heat
orchestration_hosts:
  infra1:
    ip: 172.29.236.11
  infra2:
    ip: 172.29.236.12
  infra3:
    ip: 172.29.236.13

# horizon
dashboard_hosts:
  infra1:
    ip: 172.29.236.11
  infra2:
    ip: 172.29.236.12
  infra3:
    ip: 172.29.236.13

# neutron api
network-infra_hosts:
  infra1:
    ip: 172.29.236.11
  infra2:
    ip: 172.29.236.12
  infra3:
    ip: 172.29.236.13

network-northd_hosts:
  infra1:
    ip: 172.29.236.11
  infra2:
    ip: 172.29.236.12
  infra3:
    ip: 172.29.236.13

# neutron ovn controller
network-gateway_hosts:
```

(continues on next page)

(continued from previous page)

```
  net1:
    ip: 172.29.236.21
  net2:
    ip: 172.29.236.22
  net3:
    ip: 172.29.236.23

# ceilometer (telemetry data collection)
metering-infra_hosts:
  infra1:
    ip: 172.29.236.11
  infra2:
    ip: 172.29.236.12
  infra3:
    ip: 172.29.236.13

# aodh (telemetry alarm service)
metering-alarm_hosts:
  infra1:
    ip: 172.29.236.11
  infra2:
    ip: 172.29.236.12
  infra3:
    ip: 172.29.236.13

# gnocchi (telemetry metrics storage)
metrics_hosts:
  infra1:
    ip: 172.29.236.11
  infra2:
    ip: 172.29.236.12
  infra3:
    ip: 172.29.236.13

# nova hypervisors
compute_hosts:
  compute1:
    ip: 172.29.236.16
  compute2:
    ip: 172.29.236.17

# ceilometer compute agent (telemetry data collection)
metering-compute_hosts:
  compute1:
    ip: 172.29.236.16
  compute2:
    ip: 172.29.236.17

# cinder volume hosts (NFS-backed)
```

(continues on next page)

(continued from previous page)

```
# The settings here are repeated for each infra host.
# They could instead be applied as global settings in
# user_variables, but are left here to illustrate that
# each container could have different storage targets.
storage_hosts:
  infra1:
    ip: 172.29.236.11
    container_vars:
      cinder_backends:
        limit_container_types: cinder_volume
        nfs_volume:
          volume_backend_name: NFS_VOLUME1
          volume_driver: cinder.volume.drivers.nfs.NfsDriver
          nfs_mount_options: "rsize=65535,wsize=65535,timeo=1200,actimeo=120"
          nfs_shares_config: /etc/cinder/nfs_shares
          shares:
            - ip: "172.29.244.15"
              share: "/vol/cinder"
  infra2:
    ip: 172.29.236.12
    container_vars:
      cinder_backends:
        limit_container_types: cinder_volume
        nfs_volume:
          volume_backend_name: NFS_VOLUME1
          volume_driver: cinder.volume.drivers.nfs.NfsDriver
          nfs_mount_options: "rsize=65535,wsize=65535,timeo=1200,actimeo=120"
          nfs_shares_config: /etc/cinder/nfs_shares
          shares:
            - ip: "172.29.244.15"
              share: "/vol/cinder"
  infra3:
    ip: 172.29.236.13
    container_vars:
      cinder_backends:
        limit_container_types: cinder_volume
        nfs_volume:
          volume_backend_name: NFS_VOLUME1
          volume_driver: cinder.volume.drivers.nfs.NfsDriver
          nfs_mount_options: "rsize=65535,wsize=65535,timeo=1200,actimeo=120"
          nfs_shares_config: /etc/cinder/nfs_shares
          shares:
            - ip: "172.29.244.15"
              share: "/vol/cinder"
```

Environment customizations

The optionally deployed files in `/etc/openstack_deploy/env.d` allow the customization of Ansible groups. This allows the deployer to set whether the services will run in a container (the default), or on the host (on metal).

For this environment, the `cinder-volume` runs in a container on the infrastructure hosts. To achieve this, implement `/etc/openstack_deploy/env.d/cinder.yml` with the following content:

```
---
# This file contains an example to show how to set
# the cinder-volume service to run in a container.
#
# Important note:
# When using LVM or any iSCSI-based cinder backends, such as NetApp with
# iSCSI protocol, the cinder-volume service *must* run on metal.
# Reference: https://bugs.launchpad.net/ubuntu/+source/lxc/+bug/1226855

container_skel:
  cinder_volumes_container:
    properties:
      is_metal: false
```

User variables

The `/etc/openstack_deploy/user_variables.yml` file defines the global overrides for the default variables.

For this environment, implement the load balancer on the infrastructure hosts. Ensure that Keepalived is also configured with HAProxy in `/etc/openstack_deploy/user_variables.yml` with the following content.

```
---
# This file contains an example of the global variable overrides
# which may need to be set for a production environment.
# These variables must be defined when external_lb_vip_address or
# internal_lb_vip_address is set to FQDN.
## Load Balancer Configuration (haproxy/keepalived)
haproxy_keepalived_external_vip_cidr: "<external_vip_address>/<netmask>"
haproxy_keepalived_internal_vip_cidr: "172.29.236.9/32"
haproxy_keepalived_external_interface: ens2
haproxy_keepalived_internal_interface: br-mgmt
```

Provider network groups

Many network configuration examples assume a homogenous environment, where each server is configured identically and consistent network interfaces and interface names can be assumed across all hosts.

Recent changes to OpenStack-Ansible (OSA) enables deployers to define provider networks that apply to particular inventory groups and allows for a heterogeneous network configuration within a cloud environment. New groups can be created or existing inventory groups, such as `network_hosts` or `compute_hosts`, can be used to ensure certain configurations are applied only to hosts that meet the given parameters.

Before reading this document, please review the following scenario:

- *Production environment*

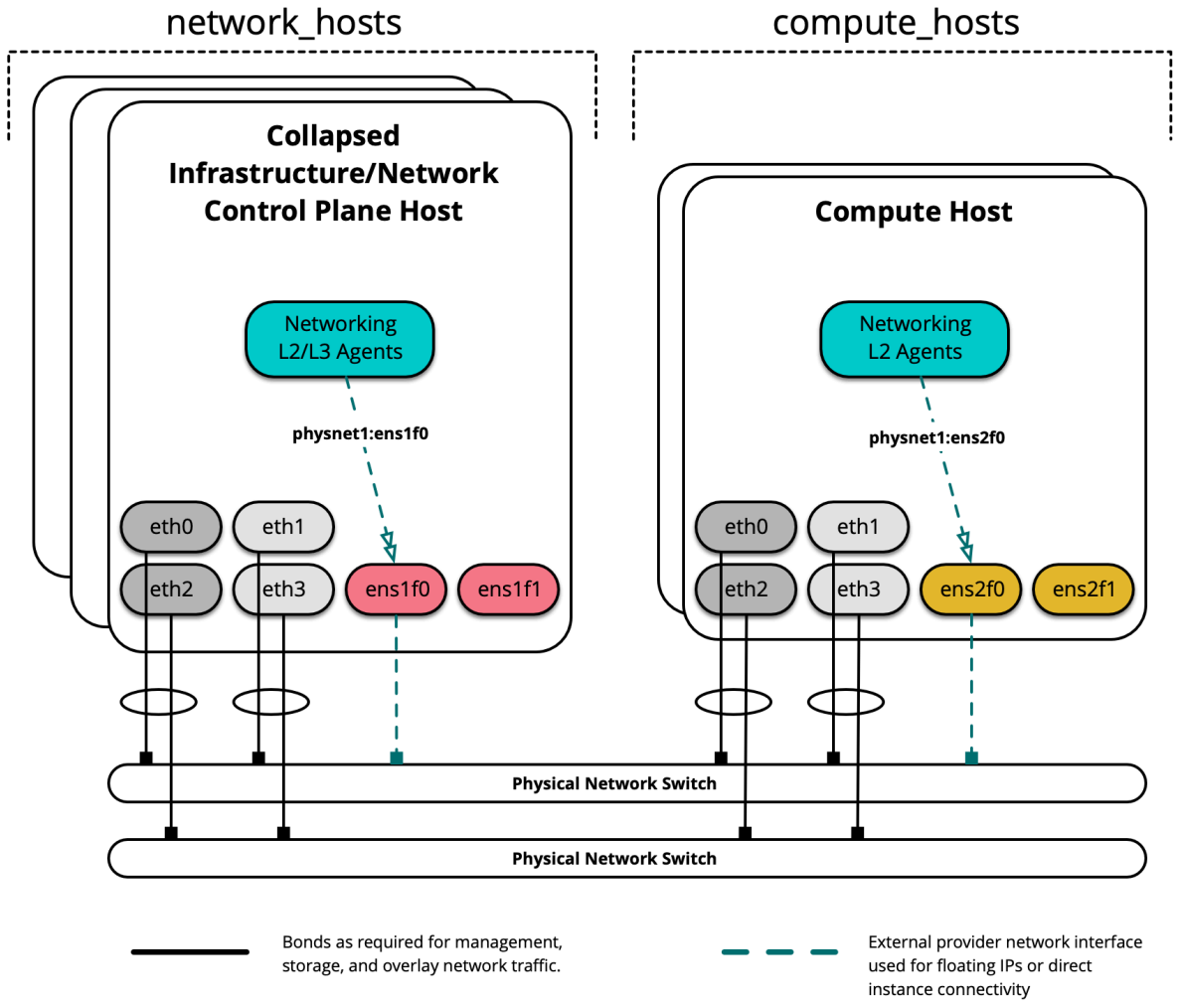
This example environment has the following characteristics:

- A `network_hosts` group consisting of three collapsed infrastructure/network (control plane) hosts
- A `compute_hosts` group consisting of two compute hosts
- Multiple Network Interface Cards (NIC) used as provider network interfaces that vary between hosts

Note

The groups `network_hosts` and `compute_hosts` are pre-defined groups in an OpenStack-Ansible deployment.

The following diagram demonstates servers with different network interface names:



In this example environment, infrastructure/network nodes hosting L2/L3/DHCP agents will utilize an interface named `ens1f0` for the provider network `physnet1`. Compute nodes, on the other hand, will utilize an interface named `ens2f0` for the same `physnet1` provider network.

Note

Differences in network interface names may be the result of a difference in drivers and/or PCI slot locations.

Deployment configuration

Environment layout

The `/etc/openstack_deploy/openstack_user_config.yml` file defines the environment layout.

The following configuration describes the layout for this environment.

```
---
cidr_networks:
  management: 172.29.236.0/22
  tunnel: 172.29.240.0/22
  storage: 172.29.244.0/22

used_ips:
  - "172.29.236.1,172.29.236.50"
  - "172.29.240.1,172.29.240.50"
  - "172.29.244.1,172.29.244.50"
  - "172.29.248.1,172.29.248.50"

global_overrides:
  internal_lb_vip_address: 172.29.236.9
  #
  # The below domain name must resolve to an IP address
  # in the CIDR specified in haproxy_keepalived_external_vip_cidr.
  # If using different protocols (https/http) for the public/internal
  # endpoints the two addresses must be different.
  #
  external_lb_vip_address: openstack.example.com
  management_bridge: "br-mgmt"
  provider_networks:
    - network:
        container_bridge: "br-mgmt"
        container_type: "veth"
        container_interface: "eth1"
        ip_from_q: "management"
        type: "raw"
        group_binds:
          - all_containers
          - hosts
        is_management_address: true
    #
    # The below provider network defines details related to vxlan traffic,
    # including the range of VNIs to assign to project/tenant networks and
    # other attributes.
    #
```

(continues on next page)

(continued from previous page)

```
    # The network details will be used to populate the respective network
    # configuration file(s) on the members of the listed groups.
    #
    - network:
        container_bridge: "br-vxlan"
        container_type: "veth"
        container_interface: "eth10"
        ip_from_q: "tunnel"
        type: "vxlan"
        range: "1:1000"
        net_name: "vxlan"
        group_binds:
          - network_hosts
          - compute_hosts
    #
    # The below provider network(s) define details related to a given provider
    # network: physnet1. Details include the name of the veth interface to
    # connect to the bridge when agent on_metal is False (container_interface)
    # or the physical interface to connect to the bridge when agent on_metal
    # is True (host_bind_override), as well as the network type. The provider
    # network name (net_name) will be used to build a physical network mapping
    # to a network interface; either container_interface or host_bind_override
    # (when defined).
    #
    # The network details will be used to populate the respective network
    # configuration file(s) on the members of the listed groups. In this
    # example, host_bind_override specifies the ens1f0 interface and applies
    # only to the members of network_hosts:
    #
    - network:
        container_bridge: "br-vlan"
        container_type: "veth"
        container_interface: "eth12"
        host_bind_override: "ens1f0"
        type: "flat"
        net_name: "physnet1"
        group_binds:
          - network_hosts
    - network:
        container_bridge: "br-vlan"
        container_type: "veth"
        container_interface: "eth11"
        host_bind_override: "ens1f0"
        type: "vlan"
        range: "101:200,301:400"
        net_name: "physnet2"
        group_binds:
          - network_hosts
    #
```

(continues on next page)

(continued from previous page)

```
    # The below provider network(s) also define details related to the
    # physnet1 provider network. In this example, however, host_bind_override
    # specifies the ens2f0 interface and applies only to the members of
    # compute_hosts:
    #
    - network:
        container_bridge: "br-vlan"
        container_type: "veth"
        container_interface: "eth12"
        host_bind_override: "ens2f0"
        type: "flat"
        net_name: "physnet1"
        group_binds:
          - compute_hosts
    - network:
        container_bridge: "br-vlan"
        container_type: "veth"
        container_interface: "eth11"
        host_bind_override: "ens2f0"
        type: "vlan"
        range: "101:200,301:400"
        net_name: "physnet1"
        group_binds:
          - compute_hosts
    #
    # The below provider network defines details related to storage traffic.
    #
    - network:
        container_bridge: "br-storage"
        container_type: "veth"
        container_interface: "eth2"
        ip_from_q: "storage"
        type: "raw"
        group_binds:
          - glance_api
          - cinder_api
          - cinder_volume
          - nova_compute

###
### Infrastructure
###

# galera, memcache, rabbitmq, utility
shared-infra_hosts:
  infra1:
    ip: 172.29.236.11
  infra2:
    ip: 172.29.236.12
```

(continues on next page)

(continued from previous page)

```
  infra3:
    ip: 172.29.236.13

# repository (apt cache, python packages, etc)
repo-infra_hosts:
  infra1:
    ip: 172.29.236.11
  infra2:
    ip: 172.29.236.12
  infra3:
    ip: 172.29.236.13

# load balancer
# Ideally the load balancer should not use the Infrastructure hosts.
# Dedicated hardware is best for improved performance and security.
load_balancer_hosts:
  infra1:
    ip: 172.29.236.11
  infra2:
    ip: 172.29.236.12
  infra3:
    ip: 172.29.236.13

###
### OpenStack
###

# keystone
identity_hosts:
  infra1:
    ip: 172.29.236.11
  infra2:
    ip: 172.29.236.12
  infra3:
    ip: 172.29.236.13

# cinder api services
storage-infra_hosts:
  infra1:
    ip: 172.29.236.11
  infra2:
    ip: 172.29.236.12
  infra3:
    ip: 172.29.236.13

# glance
# The settings here are repeated for each infra host.
# They could instead be applied as global settings in
# user_variables, but are left here to illustrate that
```

(continues on next page)

(continued from previous page)

```
# each container could have different storage targets.
image_hosts:
  infra1:
    ip: 172.29.236.11
    container_vars:
      limit_container_types: glance
      glance_remote_client:
        - what: "172.29.244.15:/images"
          where: "/var/lib/glance/images"
          type: "nfs"
          options: "_netdev,auto"
  infra2:
    ip: 172.29.236.12
    container_vars:
      limit_container_types: glance
      glance_remote_client:
        - what: "172.29.244.15:/images"
          where: "/var/lib/glance/images"
          type: "nfs"
          options: "_netdev,auto"
  infra3:
    ip: 172.29.236.13
    container_vars:
      limit_container_types: glance
      glance_remote_client:
        - what: "172.29.244.15:/images"
          where: "/var/lib/glance/images"
          type: "nfs"
          options: "_netdev,auto"

# placement
placement-infra_hosts:
  infra1:
    ip: 172.29.236.11
  infra2:
    ip: 172.29.236.12
  infra3:
    ip: 172.29.236.13

# nova api, conductor, etc services
compute-infra_hosts:
  infra1:
    ip: 172.29.236.11
  infra2:
    ip: 172.29.236.12
  infra3:
    ip: 172.29.236.13

# heat
```

(continues on next page)

(continued from previous page)

```
orchestration_hosts:
  infra1:
    ip: 172.29.236.11
  infra2:
    ip: 172.29.236.12
  infra3:
    ip: 172.29.236.13

# horizon
dashboard_hosts:
  infra1:
    ip: 172.29.236.11
  infra2:
    ip: 172.29.236.12
  infra3:
    ip: 172.29.236.13

# neutron server, agents (L3, etc)
network_hosts:
  infra1:
    ip: 172.29.236.11
  infra2:
    ip: 172.29.236.12
  infra3:
    ip: 172.29.236.13

# ceilometer (telemetry data collection)
metering-infra_hosts:
  infra1:
    ip: 172.29.236.11
  infra2:
    ip: 172.29.236.12
  infra3:
    ip: 172.29.236.13

# aodh (telemetry alarm service)
metering-alarm_hosts:
  infra1:
    ip: 172.29.236.11
  infra2:
    ip: 172.29.236.12
  infra3:
    ip: 172.29.236.13

# gnocchi (telemetry metrics storage)
metrics_hosts:
  infra1:
    ip: 172.29.236.11
  infra2:
```

(continues on next page)

(continued from previous page)

```
    ip: 172.29.236.12
  infra3:
    ip: 172.29.236.13

# nova hypervisors
compute_hosts:
  compute1:
    ip: 172.29.236.16
  compute2:
    ip: 172.29.236.17

# ceilometer compute agent (telemetry data collection)
metering-compute_hosts:
  compute1:
    ip: 172.29.236.16
  compute2:
    ip: 172.29.236.17

# cinder volume hosts (NFS-backed)
# The settings here are repeated for each infra host.
# They could instead be applied as global settings in
# user_variables, but are left here to illustrate that
# each container could have different storage targets.
storage_hosts:
  infra1:
    ip: 172.29.236.11
    container_vars:
      cinder_backends:
        limit_container_types: cinder_volume
        nfs_volume:
          volume_backend_name: NFS_VOLUME1
          volume_driver: cinder.volume.drivers.nfs.NfsDriver
          nfs_mount_options: "rsize=65535,wsize=65535,timeo=1200,actimeo=120"
          nfs_shares_config: /etc/cinder/nfs_shares
          shares:
            - ip: "172.29.244.15"
              share: "/vol/cinder"
  infra2:
    ip: 172.29.236.12
    container_vars:
      cinder_backends:
        limit_container_types: cinder_volume
        nfs_volume:
          volume_backend_name: NFS_VOLUME1
          volume_driver: cinder.volume.drivers.nfs.NfsDriver
          nfs_mount_options: "rsize=65535,wsize=65535,timeo=1200,actimeo=120"
          nfs_shares_config: /etc/cinder/nfs_shares
          shares:
            - ip: "172.29.244.15"
```

(continues on next page)

(continued from previous page)

```
            share: "/vol/cinder"
  infra3:
    ip: 172.29.236.13
    container_vars:
      cinder_backends:
        limit_container_types: cinder_volume
        nfs_volume:
          volume_backend_name: NFS_VOLUME1
          volume_driver: cinder.volume.drivers.nfs.NfsDriver
          nfs_mount_options: "rsize=65535,wsize=65535,timeo=1200,actimeo=120"
          nfs_shares_config: /etc/cinder/nfs_shares
          shares:
            - ip: "172.29.244.15"
              share: "/vol/cinder"
```

Hosts in the `network_hosts` group will map `physnet1` to the `ens1f0` interface, while hosts in the `compute_hosts` group will map `physnet1` to the `ens2f0` interface. Additional provider mappings can be established using the same format in a separate definition.

An additional provider interface definition named `physnet2` using different interfaces between hosts may resemble the following:

```
- network:
    container_bridge: "br-vlan2"
    container_type: "veth"
    container_interface: "eth13"
    host_bind_override: "ens1f1"
    type: "vlan"
    range: "2000:2999"
    net_name: "physnet2"
    group_binds:
      - network_hosts
- network:
    container_bridge: "br-vlan2"
    container_type: "veth"
    host_bind_override: "ens2f1"
    type: "vlan"
    range: "2000:2999"
    net_name: "physnet2"
    group_binds:
      - compute_hosts
```

Note

The `container_interface` parameter is only necessary when Neutron agents are run in containers, and can be excluded in many cases. The `container_bridge` and `container_type` parameters also relate to infrastructure containers, but should remain defined for legacy purposes.

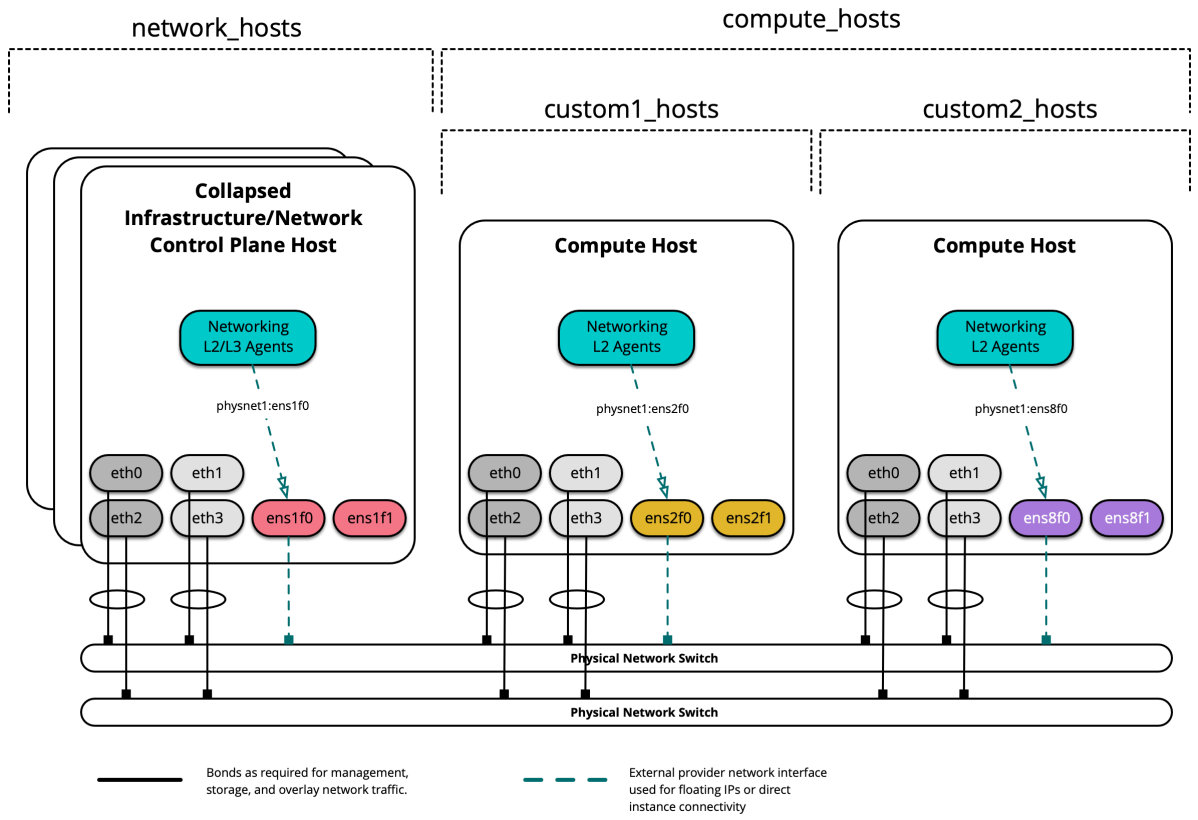
Custom Groups

Custom inventory groups can be created to assist in segmenting hosts beyond the built-in groups provided by OpenStack-Ansible.

Before creating custom groups, please review the following:

- *Configuring the inventory*

The following diagram demonstates how a custom group can be used to further segment hosts:



When creating a custom group, first create a skeleton in `/etc/openstack_deploy/env.d/`. The following is an example of an inventory skeleton for a group named `custom2_hosts` that will consist of bare metal hosts, and has been created at `/etc/openstack_deploy/env.d/custom2_hosts.yml`.

```
---
physical_skel:
  custom2_containers:
    belongs_to:
      - all_containers
  custom2_hosts:
    belongs_to:
      - hosts
```

Define the group and its members in a corresponding file in `/etc/openstack_deploy/conf.d/`. The following is an example of a group named `custom2_hosts` defined in `/etc/openstack_deploy/conf.d/custom2_hosts.yml` consisting of a single member, `compute2`:

```
---
```

(continues on next page)

(continued from previous page)

```
# custom example
custom2_hosts:
  compute2:
    ip: 172.29.236.17
```

The custom group can then be specifed when creating a provider network, as shown here:

```
- network:
    container_bridge: "br-vlan"
    container_type: "veth"
    host_bind_override: "ens8f1"
    type: "vlan"
    range: "101:200,301:400"
    net_name: "physnet1"
    group_binds:
      - custom2_hosts
```

Installing with limited connectivity

Many playbooks and roles in OpenStack-Ansible retrieve dependencies from the public Internet by default. The example configurations assume that the deployer provides a good quality Internet connection via a router on the OpenStack management network.

Deployments may encounter limited external connectivity for a number of reasons:

- Unreliable or low bandwidth external connectivity
- Firewall rules which block external connectivity
- External connectivity required to be via HTTP or SOCKS proxies
- Architectural decisions by the deployer to isolate the OpenStack networks
- High security environments where no external connectivity is permitted

When running OpenStack-Ansible in network environments that block internet connectivity, we recommend the following set of practices and configuration overrides for deployers to use.

The options below are not mutually exclusive and may be combined if desired.

Example internet dependencies

- Python packages
- Distribution specific packages
- LXC container images
- Source code repositories
- GPG keys for package validation

Practice A: Mirror internet resources locally

You may choose to operate and maintain mirrors of OpenStack-Ansible and OpenStack dependencies. Mirrors often provide a great deal of risk mitigation by reducing dependencies on resources and systems outside of your direct control. Mirrors can also provide greater stability, performance and security.

Python package repositories

Many packages used to run OpenStack are installed using *pip*. We advise mirroring the PyPi package index used by *pip*. A deployer can choose to actively mirror the entire upstream PyPi repository, but this may require a significant amount of storage. Alternatively, a caching pip proxy can be used to retain local copies of only those packages which are required.

In order to configure the deployment to use an alternative index, create the file */etc/pip.conf* with the following content and ensure that it resides on all hosts in the environment.

```
[global]
index-url = http://pip.example.org/simple
```

In addition, it is necessary to configure easy_install to use an alternative index. easy_install is used instead of pip to install anything listed under setup_requires in setup.py during wheel builds. See https://pip.pypa.io/en/latest/cli/pip_install/

To configure easy_install to use an alternative index, create the file */root/.pydistutils.cfg* with the following content.

```
[easy_install]
index_url = https://pip.example.org/simple
```

Then, in */etc/openstack_deploy/user_variables.yml*, configure the deployment to copy these files from the host into the container cache image.

```
# Copy these files from the host into the containers
lxc_container_cache_files_from_host:
  - /etc/pip.conf
  - /root/.pydistutils.cfg
```

Distribution specific packages

Many software packages are installed on Ubuntu hosts using *.deb* packages. Similar packaging mechanisms exist for other Linux distributions. We advise mirroring the repositories that host these packages.

Upstream Ubuntu repositories to mirror for Ubuntu 24.04 LTS:

- noble
- noble-updates

OpenStack-Ansible requires several other repositories to install specific components such as Galera and Ceph.

Example repositories to mirror (Ubuntu target hosts):

- https://download.ceph.com
- https://ubuntu-cloud.archive.canonical.com/ubuntu

- https://cloudsmith.io/~rabbitmq/repos/rabbitmq-erlang
- https://cloudsmith.io/~rabbitmq/repos/rabbitmq-server
- https://archive.mariadb.org

These lists are intentionally not exhaustive and equivalents will be required for other Linux distributions. Consult the OpenStack-Ansible playbooks and role documentation for further repositories and the variables that may be used to override the repository location.

LXC container images

OpenStack-Ansible builds LXC images using debootstrap or dnf depending on the distribution. In order to override the package repository you might need to adjust some variables, like `lxc_apt_mirror` or completely override build command with `lxc_hosts_container_build_command` Consult the `openstack-ansible-lxc_hosts` role for details on configuration overrides for this scenario.

Source code repositories

OpenStack-Ansible relies upon Ansible Galaxy to download Ansible roles when bootstrapping a deployment host. Deployers may wish to mirror the dependencies that are downloaded by the `bootstrap-ansible.sh` script.

Deployers can configure the script to source Ansible from an alternate Git repository by setting the environment variable `ANSIBLE_GIT_REPO`. Also, during initial bootstrap you might need to define a custom URL for upper-constraints file that is part of *openstack/requirements* repository, using the TOX_CONSTRAINTS_FILE environment variable.

Deployers can configure the script to source Ansible role dependencies from alternate locations by providing a custom role requirements file and specifying the path to that file using the environment variable `ANSIBLE_ROLE_FILE`.

Practice B: Proxy access to internet resources

Some networks have no routed access to the Internet, or require certain traffic to use application specific gateways such as HTTP or SOCKS proxy servers.

Target and deployment hosts can be configured to reach public internet resources via HTTP or SOCKS proxy server(s). OpenStack-Ansible may be used to configure target hosts to use the proxy server(s). OpenStack-Ansible does not provide automation for creating the proxy server(s).

Initial host deployment is outside the scope of OpenStack-Ansible and the deployer must ensure a minimum set of proxy configuration is in place, in particular for the system package manager.

`apt-get` proxy configuration

See Setting up apt-get to use a http-proxy

Other proxy configuration

In addition to this basic configuration, there are other network clients on the target hosts which may be configured to connect via a proxy. For example:

- Most Python network modules
- *curl*

- *wget*
- *openstack*

These tools and their underlying libraries are used by Ansible itself and the OpenStack-Ansible playbooks, so there must be a proxy configuration in place for the playbooks to successfully access external resources.

Typically these tools read environment variables containing proxy server settings. These environment variables can be configured in `/etc/environment` if required.

It is important to note that the proxy server should only be used to access external resources, and communication between the internal components of the OpenStack deployment should be direct and not through the proxy. The `no_proxy` environment variable is used to specify hosts that should be reached directly without going through the proxy. These often are the hosts in the management network.

OpenStack-Ansible provides two distinct mechanisms for configuring proxy server settings:

1. The default configuration file suggests setting a persistent proxy configuration on all target hosts and defines a persistent `no_proxy` environment variable which lists all hosts/containers management addresses as well as the load balancer internal/external addresses.

2. An alternative method applies proxy configuration in a transient manner during the execution of Ansible playbooks and defines a minimum set of management network IP addresses for `no_proxy` that are required for the playbooks to succeed. These proxy settings do not persist after an Ansible playbook run and the completed deployment does not require them in order to be functional.

The deployer must decide which of these approaches is more suitable for the target hosts, taking into account the following guidance:

1. Persistent proxy configuration is a standard practice and network clients on the target hosts will be able to access external resources after deployment.

2. The deployer must ensure that a persistent proxy configuration has complete coverage of all OpenStack management network host/containers IP addresses in the `no_proxy` environment variable. It is necessary to use a list of IP addresses, CIDR notation is not valid for `no_proxy`.

3. Transient proxy configuration guarantees that proxy environment variables will not persist, ensuring direct communication between services on the OpenStack management network after deployment. Target host network clients such as `wget` will not be able to access external resources after deployment.

4. The maximum length of `no_proxy` should not exceed 1024 characters due to a fixed size buffer in the `pam_env` PAM module. Longer environment variables will be truncated during deployment operations and this will lead to unpredictable errors during or after deployment.

Once the number of hosts/containers in a deployment reaches a certain size, the length of `no_proxy` will exceed 1024 characters at which point it is mandatory to use the transient proxy settings which only requires a subset of the management network IP addresses to be present in `no_proxy` at deployment time.

Refer to *global_environment_variables:* and *deployment_environment_variables:* in the example *user_variables.yml* for details of configuring persistent and transient proxy environment variables.

Deployment host proxy configuration for bootstrapping Ansible

Configure the `bootstrap-ansible.sh` script used to install Ansible and Ansible role dependencies on the deployment host to use a proxy by setting the environment variables `HTTPS_PROXY` or `HTTP_PROXY`.

Note

We recommend you set your `/etc/environment` variables with proxy settings before launching any scripts or playbooks to avoid failure.

For larger or complex environments a dedicated deployment host allows the most suitable proxy configuration to be applied to both deployment and target hosts.

Considerations when proxying TLS traffic

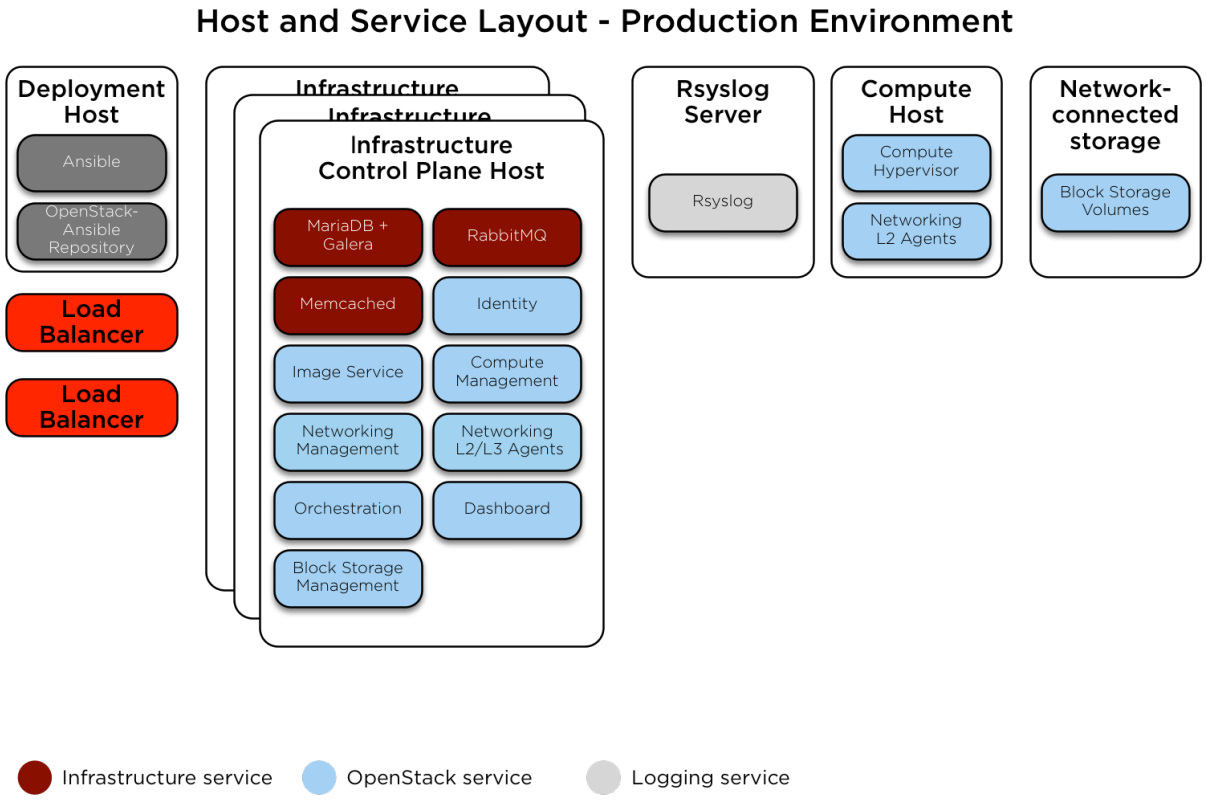
Proxying TLS traffic often interferes with the clients ability to perform successful validation of the certificate chain. Various configuration variables exist within the OpenStack-Ansible playbooks and roles that allow a deployer to ignore these validation failures. Disable certificate chain validation on a case by case basis and only after encountering failures that are known to only be caused by the proxy server(s).

Routed environment example

This section describes an example production environment for a working OpenStack-Ansible (OSA) deployment with high availability services where provider networks and connectivity between physical machines are routed (layer 3).

This example environment has the following characteristics:

- Three infrastructure (control plane) hosts
- Two compute hosts
- One NFS storage device
- One log aggregation host
- Multiple Network Interface Cards (NIC) configured as bonded pairs for each host
- Full compute kit with the Telemetry service (ceilometer) included, with NFS configured as a storage backend for the Image (glance), and Block Storage (cinder) services
- Static routes are added to allow communication between the Management, Tunnel, and Storage Networks of each pod. The gateway address is the first usable address within each networks subnet.



Network configuration

Network CIDR/VLAN assignments

The following CIDR assignments are used for this environment.

Network	CIDR	VLAN
POD 1 Management Network	172.29.236.0/24	10
POD 1 Tunnel (VXLAN) Network	172.29.237.0/24	30
POD 1 Storage Network	172.29.238.0/24	20
POD 2 Management Network	172.29.239.0/24	10
POD 2 Tunnel (VXLAN) Network	172.29.240.0/24	30
POD 2 Storage Network	172.29.241.0/24	20
POD 3 Management Network	172.29.242.0/24	10
POD 3 Tunnel (VXLAN) Network	172.29.243.0/24	30
POD 3 Storage Network	172.29.244.0/24	20
POD 4 Management Network	172.29.245.0/24	10
POD 4 Tunnel (VXLAN) Network	172.29.246.0/24	30
POD 4 Storage Network	172.29.247.0/24	20

IP assignments

The following host name and IP address assignments are used for this environment.

Host name	Management IP	Tunnel (VXLAN) IP	Storage IP
lb_vip_address	172.29.236.9		
infra1	172.29.236.10	172.29.237.10	
infra2	172.29.239.10	172.29.240.10	
infra3	172.29.242.10	172.29.243.10	
log1	172.29.236.11		
NFS Storage			172.29.244.15
compute1	172.29.245.10	172.29.246.10	172.29.247.10
compute2	172.29.245.11	172.29.246.11	172.29.247.11

Host network configuration

Each host will require the correct network bridges to be implemented. The following is the `/etc/network/interfaces` file for `infra1`.

Note

If your environment does not have `eth0`, but instead has `p1p1` or some other interface name, ensure that all references to `eth0` in all configuration files are replaced with the appropriate name. The same applies to additional network interfaces.

```
# This is a multi-NIC bonded configuration to implement the required bridges
# for OpenStack-Ansible. This illustrates the configuration of the first
# Infrastructure host and the IP addresses assigned should be adapted
# for implementation on the other hosts.
#
# After implementing this configuration, the host will need to be
# rebooted.

# Assuming that eth0/1 and eth2/3 are dual port NIC's we pair
# eth0 with eth2 and eth1 with eth3 for increased resiliency
# in the case of one interface card failing.
auto eth0
iface eth0 inet manual
    bond-master bond0
    bond-primary eth0

auto eth1
iface eth1 inet manual
    bond-master bond1
    bond-primary eth1

auto eth2
iface eth2 inet manual
```

(continues on next page)

(continued from previous page)

```
    bond-master bond0

auto eth3
iface eth3 inet manual
    bond-master bond1

# Create a bonded interface. Note that the "bond-slaves" is set to none. This
# is because the bond-master has already been set in the raw interfaces for
# the new bond0.
auto bond0
iface bond0 inet manual
    bond-slaves none
    bond-mode active-backup
    bond-miimon 100
    bond-downdelay 200
    bond-updelay 200

# This bond will carry VLAN and VXLAN traffic to ensure isolation from
# control plane traffic on bond0.
auto bond1
iface bond1 inet manual
    bond-slaves none
    bond-mode active-backup
    bond-miimon 100
    bond-downdelay 250
    bond-updelay 250

# Container/Host management VLAN interface
auto bond0.10
iface bond0.10 inet manual
    vlan-raw-device bond0

# OpenStack Networking VXLAN (tunnel/overlay) VLAN interface
auto bond1.30
iface bond1.30 inet manual
    vlan-raw-device bond1

# Storage network VLAN interface (optional)
auto bond0.20
iface bond0.20 inet manual
    vlan-raw-device bond0

# Container/Host management bridge
auto br-mgmt
iface br-mgmt inet static
    bridge_stp off
    bridge_waitport 0
    bridge_fd 0
    bridge_ports bond0.10
```

(continues on next page)

(continued from previous page)

```
    address 172.29.236.10
    netmask 255.255.255.0
    gateway 172.29.236.1
    dns-nameservers 8.8.8.8 8.8.4.4

# OpenStack Networking VXLAN (tunnel/overlay) bridge
#
# The COMPUTE, NETWORK and INFRA nodes must have an IP address
# on this bridge.
#
auto br-vxlan
iface br-vxlan inet static
    bridge_stp off
    bridge_waitport 0
    bridge_fd 0
    bridge_ports bond1.30
    address 172.29.237.10
    netmask 255.255.252.0

# OpenStack Networking VLAN bridge
auto br-vlan
iface br-vlan inet manual
    bridge_stp off
    bridge_waitport 0
    bridge_fd 0
    bridge_ports bond1

# compute1 Network VLAN bridge
#auto br-vlan
#iface br-vlan inet manual
#    bridge_stp off
#    bridge_waitport 0
#    bridge_fd 0
#
# For tenant vlan support, create a veth pair to be used when the neutron
# agent is not containerized on the compute hosts. 'eth12' is the value used on
# the host_bind_override parameter of the br-vlan network section of the
# openstack_user_config example file. The veth peer name must match the value
# specified on the host_bind_override parameter.
#
# When the neutron agent is containerized it will use the container_interface
# value of the br-vlan network, which is also the same 'eth12' value.
#
# Create veth pair, do not abort if already exists
#    pre-up ip link add br-vlan-veth type veth peer name eth12 || true
# Set both ends UP
#    pre-up ip link set br-vlan-veth up
#    pre-up ip link set eth12 up
# Delete veth pair on DOWN
```

(continues on next page)

(continued from previous page)

```
#    post-down ip link del br-vlan-veth || true
#    bridge_ports bond1 br-vlan-veth

# Storage bridge (optional)
#
# Only the COMPUTE and STORAGE nodes must have an IP address
# on this bridge. When used by infrastructure nodes, the
# IP addresses are assigned to containers which use this
# bridge.
#
auto br-storage
iface br-storage inet manual
    bridge_stp off
    bridge_waitport 0
    bridge_fd 0
    bridge_ports bond0.20

# compute1 Storage bridge
#auto br-storage
#iface br-storage inet static
#    bridge_stp off
#    bridge_waitport 0
#    bridge_fd 0
#    bridge_ports bond0.20
#    address 172.29.247.10
#    netmask 255.255.255.0
```

Deployment configuration

Environment layout

The `/etc/openstack_deploy/openstack_user_config.yml` file defines the environment layout.

For each pod, a group will need to be defined containing all hosts within that pod.

Within defined provider networks, `address_prefix` is used to override the prefix of the key added to each host that contains IP address information. This should usually be one of either `container`, `tunnel`, or `storage`. `reference_group` contains the name of a defined pod group and is used to limit the scope of each provider network to that group.

Static routes are added to allow communication of provider networks between pods.

The following configuration describes the layout for this environment.

```
---
cidr_networks:
  pod1_container: 172.29.236.0/24
  pod2_container: 172.29.237.0/24
  pod3_container: 172.29.238.0/24
  pod4_container: 172.29.239.0/24
  pod1_tunnel: 172.29.240.0/24
```

(continues on next page)

(continued from previous page)

```
  pod2_tunnel: 172.29.241.0/24
  pod3_tunnel: 172.29.242.0/24
  pod4_tunnel: 172.29.243.0/24
  pod1_storage: 172.29.244.0/24
  pod2_storage: 172.29.245.0/24
  pod3_storage: 172.29.246.0/24
  pod4_storage: 172.29.247.0/24

used_ips:
  - "172.29.236.1,172.29.236.50"
  - "172.29.237.1,172.29.237.50"
  - "172.29.238.1,172.29.238.50"
  - "172.29.239.1,172.29.239.50"
  - "172.29.240.1,172.29.240.50"
  - "172.29.241.1,172.29.241.50"
  - "172.29.242.1,172.29.242.50"
  - "172.29.243.1,172.29.243.50"
  - "172.29.244.1,172.29.244.50"
  - "172.29.245.1,172.29.245.50"
  - "172.29.246.1,172.29.246.50"
  - "172.29.247.1,172.29.247.50"

global_overrides:
  #
  # The below domains name must resolve to an IP address
  # in the CIDR specified in haproxy_keepalived_external_vip_cidr and
  # haproxy_keepalived_internal_vip_cidr.
  # If using different protocols (https/http) for the public/internal
  # endpoints the two addresses must be different.
  #
  internal_lb_vip_address: internal-openstack.example.com
  external_lb_vip_address: openstack.example.com
  management_bridge: "br-mgmt"
  provider_networks:
    - network:
        container_bridge: "br-mgmt"
        container_type: "veth"
        container_interface: "eth1"
        ip_from_q: "pod1_container"
        address_prefix: "management"
        type: "raw"
        group_binds:
          - all_containers
          - hosts
        reference_group: "pod1_hosts"
        is_management_address: true
        # Containers in pod1 need routes to the container networks of other␣
↪pods
        static_routes:
```

(continues on next page)

(continued from previous page)

```
        # Route to container networks
        - cidr: 172.29.236.0/22
          gateway: 172.29.236.1
    - network:
        container_bridge: "br-mgmt"
        container_type: "veth"
        container_interface: "eth1"
        ip_from_q: "pod2_container"
        address_prefix: "management"
        type: "raw"
        group_binds:
          - all_containers
          - hosts
        reference_group: "pod2_hosts"
        is_management_address: true
        # Containers in pod2 need routes to the container networks of other␣
↪pods
        static_routes:
          # Route to container networks
          - cidr: 172.29.236.0/22
            gateway: 172.29.237.1
    - network:
        container_bridge: "br-mgmt"
        container_type: "veth"
        container_interface: "eth1"
        ip_from_q: "pod3_container"
        address_prefix: "management"
        type: "raw"
        group_binds:
          - all_containers
          - hosts
        reference_group: "pod3_hosts"
        is_management_address: true
        # Containers in pod3 need routes to the container networks of other␣
↪pods
        static_routes:
          # Route to container networks
          - cidr: 172.29.236.0/22
            gateway: 172.29.238.1
    - network:
        container_bridge: "br-mgmt"
        container_type: "veth"
        container_interface: "eth1"
        ip_from_q: "pod4_container"
        address_prefix: "management"
        type: "raw"
        group_binds:
          - all_containers
          - hosts
```

(continues on next page)

(continued from previous page)

```
        reference_group: "pod4_hosts"
        is_management_address: true
        # Containers in pod4 need routes to the container networks of other␣
↪pods
        static_routes:
          # Route to container networks
          - cidr: 172.29.236.0/22
            gateway: 172.29.239.1
    - network:
        container_bridge: "br-vxlan"
        container_type: "veth"
        container_interface: "eth10"
        ip_from_q: "pod1_tunnel"
        address_prefix: "tunnel"
        type: "vxlan"
        range: "1:1000"
        net_name: "vxlan"
        group_binds:
          - neutron_openvswitch_agent
        reference_group: "pod1_hosts"
        # Containers in pod1 need routes to the tunnel networks of other pods
        static_routes:
          # Route to tunnel networks
          - cidr: 172.29.240.0/22
            gateway: 172.29.240.1
    - network:
        container_bridge: "br-vxlan"
        container_type: "veth"
        container_interface: "eth10"
        ip_from_q: "pod2_tunnel"
        address_prefix: "tunnel"
        type: "vxlan"
        range: "1:1000"
        net_name: "vxlan"
        group_binds:
          - neutron_openvswitch_agent
        reference_group: "pod2_hosts"
        # Containers in pod2 need routes to the tunnel networks of other pods
        static_routes:
          # Route to tunnel networks
          - cidr: 172.29.240.0/22
            gateway: 172.29.241.1
    - network:
        container_bridge: "br-vxlan"
        container_type: "veth"
        container_interface: "eth10"
        ip_from_q: "pod3_tunnel"
        address_prefix: "tunnel"
        type: "vxlan"
```

(continues on next page)

(continued from previous page)

```
      range: "1:1000"
      net_name: "vxlan"
      group_binds:
        - neutron_openvswitch_agent
      reference_group: "pod3_hosts"
      # Containers in pod3 need routes to the tunnel networks of other pods
      static_routes:
        # Route to tunnel networks
        - cidr: 172.29.240.0/22
          gateway: 172.29.242.1
  - network:
      container_bridge: "br-vxlan"
      container_type: "veth"
      container_interface: "eth10"
      ip_from_q: "pod4_tunnel"
      address_prefix: "tunnel"
      type: "vxlan"
      range: "1:1000"
      net_name: "vxlan"
      group_binds:
        - neutron_openvswitch_agent
      reference_group: "pod4_hosts"
      # Containers in pod4 need routes to the tunnel networks of other pods
      static_routes:
        # Route to tunnel networks
        - cidr: 172.29.240.0/22
          gateway: 172.29.243.1
  - network:
      container_bridge: "br-vlan"
      container_type: "veth"
      container_interface: "eth12"
      host_bind_override: "eth12"
      type: "flat"
      net_name: "physnet2"
      group_binds:
        - neutron_openvswitch_agent
  - network:
      container_bridge: "br-vlan"
      container_type: "veth"
      container_interface: "eth11"
      type: "vlan"
      range: "101:200,301:400"
      net_name: "physnet1"
      group_binds:
        - neutron_openvswitch_agent
  - network:
      container_bridge: "br-storage"
      container_type: "veth"
      container_interface: "eth2"
```

(continues on next page)

(continued from previous page)

```
      ip_from_q: "pod1_storage"
      address_prefix: "storage"
      type: "raw"
      group_binds:
        - glance_api
        - cinder_api
        - cinder_volume
        - nova_compute
      reference_group: "pod1_hosts"
      # Containers in pod1 need routes to the storage networks of other pods
      static_routes:
        # Route to storage networks
        - cidr: 172.29.244.0/22
          gateway: 172.29.244.1
    - network:
      container_bridge: "br-storage"
      container_type: "veth"
      container_interface: "eth2"
      ip_from_q: "pod2_storage"
      address_prefix: "storage"
      type: "raw"
      group_binds:
        - glance_api
        - cinder_api
        - cinder_volume
        - nova_compute
      reference_group: "pod2_hosts"
      # Containers in pod2 need routes to the storage networks of other pods
      static_routes:
        # Route to storage networks
        - cidr: 172.29.244.0/22
          gateway: 172.29.245.1
    - network:
      container_bridge: "br-storage"
      container_type: "veth"
      container_interface: "eth2"
      ip_from_q: "pod3_storage"
      address_prefix: "storage"
      type: "raw"
      group_binds:
        - glance_api
        - cinder_api
        - cinder_volume
        - nova_compute
      reference_group: "pod3_hosts"
      # Containers in pod3 need routes to the storage networks of other pods
      static_routes:
        # Route to storage networks
        - cidr: 172.29.244.0/22
```

(continues on next page)

(continued from previous page)

```
            gateway: 172.29.246.1
    - network:
        container_bridge: "br-storage"
        container_type: "veth"
        container_interface: "eth2"
        ip_from_q: "pod4_storage"
        address_prefix: "storage"
        type: "raw"
        group_binds:
          - glance_api
          - cinder_api
          - cinder_volume
          - nova_compute
        reference_group: "pod4_hosts"
        # Containers in pod4 need routes to the storage networks of other pods
        static_routes:
          # Route to storage networks
          - cidr: 172.29.244.0/22
            gateway: 172.29.247.1

###
### Infrastructure
###

pod1_hosts: &pod1
  infra1:
    ip: 172.29.236.10

pod2_hosts: &pod2
  infra2:
    ip: 172.29.239.10

pod3_hosts: &pod3
  infra3:
    ip: 172.29.242.10

pod4_hosts: &pod4
  compute1:
    ip: 172.29.245.10
  compute2:
    ip: 172.29.245.11

# galera, memcache, rabbitmq, utility
shared-infra_hosts: &controllers
  <<: [*pod1, *pod2, *pod3]

# repository (apt cache, python packages, etc)
repo-infra_hosts: *controllers
```

(continues on next page)

(continued from previous page)

```
# load balancer
# Ideally the load balancer should not use the Infrastructure hosts.
# Dedicated hardware is best for improved performance and security.
load_balancer_hosts: *controllers

###
### OpenStack
###

# keystone
identity_hosts: *controllers

# cinder api services
storage-infra_hosts: *controllers

# glance
# The settings here are repeated for each infra host.
# They could instead be applied as global settings in
# user_variables, but are left here to illustrate that
# each container could have different storage targets.
image_hosts:
  infra1:
    ip: 172.29.236.11
    container_vars:
      limit_container_types: glance
      glance_remote_client:
        - what: "172.29.244.15:/images"
          where: "/var/lib/glance/images"
          type: "nfs"
          options: "_netdev,auto"
  infra2:
    ip: 172.29.236.12
    container_vars:
      limit_container_types: glance
      glance_remote_client:
        - what: "172.29.244.15:/images"
          where: "/var/lib/glance/images"
          type: "nfs"
          options: "_netdev,auto"
  infra3:
    ip: 172.29.236.13
    container_vars:
      limit_container_types: glance
      glance_remote_client:
        - what: "172.29.244.15:/images"
          where: "/var/lib/glance/images"
          type: "nfs"
          options: "_netdev,auto"
```

(continues on next page)

(continued from previous page)

```
# nova api, conductor, etc services
compute-infra_hosts: *controllers

# heat
orchestration_hosts: *controllers

# horizon
dashboard_hosts: *controllers

# neutron server, agents (L3, etc)
network_hosts: *controllers

# ceilometer (telemetry data collection)
metering-infra_hosts: *controllers

# aodh (telemetry alarm service)
metering-alarm_hosts: *controllers
# gnocchi (telemetry metrics storage)
metrics_hosts: *controllers

# nova hypervisors
compute_hosts: *pod4

# ceilometer compute agent (telemetry data collection)
metering-compute_hosts: *pod4

# cinder volume hosts (NFS-backed)
# The settings here are repeated for each infra host.
# They could instead be applied as global settings in
# user_variables, but are left here to illustrate that
# each container could have different storage targets.
storage_hosts:
  infra1:
    ip: 172.29.236.11
    container_vars:
      cinder_backends:
        limit_container_types: cinder_volume
        nfs_volume:
          volume_backend_name: NFS_VOLUME1
          volume_driver: cinder.volume.drivers.nfs.NfsDriver
          nfs_mount_options: "rsize=65535,wsize=65535,timeo=1200,actimeo=120"
          nfs_shares_config: /etc/cinder/nfs_shares
          shares:
            - ip: "172.29.244.15"
              share: "/vol/cinder"
  infra2:
    ip: 172.29.236.12
    container_vars:
      cinder_backends:
```

(continues on next page)

(continued from previous page)

```
      limit_container_types: cinder_volume
      nfs_volume:
        volume_backend_name: NFS_VOLUME1
        volume_driver: cinder.volume.drivers.nfs.NfsDriver
        nfs_mount_options: "rsize=65535,wsize=65535,timeo=1200,actimeo=120"
        nfs_shares_config: /etc/cinder/nfs_shares
        shares:
          - ip: "172.29.244.15"
            share: "/vol/cinder"
  infra3:
    ip: 172.29.236.13
    container_vars:
      cinder_backends:
        limit_container_types: cinder_volume
        nfs_volume:
          volume_backend_name: NFS_VOLUME1
          volume_driver: cinder.volume.drivers.nfs.NfsDriver
          nfs_mount_options: "rsize=65535,wsize=65535,timeo=1200,actimeo=120"
          nfs_shares_config: /etc/cinder/nfs_shares
          shares:
            - ip: "172.29.244.15"
              share: "/vol/cinder"
```

Environment customizations

The optionally deployed files in `/etc/openstack_deploy/env.d` allow the customization of Ansible groups. This allows the deployer to set whether the services will run in a container (the default), or on the host (on metal).

For this environment, the `cinder-volume` runs in a container on the infrastructure hosts. To achieve this, implement `/etc/openstack_deploy/env.d/cinder.yml` with the following content:

```
---
# This file contains an example to show how to set
# the cinder-volume service to run in a container.
#
# Important note:
# When using LVM or any iSCSI-based cinder backends, such as NetApp with
# iSCSI protocol, the cinder-volume service *must* run on metal.
# Reference: https://bugs.launchpad.net/ubuntu/+source/lxc/+bug/1226855

container_skel:
  cinder_volumes_container:
    properties:
      is_metal: false
```

You can also declare a custom group for each pod that will also include all containers from hosts that belong to this pod. This might be handy if you want to define some variable for all hosts in the pod using group_variables.

For that create `/etc/openstack_deploy/env.d/pod.yml` with the following content:

```
---

component_skel:
  pod1_containers:
    belongs_to:
      - pod1_all
  pod1_hosts:
    belongs_to:
      - pod1_all

  pod2_containers:
    belongs_to:
      - pod2_all
  pod2_hosts:
    belongs_to:
      - pod2_all

  pod3_containers:
    belongs_to:
      - pod3_all
  pod3_hosts:
    belongs_to:
      - pod3_all

  pod4_containers:
    belongs_to:
      - pod3_all
  pod4_hosts:
    belongs_to:
      - pod3_all

container_skel:
  pod1_containers:
    properties:
      is_nest: true
  pod2_containers:
    properties:
      is_nest: true
  pod3_containers:
    properties:
      is_nest: true
  pod4_containers:
    properties:
      is_nest: true
```

Above example will create following groups:

- `podN_hosts` which will contain only bare metal nodes
- **`podN_containers` that will contain all containers that are spawned on** bare metal nodes, that are part of the pod.

- `podN_all` that will contain *podN_hosts* and *podN_containers* members

User variables

The `/etc/openstack_deploy/user_variables.yml` file defines the global overrides for the default variables.

For this environment, implement the load balancer on the infrastructure hosts. Ensure that Keepalived is also configured with HAProxy in `/etc/openstack_deploy/user_variables.yml` with the following content.

```
---
# This file contains an example of the global variable overrides
# which may need to be set for a production environment.
# These variables must be defined when external_lb_vip_address or
# internal_lb_vip_address is set to FQDN.
## Load Balancer Configuration (haproxy/keepalived)
haproxy_keepalived_external_vip_cidr: "<external_vip_address>/<netmask>"
haproxy_keepalived_internal_vip_cidr: "172.29.236.9/32"
haproxy_keepalived_external_interface: ens2
haproxy_keepalived_internal_interface: br-mgmt
```

Example of multi-AZ environment configuration

On this page, we will provide an example configuration that can be used in production environments with multiple Availability Zones.

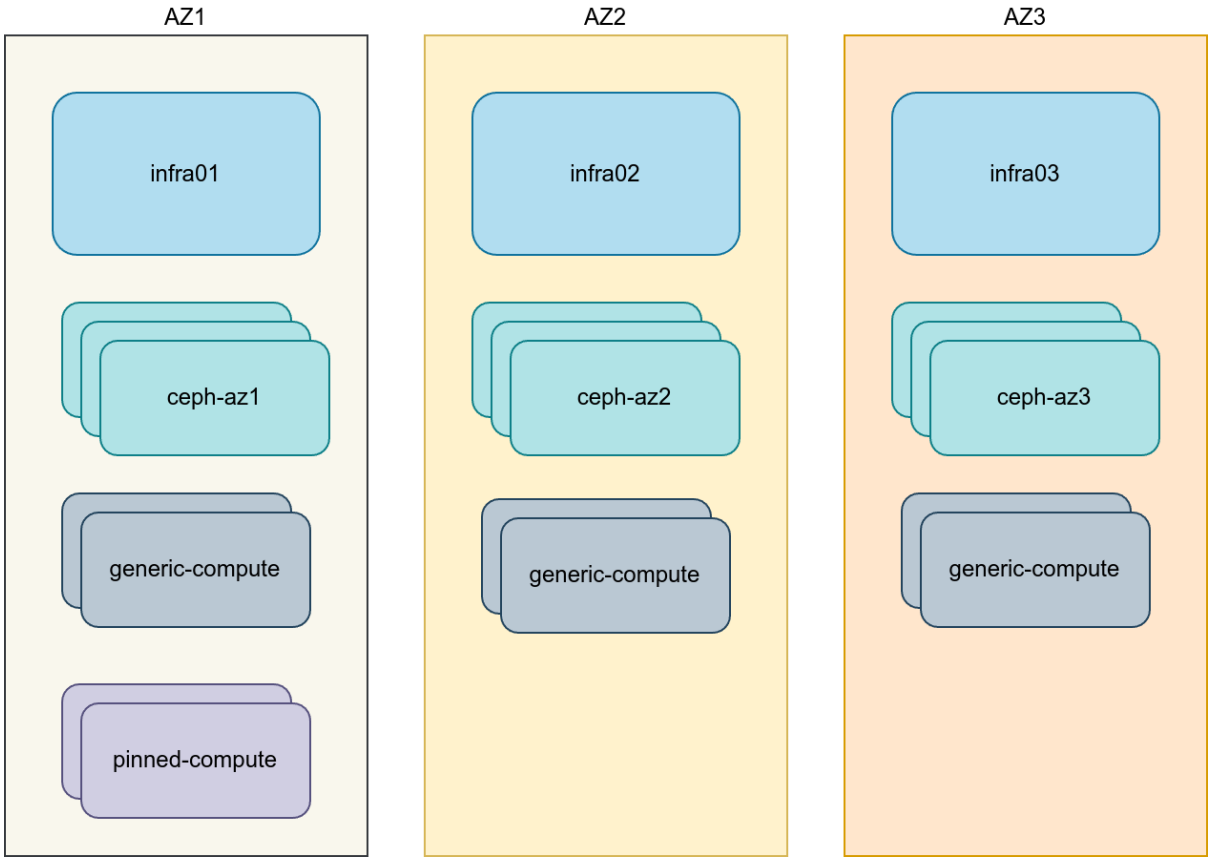
It will be an extended and more specific version of *Routed environment example* so it is expected that you are aware of the concepts and approaches defined there.

To better understand why some configuration options were applied in examples it is also recommended to look through *Configuring the inventory*

Generic design

The following design decisions were made in the example below:

- Three Availability Zones (AZs)
- Three infrastructure (control plane) hosts, each host is placed in a different Availability Zone
- Eight compute hosts, 2 compute hosts in each Availability Zone. First Availability Zone has two extra compute hosts for pinned CPU aggregate.
- Three Ceph storage clusters provisioned with Ceph Ansible.
- Compute hosts act as OVN gateway hosts
- Tunnel networks which are reachable between Availability Zones
- Public API, OpenStack external and management networks are represented as stretched L2 networks between Availability Zones.



Load Balancing

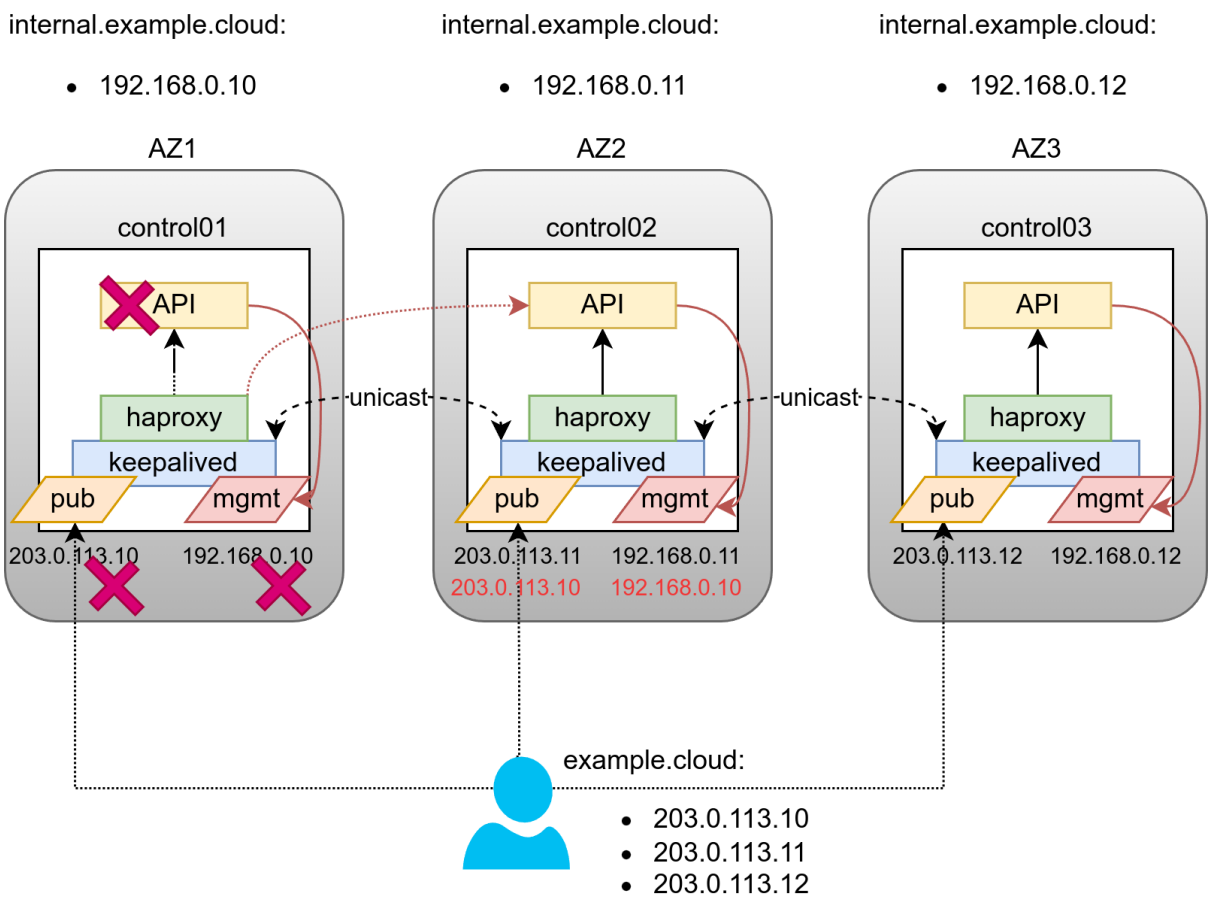
A Load Balancer (HAProxy) is usually deployed on infrastructure hosts. With infrastructure hosts being spread across Availability Zones we need to come up with a more complex design which is aimed at solving the following issues:

- Withstand a single Availability Zone failure
- Reduce amount of cross-AZ traffic
- Spread load across Availability Zones

To address these challenges, the following changes to the basic design were made:

- Leverage DNS Round Robin (an A/AAAA record per AZ) for Public API
- Define Internal API FQDN through /etc/hosts overrides, which are unique per Availability Zone
- Define 6 Keepalived instances: 3 for public and 3 for internal Virtual IP addresses (VIPs)
- Ensure HAProxy to prioritize a backend from own Availability Zone over remote ones

The example also deploys HAProxy with Keepalived in their own LXC containers on the contrary to a more conventional bare metal deployment. You can check a *HAProxy and Keepalived in LXC containers* for more details on how to do that.



Storage design complications

There are multiple complications related to organizing storage where the storage is not stretched between Availability Zones.

First, there is only a single controller in any given Availability Zone, while multiple copies of `cinder_volume` needs to be run for each storage provider for High Availability. As `cinder_volume` needs access to storage network, one of the best places for it are `ceph-mon` hosts.

Another challenge is to organize shared storage for Glance Images, as `rbd` cant be used consistently anymore. While Glance Interoperable Import interface could be leveraged for syncing images between `rbd` backends, in fact not all clients and services can work with Glances import API. One of the most obvious solutions here can be usage of Swift API, while configuring Ceph RadosGW policy to replicate the bucket between independent instances located in their Availability Zones.

Last, but not the least complication is Nova scheduling when `cross_az_attach` is disabled. As Nova will not add an Availability Zone to instances `request_specs` when an instance is created from a volume directly, on the contrary to creating volume manually in advance and supplying volume UUID to the instance create API call. The problem with that behavior, is that Nova will attempt to Live Migrate or re-schedule instances without an Availability Zone in `request_specs` to other AZs, which will result in failure, as `cross_az_attach` is disabled. You can read more about this in a Nova bug report In order to work around this Bug you need to set a `default_schedule_zone` for Nova and Cinder, which will ensure AZ always being defined in `request_specs`. You can also go further and define an actual Availability Zone as `default_schedule_zone`, making each controller to have its own default. As Load Balancer will attempt to send requests only to local backends first, this approach does work to distribute

new VMs across all AZs when user does not supply AZ explicitly. Otherwise, the default AZ will be accepting significantly more new signups.

Configuration examples

Network configuration

Network CIDR/VLAN assignments

The following CIDR assignments are used for this environment.

Network	CIDR	VLAN
Management Network	172.29.236.0/22	10
AZ1 Storage Network	172.29.244.0/24	20
AZ1 Tunnel (Geneve) Network	172.29.240.0/24	30
AZ2 Storage Network	172.29.245.0/24	21
AZ2 Tunnel (Geneve) Network	172.29.241.0/24	31
AZ3 Storage Network	172.29.246.0/24	22
AZ3 Tunnel (Geneve) Network	172.29.242.0/24	32
Public API VIPs	203.0.113.0/28	400

IP assignments

The following host name and IP address assignments are used for this environment.

Host name	Management IP	Tunnel (Geneve) IP	Storage IP
infra1	172.29.236.11		
infra2	172.29.236.12		
infra3	172.29.236.13		
az1_ceph1	172.29.237.201		172.29.244.201
az1_ceph2	172.29.237.202		172.29.244.202
az1_ceph3	172.29.237.203		172.29.244.203
az2_ceph1	172.29.238.201		172.29.245.201
az2_ceph2	172.29.238.202		172.29.245.202
az2_ceph3	172.29.238.203		172.29.245.203
az3_ceph1	172.29.239.201		172.29.246.201
az3_ceph2	172.29.239.202		172.29.246.202
az3_ceph3	172.29.239.203		172.29.246.203
az1_compute1	172.29.237.11	172.29.240.11	172.29.244.11
az1_compute2	172.29.237.12	172.29.240.12	172.29.244.12
az1_pin_compute1	172.29.237.13	172.29.240.13	172.29.244.13
az1_pin_compute2	172.29.237.14	172.29.240.14	172.29.244.14
az2_compute1	172.29.238.11	172.29.241.11	172.29.245.11
az2_compute2	172.29.238.12	172.29.241.12	172.29.245.12
az3_compute1	172.29.239.11	172.29.242.11	172.29.246.11
az3_compute3	172.29.239.12	172.29.242.12	172.29.246.12

Host network configuration

Each host does require the correct network bridges to be implemented. In this example, we leverage the `systemd_networkd` role that performs configuration for us during `openstack_hosts` execution. It creates all required vlans and bridges. The only pre-requirement is to have a connection to the host via SSH available for Ansible to manage the host.

Note

Example assumes that default gateway is set through `bond0` interface, which aggregates `eth0` and `eth1` links. If your environment does not have `eth0`, but instead has `p1p1` or some other interface name, ensure that references to `eth0` are replaced with the appropriate name. The same applies to additional network interfaces

```
---
# VLAN Mappings
_az_vlan_mappings:
  az1:
    management: 10
    storage: 20
    tunnel: 30
    public-api: 400
  az2:
    management: 10
    storage: 21
    tunnel: 31
    public-api: 400
  az3:
    management: 10
    storage: 22
    tunnel: 32
    public-api: 400

# Bonding interfaces
_bond0_interfaces:
  - eth0
  - eth1

# NETDEV defenition
_systemd_networkd_default_devices:
  - NetDev:
      Name: vlan-mgmt
      Kind: vlan
    VLAN:
      Id: "{{ _az_vlan_mappings[az_name]['management'] }}"
    filename: 10-openstack-vlan-mgmt
  - NetDev:
      Name: bond0
      Kind: bond
    Bond:
```

(continues on next page)

(continued from previous page)

```
      Mode: 802.3ad
      TransmitHashPolicy: layer3+4
      LACPTransmitRate: fast
      MIIMonitorSec: 100
    filename: 05-general-bond0
  - NetDev:
      Name: "{{ management_bridge }}"
      Kind: bridge
    Bridge:
      ForwardDelaySec: 0
      HelloTimeSec: 2
      MaxAgeSec: 12
      STP: off
    filename: "11-openstack-{{ management_bridge }}"

_systemd_networkd_storage_devices:
  - NetDev:
      Name: vlan-stor
      Kind: vlan
    VLAN:
      Id: "{{ _az_vlan_mappings[az_name]['storage'] }}"
    filename: 12-openstack-vlan-stor
  - NetDev:
      Name: br-storage
      Kind: bridge
    Bridge:
      ForwardDelaySec: 0
      HelloTimeSec: 2
      MaxAgeSec: 12
      STP: off
    filename: 13-openstack-br-storage

_systemd_networkd_tunnel_devices:
  - NetDev:
      Name: vlan-tunnel
      Kind: vlan
    VLAN:
      Id: "{{ _az_vlan_mappings[az_name]['tunnel'] }}"
    filename: 16-openstack-vlan-tunnel

_systemd_networkd_pub_api_devices:
  - NetDev:
      Name: vlan-public-api
      Kind: vlan
    VLAN:
      Id: "{{ _az_vlan_mappings[az_name]['public-api'] }}"
    filename: 17-openstack-vlan-public-api
  - NetDev:
      Name: br-public-api
```

(continues on next page)

(continued from previous page)

```
      Kind: bridge
    Bridge:
      ForwardDelaySec: 0
      HelloTimeSec: 2
      MaxAgeSec: 12
      STP: off
    filename: 18-openstack-br-public-api

openstack_hosts_systemd_networkd_devices: |-
  {% set devices = [] %}
  {% if is_metal %}
  {%   set _ = devices.extend(_systemd_networkd_default_devices) %}
  {%   if inventory_hostname in (groups['compute_hosts'] + groups['storage_
↪hosts']) %}
  {%     set _ = devices.extend(_systemd_networkd_storage_devices) %}
  {%   endif %}
  {%   if inventory_hostname in (groups[az_name ~ '_ceph_mon_hosts'] +␣
↪groups[az_name ~ '_ceph_osd_hosts']) %}
  {%     set _ = devices.extend(_systemd_networkd_cluster_devices) %}
  {%   endif %}
  {%   if inventory_hostname in groups['compute_hosts'] %}
  {%     set _ = devices.extend(_systemd_networkd_tunnel_devices) %}
  {%   endif %}
  {%   if inventory_hostname in groups['haproxy_hosts'] %}
  {%     set _ = devices.extend(_systemd_networkd_pub_api_devices) %}
  {%   endif %}
  {% endif %}
  {{ devices }}

# NETWORK definition

# NOTE: this can work only in case management network has the same netmask as␣
↪all other networks
#       while in example manaement is /22 while rest are /24
# _management_rank: "{{ management_address | ansible.utils.
↪ipsubnet(hostvars[inventory_hostname]['cidr_networks']['management']) }}"
_management_rank: "{{ (management_address | split('.'))[-1] }}"

# NOTE: `05` is prefixed to filename to have precedence over netplan
_systemd_networkd_bonded_networks: |-
  {% set struct = [] %}
  {% for interface in _bond0_interfaces %}
  {%   set interface_data = ansible_facts[interface | replace('-', '_')] %}
  {%   set _ = struct.append({
                'interface': interface_data['device'],
                'filename' : '05-general-' ~ interface_data['device'],
                'bond': 'bond0',
                'link_config_overrides': {
                  'Match': {
```

(continues on next page)

(continued from previous page)

```
                    'MACAddress': interface_data['macaddress']
                  }
                }
              })
  %}
  {% endfor %}
  {% set bond_vlans = ['vlan-mgmt'] %}
  {% if inventory_hostname in (groups['compute_hosts'] + groups['storage_hosts
↪']) %}
  {%   set _ = bond_vlans.append('vlan-stor') %}
  {% endif %}
  {% if inventory_hostname in groups['haproxy_hosts'] %}
  {%   set _ = bond_vlans.append('vlan-public-api') %}
  {% endif %}
  {% if inventory_hostname in groups['compute_hosts'] %}
  {%   set _ = bond_vlans.append('vlan-tunnel') %}
  {% endif %}
  {% set _ = struct.append({
              'interface': 'bond0',
              'filename': '05-general-bond0',
              'vlan': bond_vlans
            })
  %}
  {{ struct }}

_systemd_networkd_mgmt_networks:
  - interface: "vlan-mgmt"
    bridge: "{{ management_bridge }}"
    filename: 10-openstack-vlan-mgmt
  - interface: "{{ management_bridge }}"
    address: "{{ management_address }}"
    netmask: "{{ cidr_networks['management'] | ansible.utils.ipaddr('netmask
↪') }}"
    filename: "11-openstack-{{ management_bridge }}"

_systemd_networkd_storage_networks:
  - interface: "vlan-stor"
    bridge: "br-storage"
    filename: 12-openstack-vlan-stor
  - interface: "br-storage"
    address: "{{ cidr_networks['storage_' ~ az_name] | ansible.utils.ipmath(_
↪management_rank) }}"
    netmask: "{{ cidr_networks['storage_' ~ az_name] | ansible.utils.ipaddr(
↪'netmask') }}"
    filename: "13-openstack-br-storage"

_systemd_networkd_tunnel_networks:
  - interface: "vlan-tunnel"
    filename: 16-openstack-vlan-tunnel
```

(continues on next page)

(continued from previous page)

```
    address: "{{ cidr_networks['tunnel_' ~ az_name] | ansible.utils.ipmath(_
↪management_rank) }}"
    netmask: "{{ cidr_networks['tunnel_' ~ az_name] | ansible.utils.ipaddr(
↪'netmask') }}"
    static_routes: |-
      {% set routes = [] %}
      {% set tunnel_cidrs = cidr_networks | dict2items | selectattr('key',
↪'match', 'tunnel_az[0-9]') | map(attribute='value') %}
      {% set gateway = cidr_networks['tunnel_' ~ az_name] | ansible.utils.
↪ipaddr('1') | ansible.utils.ipaddr('address') %}
      {% for cidr in tunnel_cidrs | reject('eq', cidr_networks['tunnel_' ~ az_
↪name]) %}
      {%   set _ = routes.append({'cidr': cidr, 'gateway': gateway}) %}
      {% endfor %}
      {{ routes }}

_systemd_networkd_pub_api_networks:
  - interface: "vlan-public-api"
    bridge: "br-public-api"
    filename: 17-openstack-vlan-public-api
  - interface: "br-public-api"
    filename: "18-openstack-br-public-api"

openstack_hosts_systemd_networkd_networks: |-
  {% set networks = [] %}
  {% if is_metal %}
  {%   set _ = networks.extend(_systemd_networkd_mgmt_networks + _systemd_
↪networkd_bonded_networks) %}
  {%   if inventory_hostname in (groups['compute_hosts'] + groups['storage_
↪hosts']) %}
  {%     set _ = networks.extend(_systemd_networkd_storage_networks) %}
  {%   endif %}
  {%   if inventory_hostname in groups['compute_hosts'] %}
  {%     set _ = networks.extend(_systemd_networkd_tunnel_networks) %}
  {%   endif %}
  {%   if inventory_hostname in groups['haproxy_hosts'] %}
  {%     set _ = networks.extend(_systemd_networkd_pub_api_networks) %}
  {%   endif %}
  {% endif %}
  {{ networks }}
```

Deployment configuration

Environment customizations

Deployed files in `/etc/openstack_deploy/env.d` allow the customization of Ansible groups.

To deploy HAProxy in container we need to create a file `/etc/openstack_deploy/env.d/haproxy.yml` with the following content:

```
---
# This file contains an example to show how to set
# the cinder-volume service to run in a container.
#
# Important note:
# In most cases you need to ensure that default route inside of the
# container doesn't go through eth0, which is part of lxcbr0 and
# SRC nat-ed. You need to pass "public" VIP interface inside of the
# container and ensure "default" route presence on it.

container_skel:
  haproxy_container:
    properties:
      is_metal: false
```

As we are using Ceph for this environment, so the `cinder-volume` runs in a container on the Ceph Monitor hosts. To achieve this, implement `/etc/openstack_deploy/env.d/cinder.yml` with the following content:

```
---
# This file contains an example to show how to set
# the cinder-volume service to run in a container.
#
# Important note:
# When using LVM or any iSCSI-based cinder backends, such as NetApp with
# iSCSI protocol, the cinder-volume service *must* run on metal.
# Reference: https://bugs.launchpad.net/ubuntu/+source/lxc/+bug/1226855

container_skel:
  cinder_volumes_container:
    properties:
      is_metal: false
```

In order to be able to execute a playbook only against hosts in a single Availability Zone, as well as be able to set AZ-specific variables, we need to define groups definitions. For that, create a file `/etc/openstack_deploy/env.d/az.yml` with the following content:

```
---

component_skel:
  az1_containers:
    belongs_to:
      - az1_all
  az1_hosts:
    belongs_to:
      - az1_all

  az2_containers:
    belongs_to:
      - az2_all
```

(continues on next page)

(continued from previous page)

```
  az2_hosts:
    belongs_to:
      - az2_all

  az3_containers:
    belongs_to:
      - az3_all
  az3_hosts:
    belongs_to:
      - az3_all

container_skel:
  az1_containers:
    properties:
      is_nest: true

  az2_containers:
    properties:
      is_nest: true

  az3_containers:
    properties:
      is_nest: true
```

Above example will create following groups:

- `azN_hosts` which will contain only bare metal nodes
- **`azN_containers` that will contain all containers that are spawned on** bare metal nodes, that are part of the pod.
- `azN_all` that will contain *azN_hosts* and *azN_containers* members

We also need to define a complete new set of groups for Ceph, to deploy multiple independent instances of it.

For that, create a file `/etc/openstack_deploy/env.d/ceph.yml` with the following content:

```
---
component_skel:
  # Ceph MON
  ceph_mon_az1:
    belongs_to:
      - ceph-mon
      - ceph_all
      - az1_all
  ceph_mon_az2:
    belongs_to:
      - ceph-mon
      - ceph_all
      - az2_all
  ceph_mon_az3:
```

(continues on next page)

(continued from previous page)

```
    belongs_to:
      - ceph-mon
      - ceph_all
      - az3_all

  # Ceph OSD
  ceph_osd_az1:
    belongs_to:
      - ceph-osd
      - ceph_all
      - az1_all
  ceph_osd_az2:
    belongs_to:
      - ceph-osd
      - ceph_all
      - az2_all
  ceph_osd_az3:
    belongs_to:
      - ceph-osd
      - ceph_all
      - az3_all

  # Ceph RGW
  ceph_rgw_az1:
    belongs_to:
      - ceph-rgw
      - ceph_all
      - az1_all
  ceph_rgw_az2:
    belongs_to:
      - ceph-rgw
      - ceph_all
      - az2_all
  ceph_rgw_az3:
    belongs_to:
      - ceph-rgw
      - ceph_all
      - az3_all

container_skel:
  # Ceph MON
  ceph_mon_container_az1:
    belongs_to:
      - az1_ceph_mon_containers
    contains:
      - ceph_mon_az1
  ceph_mon_container_az2:
    belongs_to:
      - az2_ceph_mon_containers
```

(continues on next page)

(continued from previous page)

```
    contains:
      - ceph_mon_az2
  ceph_mon_container_az3:
    belongs_to:
      - az3_ceph_mon_containers
    contains:
      - ceph_mon_az3

  # Ceph RGW
  ceph_rgw_container_az1:
    belongs_to:
      - az1_ceph_rgw_containers
    contains:
      - ceph_rgw_az1
  ceph_rgw_container_az2:
    belongs_to:
      - az2_ceph_rgw_containers
    contains:
      - ceph_rgw_az2
  ceph_rgw_container_az3:
    belongs_to:
      - az3_ceph_rgw_containers
    contains:
      - ceph_rgw_az3

  # Ceph OSD
  ceph_osd_container_az1:
    belongs_to:
      - az1_ceph_osd_containers
    contains:
      - ceph_osd_az1
    properties:
      is_metal: true
  ceph_osd_container_az2:
    belongs_to:
      - az2_ceph_osd_containers
    contains:
      - ceph_osd_az2
    properties:
      is_metal: true
  ceph_osd_container_az3:
    belongs_to:
      - az3_ceph_osd_containers
    contains:
      - ceph_osd_az3
    properties:
      is_metal: true
```

(continues on next page)

(continued from previous page)

```
physical_skel:
  # Ceph MON
  az1_ceph_mon_containers:
    belongs_to:
      - all_containers
  az1_ceph_mon_hosts:
    belongs_to:
      - hosts
  az2_ceph_mon_containers:
    belongs_to:
      - all_containers
  az2_ceph_mon_hosts:
    belongs_to:
      - hosts
  az3_ceph_mon_containers:
    belongs_to:
      - all_containers
  az3_ceph_mon_hosts:
    belongs_to:
      - hosts

  # Ceph OSD
  az1_ceph_osd_containers:
    belongs_to:
      - all_containers
  az1_ceph_osd_hosts:
    belongs_to:
      - hosts
  az2_ceph_osd_containers:
    belongs_to:
      - all_containers
  az2_ceph_osd_hosts:
    belongs_to:
      - hosts
  az3_ceph_osd_containers:
    belongs_to:
      - all_containers
  az3_ceph_osd_hosts:
    belongs_to:
      - hosts

  # Ceph RGW
  az1_ceph_rgw_containers:
    belongs_to:
      - all_containers
  az1_ceph_rgw_hosts:
    belongs_to:
      - hosts
  az2_ceph_rgw_containers:
```

(continues on next page)

(continued from previous page)

```
    belongs_to:
      - all_containers
  az2_ceph_rgw_hosts:
    belongs_to:
      - hosts
  az3_ceph_rgw_containers:
    belongs_to:
      - all_containers
  az3_ceph_rgw_hosts:
    belongs_to:
      - hosts
```

Environment layout

The `/etc/openstack_deploy/openstack_user_config.yml` file defines the environment layout.

For each AZ, a group will need to be defined containing all hosts within that AZ.

Within defined provider networks, `address_prefix` is used to override the prefix of the key added to each host that contains IP address information. We use AZ-specific prefixes for `container`, `tunnel`, or `storage`. `reference_group` contains the name of a defined AZ group and is used to limit the scope of each provider network to that group.

YAML Anchors and Aliases are used heavily in the example below to populate all groups that might become handy while not repeating hosts definitions each time. You can read more about the topic in Ansible Documentation

The following configuration describes the layout for this environment.

```
---

cidr_networks: &os_cidrs
  management: 172.29.236.0/22
  tunnel_az1: 172.29.240.0/24
  tunnel_az2: 172.29.241.0/24
  tunnel_az3: 172.29.242.0/24
  storage_az1: 172.29.244.0/24
  storage_az2: 172.29.245.0/24
  storage_az3: 172.29.246.0/24
  public_api_vip: 203.0.113.0/28

used_ips:
  # management network - openstack VIPs
  - "172.29.236.1,172.29.236.30"
  # management network - other hosts not managed by OSA dynamic inventory
  - "172.29.238.0,172.29.239.255"
  # storage network - reserved for ceph hosts
  - "172.29.244.200,172.29.244.250"
  - "172.29.245.200,172.29.245.250"
  - "172.29.246.200,172.29.246.250"
  # public_api
```

(continues on next page)

(continued from previous page)

```
  - "203.0.113.1,203.0.113.10"

global_overrides:
  internal_lb_vip_address: internal.example.cloud
  external_lb_vip_address: example.cloud
  management_bridge: "br-mgmt"
  cidr_networks: *os_cidrs
  provider_networks:
    - network:
        group_binds:
          - all_containers
          - hosts
        type: "raw"
        container_bridge: "br-mgmt"
        container_interface: "eth1"
        container_type: "veth"
        ip_from_q: "management"
        is_management_address: true
    - network:
        container_bridge: "br-storage"
        container_type: "veth"
        container_interface: "eth2"
        ip_from_q: "storage_az1"
        address_prefix: "storage_az1"
        type: "raw"
        group_binds:
          - cinder_volume
          - nova_compute
          - ceph_all
        reference_group: "az1_all"
    - network:
        container_bridge: "br-storage"
        container_type: "veth"
        container_interface: "eth2"
        ip_from_q: "storage_az2"
        address_prefix: "storage_az2"
        type: "raw"
        group_binds:
          - cinder_volume
          - nova_compute
          - ceph_all
        reference_group: "az2_all"
    - network:
        container_bridge: "br-storage"
        container_type: "veth"
        container_interface: "eth2"
        ip_from_q: "storage_az3"
        address_prefix: "storage_az3"
        type: "raw"
```

(continues on next page)

(continued from previous page)

```
        group_binds:
          - cinder_volume
          - nova_compute
          - ceph_all
        reference_group: "az3_all"
    - network:
        container_bridge: "vlan-tunnel"
        container_type: "veth"
        container_interface: "eth4"
        ip_from_q: "tunnel_az1"
        address_prefix: "tunnel"
        type: "raw"
        group_binds:
          - neutron_ovn_controller
        reference_group: "az1_all"
    - network:
        container_bridge: "vlan-tunnel"
        container_type: "veth"
        container_interface: "eth4"
        ip_from_q: "tunnel_az2"
        address_prefix: "tunnel"
        type: "raw"
        group_binds:
          - neutron_ovn_controller
        reference_group: "az2_all"
    - network:
        container_bridge: "vlan-tunnel"
        container_type: "veth"
        container_interface: "eth4"
        ip_from_q: "tunnel_az3"
        address_prefix: "tunnel"
        type: "raw"
        group_binds:
          - neutron_ovn_controller
        reference_group: "az3_all"
    - network:
        group_binds:
          - haproxy
        type: "raw"
        container_bridge: "br-public-api"
        container_interface: "eth20"
        container_type: "veth"
        ip_from_q: public_api_vip
        static_routes:
          - cidr: 0.0.0.0/0
            gateway: 203.0.113.1

### conf.d configuration ###
```

(continues on next page)

(continued from previous page)

```
# Control plane
az1_controller_hosts: &controller_az1
  infra1:
    ip: 172.29.236.11

az2_controller_hosts: &controller_az2
  infra2:
    ip: 172.29.236.12

az3_controller_hosts: &controller_az3
  infra3:
    ip: 172.29.236.13

# Computes

## AZ1
az1_shared_compute_hosts: &shared_computes_az1
  az1_compute1:
    ip: 172.29.237.11
  az1_compute2:
    ip: 172.29.237.12

az1_pinned_compute_hosts: &pinned_computes_az1
  az1_pin_compute1:
    ip: 172.29.237.13
  az1_pin_compute2:
    ip: 172.29.237.14

## AZ2

az2_shared_compute_hosts: &shared_computes_az2
  az2_compute1:
    ip: 172.29.238.11
  az2_compute2:
    ip: 172.29.238.12

## AZ3
az3_shared_compute_hosts: &shared_computes_az3
  az3_compute1:
    ip: 172.29.239.11
  az3_compute2:
    ip: 172.29.239.12

# Storage

## AZ1
az1_storage_hosts: &storage_az1
  az1_ceph1:
    ip: 172.29.237.201
```

(continues on next page)

(continued from previous page)

```
  az1_ceph2:
    ip: 172.29.237.202
  az1_ceph3:
    ip: 172.29.237.203

## AZ2
az2_storage_hosts: &storage_az2
  az2_ceph1:
    ip: 172.29.238.201
  az2_ceph2:
    ip: 172.29.238.202
  az2_ceph3:
    ip: 172.29.238.203

## AZ3
az3_storage_hosts: &storage_az3
  az3_ceph1:
    ip: 172.29.239.201
  az3_ceph2:
    ip: 172.29.239.202
  az3_ceph3:
    ip: 172.29.239.203

# AZ association

az1_compute_hosts: &compute_hosts_az1
  <<: [*shared_computes_az1, *pinned_computes_az1]

az2_compute_hosts: &compute_hosts_az2
  <<: *shared_computes_az2

az3_compute_hosts: &compute_hosts_az3
  <<: *shared_computes_az3

az1_hosts:
  <<: [*compute_hosts_az1, *controller_az1, *storage_az1]

az2_hosts:
  <<: [*compute_hosts_az2, *controller_az2, *storage_az2]

az3_hosts:
  <<: [*compute_hosts_az3, *controller_az3, *storage_az3]

# Final mappings
shared_infra_hosts: &controllers
  <<: [*controller_az1, *controller_az2, *controller_az3]

repo-infra_hosts: *controllers
memcaching_hosts: *controllers
```

(continues on next page)

(continued from previous page)

```
database_hosts: *controllers
mq_hosts: *controllers
operator_hosts: *controllers
identity_hosts: *controllers
image_hosts: *controllers
dashboard_hosts: *controllers
compute-infra_hosts: *controllers
placement-infra_hosts: *controllers
storage-infra_hosts: *controllers
network-infra_hosts: *controllers
network-northd_hosts: *controllers
coordination_hosts: *controllers

compute_hosts: &computes
  <<: [*compute_hosts_az1, *compute_hosts_az2, *compute_hosts_az3]

pinned_compute_hosts:
  <<: *pinned_computes_az1

shared_compute_hosts:
  <<: [*shared_computes_az1, *shared_computes_az2, *shared_computes_az3]

network-gateway_hosts: *computes

storage_hosts: &storage
  <<: [*storage_az1, *storage_az2, *storage_az3]

az1_ceph_osd_hosts:
  <<: *storage_az1

az2_ceph_osd_hosts:
  <<: *storage_az2

az3_ceph_osd_hosts:
  <<: *storage_az3

az1_ceph_mon_hosts:
  <<: *storage_az1

az2_ceph_mon_hosts:
  <<: *storage_az2

az3_ceph_mon_hosts:
  <<: *storage_az3

az1_ceph_rgw_hosts:
  <<: *storage_az1

az2_ceph_rgw_hosts:
```

(continues on next page)

(continued from previous page)

```
  <<: *storage_az2

az3_ceph_rgw_hosts:
  <<: *storage_az3
```

User variables

In order to properly configure Availability Zones, we need to leverage `group_vars` and define Availability Zone name used for each AZ there. For this, create files:

- `/etc/openstack_deploy/group_vars/az1_all.yml`
- `/etc/openstack_deploy/group_vars/az2_all.yml`
- `/etc/openstack_deploy/group_vars/az3_all.yml`

With content like below, where N should be AZ number depending on the file:

```
az_name: azN
```

As for this environment, the load balancer is created in the LXC containers on the infrastructure hosts, we need to ensure absence of the default route on `eth0` interface. To prevent that from happening, we override `lxc_container_networks` in `/etc/openstack_deploy/group_vars/haproxy/lxc_network.yml` file:

```
---
lxc_container_networks:
  lxcbr0_address:
    bridge: "{{ lxc_net_bridge | default('lxcbr0') }}"
    bridge_type: "{{ lxc_net_bridge_type | default('linuxbridge') }}"
    interface: eth0
    type: veth
    dhcp_use_routes: False
```

Next, we want to secure HAProxy pointing always to the backend which is considered as local to the HAProxy. For that we switch balancing algorithm to `first` and order re-backends so that the one from current Availability Zone appears to be the first in the list. This can be done by creating file `/etc/openstack_deploy/group_vars/haproxy/backend_overrides.yml` with content:

```
---

haproxy_drain: true
haproxy_ssl_all_vips: true
haproxy_bind_external_lb_vip_interface: eth20
haproxy_bind_internal_lb_vip_interface: eth1
haproxy_bind_external_lb_vip_address: "*"
haproxy_bind_internal_lb_vip_address: "*"
haproxy_vip_binds:
  - address: "{{ haproxy_bind_external_lb_vip_address }}"
    interface: "{{ haproxy_bind_external_lb_vip_interface }}"
    type: external
  - address: "{{ haproxy_bind_internal_lb_vip_address }}"
```

(continues on next page)

(continued from previous page)

```
    interface: "{{ haproxy_bind_internal_lb_vip_interface }}"
    type: internal

haproxy_cinder_api_service_overrides:
  haproxy_backend_nodes: "{{ groups['cinder_api'] | select('in', groups[az_
↪name ~ '_containers']) | union(groups['cinder_api']) | unique | default([])␣
↪}}"
  haproxy_balance_alg: first
  haproxy_limit_hosts: "{{ groups['haproxy_all'] | intersect(groups[az_name ~
↪'_all']) }}"

haproxy_horizon_service_overrides:
  haproxy_backend_nodes: "{{ groups['horizon_all'] | select('in', groups[az_
↪name ~ '_containers']) | union(groups['horizon_all']) | unique |␣
↪default([]) }}"
  haproxy_balance_alg: first
  haproxy_limit_hosts: "{{ groups['haproxy_all'] | intersect(groups[az_name ~
↪'_all']) }}"

haproxy_keystone_service_overrides:
  haproxy_backend_nodes: "{{ groups['keystone_all'] | select('in', groups[az_
↪name ~ '_containers']) | union(groups['keystone_all']) | unique |␣
↪default([]) }}"
  haproxy_balance_alg: first
  haproxy_limit_hosts: "{{ groups['haproxy_all'] | intersect(groups[az_name ~
↪'_all']) }}"

haproxy_neutron_server_service_overrides:
  haproxy_backend_nodes: "{{ groups['neutron_server'] | select('in',␣
↪groups[az_name ~ '_containers']) | union(groups['neutron_server']) | unique␣
↪| default([]) }}"
  haproxy_balance_alg: first
  haproxy_limit_hosts: "{{ groups['haproxy_all'] | intersect(groups[az_name ~
↪'_all']) }}"

haproxy_nova_api_compute_service_overrides:
  haproxy_backend_nodes: "{{ groups['nova_api_os_compute'] | select('in',␣
↪groups[az_name ~ '_containers']) | union(groups['nova_api_os_compute']) |␣
↪unique | default([]) }}"
  haproxy_balance_alg: first
  haproxy_limit_hosts: "{{ groups['haproxy_all'] | intersect(groups[az_name ~
↪'_all']) }}"

haproxy_nova_api_metadata_service_overrides:
  haproxy_backend_nodes: "{{ groups['nova_api_metadata'] | select('in',␣
↪groups[az_name ~ '_containers']) | union(groups['nova_api_metadata']) |␣
↪unique | default([]) }}"
  haproxy_balance_alg: first
  haproxy_limit_hosts: "{{ groups['haproxy_all'] | intersect(groups[az_name ~
```

(continues on next page)

(continued from previous page)

```
↪'_all']) }}"

haproxy_placement_service_overrides:
  haproxy_backend_nodes: "{{ groups['placement_all'] | select('in', groups[az_
↪name ~ '_containers']) | union(groups['placement_all']) | unique |␣
↪default([]) }}"
  haproxy_balance_alg: first
  haproxy_limit_hosts: "{{ groups['haproxy_all'] | intersect(groups[az_name ~
↪'_all']) }}"

haproxy_repo_service_overrides:
  haproxy_backend_nodes: "{{ groups['repo_all'] | select('in', groups[az_name␣
↪~ '_containers']) | union(groups['repo_all']) | unique | default([]) }}"
  haproxy_balance_alg: first
  haproxy_limit_hosts: "{{ groups['haproxy_all'] | intersect(groups[az_name ~
↪'_all']) }}"
```

We also need to define a couple of extra Keepalived instances in order to secure DNS RR approach, along with configuring Keepalived in unicast mode. For that create a file `/etc/openstack_deploy/group_vars/haproxy/keepalived.yml` with following content:

```
---

haproxy_keepalived_external_vip_cidr_az1: 203.0.113.5/32
haproxy_keepalived_external_vip_cidr_az2: 203.0.113.6/32
haproxy_keepalived_external_vip_cidr_az3: 203.0.113.7/32
haproxy_keepalived_internal_vip_cidr_az1: 172.29.236.21/32
haproxy_keepalived_internal_vip_cidr_az2: 172.29.236.22/32
haproxy_keepalived_internal_vip_cidr_az3: 172.29.236.23/32
haproxy_keepalived_external_interface: "{{ haproxy_bind_external_lb_vip_
↪interface }}"
haproxy_keepalived_internal_interface: "{{ haproxy_bind_internal_lb_vip_
↪interface }}"

keepalived_unicast_peers:
  internal: |-
    {% set peers = [] %}
    {% for addr in groups['haproxy'] | map('extract', hostvars, ['container_
↪networks', 'management_address']) %}
    {%   set _ = peers.append((addr['address'] ~ '/' ~ addr['netmask']) |␣
↪ansible.utils.ipaddr('host/prefix')) %}
    {% endfor %}
    {{ peers }}
  external: |-
    {% set peers = [] %}
    {% for addr in groups['haproxy'] | map('extract', hostvars, ['container_
↪networks', 'public_api_vip_address']) %}
    {%   set _ = peers.append((addr['address'] ~ '/' ~ addr['netmask']) |␣
↪ansible.utils.ipaddr('host/prefix')) %}
    {% endfor %}
```

(continues on next page)

(continued from previous page)

```
    {{ peers }}

keepalived_internal_unicast_src_ip: >-
  {{ (management_address ~ '/' ~ container_networks['management_address'][
↪'netmask']) | ansible.utils.ipaddr('host/prefix') }}
keepalived_external_unicast_src_ip: >-
  {{ (container_networks['public_api_vip_address']['address'] ~ '/' ~␣
↪container_networks['public_api_vip_address']['netmask']) | ansible.utils.
↪ipaddr('host/prefix') }}

keepalived_instances:
  az1-external:
    interface: "{{ haproxy_keepalived_external_interface | default(management_
↪bridge) }}"
    state: "{{ (inventory_hostname in groups['az1_all']) | ternary('MASTER',
↪'BACKUP') }}"
    virtual_router_id: 40
    priority: "{{ (inventory_hostname in groups['az1_all']) | ternary(200,␣
↪(groups['haproxy']|length-groups['haproxy'].index(inventory_hostname))*50) }
↪}"
    vips:
      - "{{ haproxy_keepalived_external_vip_cidr_az1 | default('169.254.1.1/24
↪')  }} dev {{ haproxy_keepalived_external_interface | default(management_
↪bridge) }}"
    track_scripts: "{{ keepalived_scripts | dict2items | json_query('[*].
↪{name: key, instance: value.instance}') | rejectattr('instance', 'equalto',
↪'internal') | map(attribute='name') | list }}"
    unicast_src_ip: "{{ keepalived_external_unicast_src_ip }}"
    unicast_peers: "{{ keepalived_unicast_peers['external'] |␣
↪difference([keepalived_external_unicast_src_ip]) }}"
  az1-internal:
    interface: "{{ haproxy_keepalived_internal_interface | default(management_
↪bridge) }}"
    state: "{{ (inventory_hostname in groups['az1_all']) | ternary('MASTER',
↪'BACKUP') }}"
    virtual_router_id: 41
    priority: "{{ (inventory_hostname in groups['az1_all']) | ternary(200,␣
↪(groups['haproxy']|length-groups['haproxy'].index(inventory_hostname))*50) }
↪}"
    vips:
      - "{{ haproxy_keepalived_internal_vip_cidr_az1 | default('169.254.2.1/24
↪') }} dev {{ haproxy_keepalived_internal_interface | default(management_
↪bridge) }}"
    track_scripts: "{{ keepalived_scripts | dict2items | json_query('[*].
↪{name: key, instance: value.instance}') | rejectattr('instance', 'equalto',
↪'external') | map(attribute='name') | list }}"
    unicast_src_ip: "{{ keepalived_internal_unicast_src_ip }}"
    unicast_peers: "{{ keepalived_unicast_peers['internal'] |␣
↪difference([keepalived_internal_unicast_src_ip]) }}"
```

(continues on next page)

(continued from previous page)

```
  az2-external:
    interface: "{{ haproxy_keepalived_external_interface | default(management_
↪bridge) }}"
    state: "{{ (inventory_hostname in groups['az2_all']) | ternary('MASTER',
↪'BACKUP') }}"
    virtual_router_id: 42
    priority: "{{ (inventory_hostname in groups['az2_all']) | ternary(200,␣
↪(groups['haproxy']|length-groups['haproxy'].index(inventory_hostname))*50) }
↪}"
    vips:
      - "{{ haproxy_keepalived_external_vip_cidr_az2 | default('169.254.1.1/24
↪')  }} dev {{ haproxy_keepalived_external_interface | default(management_
↪bridge) }}"
    track_scripts: "{{ keepalived_scripts | dict2items | json_query('[*].
↪{name: key, instance: value.instance}') | rejectattr('instance', 'equalto',
↪'internal') | map(attribute='name') | list }}"
    unicast_src_ip: "{{ keepalived_external_unicast_src_ip }}"
    unicast_peers: "{{ keepalived_unicast_peers['external'] |␣
↪difference([keepalived_external_unicast_src_ip]) }}"
  az2-internal:
    interface: "{{ haproxy_keepalived_internal_interface | default(management_
↪bridge) }}"
    state: "{{ (inventory_hostname in groups['az2_all']) | ternary('MASTER',
↪'BACKUP') }}"
    virtual_router_id: 43
    priority: "{{ (inventory_hostname in groups['az2_all']) | ternary(200,␣
↪(groups['haproxy']|length-groups['haproxy'].index(inventory_hostname))*50) }
↪}"
    vips:
      - "{{ haproxy_keepalived_internal_vip_cidr_az2 | default('169.254.2.1/24
↪') }} dev {{ haproxy_keepalived_internal_interface | default(management_
↪bridge) }}"
    track_scripts: "{{ keepalived_scripts | dict2items | json_query('[*].
↪{name: key, instance: value.instance}') | rejectattr('instance', 'equalto',
↪'external') | map(attribute='name') | list }}"
    unicast_src_ip: "{{ keepalived_internal_unicast_src_ip }}"
    unicast_peers: "{{ keepalived_unicast_peers['internal'] |␣
↪difference([keepalived_internal_unicast_src_ip]) }}"

  az3-external:
    interface: "{{ haproxy_keepalived_external_interface | default(management_
↪bridge) }}"
    state: "{{ (inventory_hostname in groups['az3_all']) | ternary('MASTER',
↪'BACKUP') }}"
    virtual_router_id: 44
    priority: "{{ (inventory_hostname in groups['az3_all']) | ternary(200,␣
↪(groups['haproxy']|length-groups['haproxy'].index(inventory_hostname))*50) }
↪}"
```

(continues on next page)

(continued from previous page)

```
    vips:
      - "{{ haproxy_keepalived_external_vip_cidr_az3 | default('169.254.1.1/24
↪')  }} dev {{ haproxy_keepalived_external_interface | default(management_
↪bridge) }}"
    track_scripts: "{{ keepalived_scripts | dict2items | json_query('[*].
↪{name: key, instance: value.instance}') | rejectattr('instance', 'equalto',
↪'internal') | map(attribute='name') | list }}"
    unicast_src_ip: "{{ keepalived_external_unicast_src_ip }}"
    unicast_peers: "{{ keepalived_unicast_peers['external'] |␣
↪difference([keepalived_external_unicast_src_ip]) }}"
  az3-internal:
    interface: "{{ haproxy_keepalived_internal_interface | default(management_
↪bridge) }}"
    state: "{{ (inventory_hostname in groups['az3_all']) | ternary('MASTER',
↪'BACKUP') }}"
    virtual_router_id: 45
    priority: "{{ (inventory_hostname in groups['az3_all']) | ternary(200,␣
↪(groups['haproxy']|length-groups['haproxy'].index(inventory_hostname))*50) }
↪}"
    vips:
      - "{{ haproxy_keepalived_internal_vip_cidr_az3 | default('169.254.2.1/24
↪') }} dev {{ haproxy_keepalived_internal_interface | default(management_
↪bridge) }}"
    track_scripts: "{{ keepalived_scripts | dict2items | json_query('[*].
↪{name: key, instance: value.instance}') | rejectattr('instance', 'equalto',
↪'external') | map(attribute='name') | list }}"
    unicast_src_ip: "{{ keepalived_internal_unicast_src_ip }}"
    unicast_peers: "{{ keepalived_unicast_peers['internal'] |␣
↪difference([keepalived_internal_unicast_src_ip]) }}"
```

In order to add support for multiple compute tiers (in with CPU overcommit and pinned CPUs) you need to create a file `/etc/openstack_deploy/group_vars/pinned_compute_hosts` with content:

```
nova_cpu_allocation_ratio: 1.0
nova_ram_allocation_ratio: 1.0
```

Rest of variables can be defined in `/etc/openstack_deploy/user_variables.yml` but a lot of them will be referencing `az_name` variable, so its presence (along with corresponding groups) are vital for this scenario.

```
---
# Set a different scheduling AZ name on each controller
# You can change that to a specific AZ name which will be used as default one
default_availability_zone: "{{ az_name }}"

# Defining unique internal VIP in hosts per AZ
_openstack_internal_az_vip: "{{ hostvars[groups['haproxy'][0]]['haproxy_
↪keepalived_internal_vip_cidr_' ~ az_name] | ansible.utils.ipaddr('address')␣
↪}}"
openstack_host_custom_hosts_records: "{{ _openstack_services_fqdns['internal
```

(continues on next page)

(continued from previous page)

```
↪'] | map('regex_replace', '^(.*)$', _openstack_internal_az_vip ~ ' \\1') }}"

# Use local to AZ memcached inside of AZ
memcached_servers: >-
  {{
    groups['memcached'] | intersect(groups[az_name ~ '_containers'])
      | map('extract', hostvars, 'management_address')
      | map('regex_replace', '(.+)', '\1:' ~ memcached_port)
      | list | join(',')
  }}

# Ceph-Ansible variables
ceph_cluster_name: "ceph-{{ az_name }}"
ceph_keyrings_dir: "/etc/openstack_deploy/ceph/{{ ceph_cluster_name }}"
ceph_conf_file: "{{ lookup('file', ceph_keyrings_dir ~ '/ceph.conf') }}"
cluster: "{{ ceph_cluster_name }}"
cluster_network: "{{ public_network }}"
monitor_address: "{{ container_networks['storage_address']['address'] }}"
mon_group_name: "ceph_mon_{{ az_name }}"
mgr_group_name: "{{ mon_group_name }}"
osd_group_name: "ceph_osd_{{ az_name }}"
public_network: "{{ cidr_networks['storage_' ~ az_name] }}"
rgw_group_name: "ceph_rgw_{{ az_name }}"
rgw_zone: "{{ az_name }}"

# Cinder variables
cinder_active_active_cluster_name: "{{ ceph_cluster_name }}"
cinder_default_availability_zone: "{{ default_availability_zone }}"
cinder_storage_availability_zone: "{{ az_name }}"

# Glance to use Swift as a backend
glance_default_store: swift
glance_use_uwsgi: False

# Neutron variables
neutron_availability_zone: "{{ az_name }}"
neutron_default_availability_zones:
  - az1
  - az2
  - az3
neutron_ovn_distributed_fip: True
neutron_plugin_type: ml2.ovn
neutron_plugin_base:
  - ovn-router
  - qos
  - auto_allocate
neutron_ml2_drivers_type: geneve,vlan
neutron_provider_networks:
  network_types: "{{ neutron_ml2_drivers_type }}"
```

(continues on next page)

(continued from previous page)

```
  network_geneve_ranges: "1:65000"
  network_vlan_ranges: >-
    vlan:100:200
  network_mappings: "vlan:br-vlan"
  network_interface_mappings: "br-vlan:bond0"

# Nova variables
nova_cinder_rbd_inuse: True
nova_glance_rbd_inuse: false
nova_libvirt_images_rbd_pool: ""
nova_libvirt_disk_cachemodes: network=writeback,file=directsync
nova_libvirt_hw_disk_discard: unmap
nova_nova_conf_overrides:
  DEFAULT:
    default_availability_zone: "{{ default_availability_zone }}"
    default_schedule_zone: "{{ default_availability_zone }}"
  cinder:
    cross_az_attach: false

# Create required aggregates and flavors
cpu_pinned_flavors:
  specs:
    - name: pinned.small
      vcpus: 2
      ram: 2048
    - name: pinned.medium
      vcpus: 4
      ram: 8192
  extra_specs:
    hw:cpu_policy: dedicated
    hw:vif_multiqueue_enabled: 'true'
    trait:CUSTOM_PINNED_CPU: required

cpu_shared_flavors:
  specs:
    - name: shared.small
      vcpus: 1
      ram: 1024
    - name: shared.medium
      vcpus: 2
      ram: 4096

openstack_user_compute:
  flavors:
    - "{{ cpu_shared_flavors }}"
    - "{{ cpu_pinned_flavors }}"
  aggregates:
    - name: az1-shared
      hosts: "{{ groups['az1_shared_compute_hosts'] }}"
```

(continues on next page)

(continued from previous page)

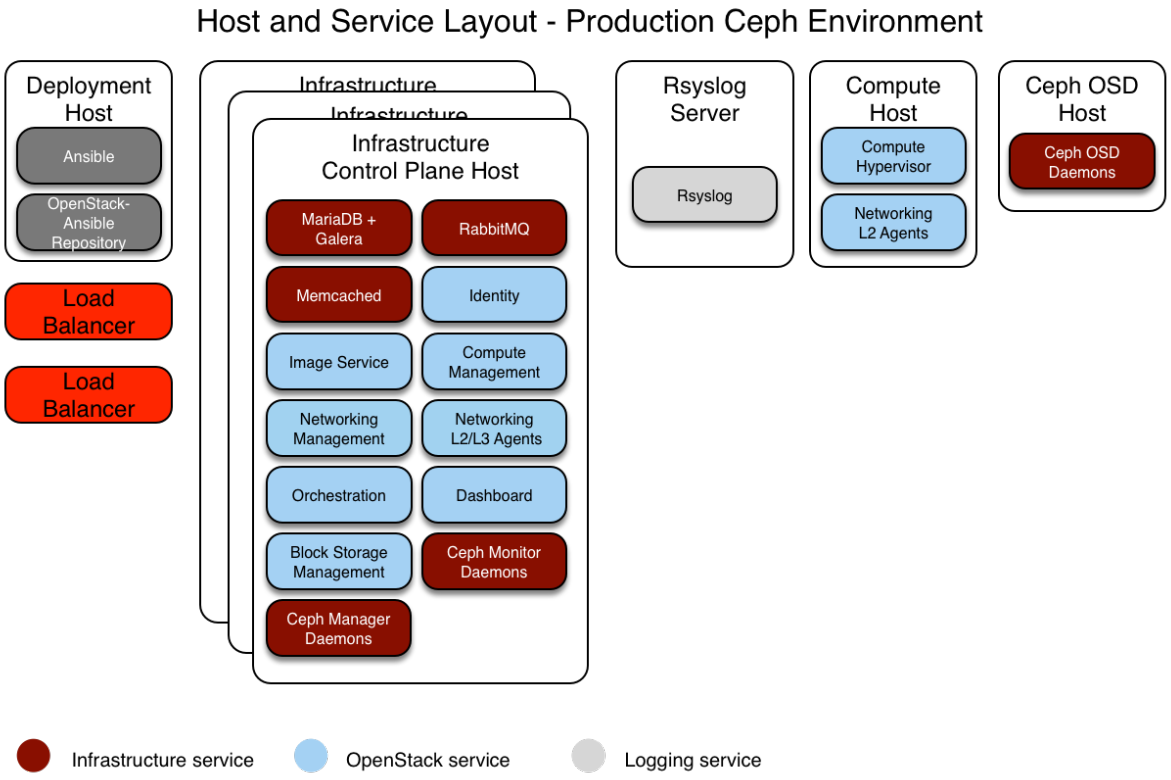
```
    availability_zone: az1
  - name: az1-pinned
    hosts: "{{ groups['az1_pinned_compute_hosts'] }}"
    availability_zone: az1
    metadata:
      trait:CUSTOM_PINNED_CPU: required
      pinned-cpu: 'true'
  - name: az2-shared
    hosts: "{{ groups['az2_shared_compute_hosts'] }}"
    availability_zone: az2
  - name: az3-shared
    hosts: "{{ groups['az3_shared_compute_hosts'] }}"
    availability_zone: az3
```

Ceph production example

This section describes an example production environment for a working OpenStack-Ansible (OSA) deployment with high availability services and using the Ceph backend for images, volumes, and instances.

This example environment has the following characteristics:

- Three infrastructure (control plane) hosts with ceph-mon containers
- Two compute hosts
- Three Ceph OSD storage hosts
- One log aggregation host
- Multiple Network Interface Cards (NIC) configured as bonded pairs for each host
- Full compute kit with the Telemetry service (ceilometer) included, with Ceph configured as a storage back end for the Image (glance), and Block Storage (cinder) services
- Internet access via the router address 172.29.236.1 on the Management Network



Integration with Ceph

OpenStack-Ansible allows Ceph storage cluster integration in three ways:

- connecting to your own pre-deployed ceph cluster by pointing to its information in `user_variables.yml` and allowing OpenStack-Ansible to ssh to the ceph monitors to retrieve the contents of ceph.conf and the keyrings.

  This method only requires a very small amount of configuration in `user_variables.yml` to point to the external ceph cluster monitors. The whole configuration for ceph-ansible would live outside the OpenStack-Ansible deployment and there is no duplication. The `ceph_mons` variable expects a list of IP addresses for the Ceph Monitor servers in the external ceph deployment:

Note

Overriding ceph_mons is required only when you are using external cluster which does not present in the OpenStack-Ansibles inventory (ie group `mon_group_name` is not defined).

```
ceph_mons:
    - 172.29.244.151
    - 172.29.244.152
    - 172.29.244.153
```

- connecting to your own pre-deployed ceph cluster by pointing to its monitors in `user_variables.yml` as above and providing data to populate ceph.conf and ceph keyring files on the deploy host. This is described here. No ssh access by OpenStack-Ansible is required to the ceph cluster.
- deploying a ceph cluster as part of the OpenStack-Ansible deployment by using the

roles maintained by the Ceph-Ansible project. Deployers can enable the `ceph-install.yml` playbook by adding hosts to the `ceph-mon_hosts` and `ceph-osd_hosts` groups in `openstack_user_config.yml`. In order to enable `ceph-rgw-install.yml` playbook you need to add `ceph-rgw_hosts` in `openstack_user_config.yml`.

Note

Please mention, that RGW installation should be performed after deployment of Keystone service.

Once groups are defined, you can proceed with configuring Ceph-Ansible specific vars in the OpenStack-Ansible `user_variables.yml` file.

Warning

Deploying ceph cluster as part of OpenStack-Ansible is not recommended since ceph-ansible upgrade path is not tested or supported. This option is mainly used for CI and AIO deployments to test and demonstrate a sample integration of the software stack.

This example will focus on the deployment of both OpenStack-Ansible and its Ceph cluster.

Network configuration

Network CIDR/VLAN assignments

The following CIDR and VLAN assignments are used for this environment.

Network	CIDR	VLAN
Management Network	172.29.236.0/22	10
Tunnel (VXLAN) Network	172.29.240.0/22	30
Storage Network	172.29.244.0/22	20

IP assignments

The following host name and IP address assignments are used for this environment.

Host name	Management IP	Tunnel (VXLAN) IP	Storage IP
lb_vip_address	172.29.236.9		
infra1	172.29.236.11	172.29.240.11	
infra2	172.29.236.12	172.29.240.12	
infra3	172.29.236.13	172.29.240.13	
log1	172.29.236.14		
compute1	172.29.236.16	172.29.240.16	172.29.244.16
compute2	172.29.236.17	172.29.240.17	172.29.244.17
osd1	172.29.236.18		172.29.244.18
osd2	172.29.236.19		172.29.244.19
osd3	172.29.236.20		172.29.244.20

Host network configuration

Each host will require the correct network bridges to be implemented. The following is the `/etc/network/interfaces` file for `infra1`.

Note

If your environment does not have `eth0`, but instead has `p1p1` or some other interface name, ensure that all references to `eth0` in all configuration files are replaced with the appropriate name. The same applies to additional network interfaces.

```
# This is a multi-NIC bonded configuration to implement the required bridges
# for OpenStack-Ansible. This illustrates the configuration of the first
# Infrastructure host and the IP addresses assigned should be adapted
# for implementation on the other hosts.
#
# After implementing this configuration, the host will need to be
# rebooted.

# Assuming that eth0/1 and eth2/3 are dual port NIC's we pair
# eth0 with eth2 and eth1 with eth3 for increased resiliency
# in the case of one interface card failing.
auto eth0
iface eth0 inet manual
    bond-master bond0
    bond-primary eth0

auto eth1
iface eth1 inet manual
    bond-master bond1
    bond-primary eth1

auto eth2
iface eth2 inet manual
    bond-master bond0

auto eth3
iface eth3 inet manual
    bond-master bond1

# Create a bonded interface. Note that the "bond-slaves" is set to none. This
# is because the bond-master has already been set in the raw interfaces for
# the new bond0.
auto bond0
iface bond0 inet manual
    bond-slaves none
    bond-mode active-backup
    bond-miimon 100
    bond-downdelay 200
    bond-updelay 200
```

(continues on next page)

(continued from previous page)

```
# This bond will carry VLAN and VXLAN traffic to ensure isolation from
# control plane traffic on bond0.
auto bond1
iface bond1 inet manual
    bond-slaves none
    bond-mode active-backup
    bond-miimon 100
    bond-downdelay 250
    bond-updelay 250

# Container/Host management VLAN interface
auto bond0.10
iface bond0.10 inet manual
    vlan-raw-device bond0

# OpenStack Networking VXLAN (tunnel/overlay) VLAN interface
auto bond1.30
iface bond1.30 inet manual
    vlan-raw-device bond1

# Storage network VLAN interface (optional)
auto bond0.20
iface bond0.20 inet manual
    vlan-raw-device bond0

# Container/Host management bridge
auto br-mgmt
iface br-mgmt inet static
    bridge_stp off
    bridge_waitport 0
    bridge_fd 0
    bridge_ports bond0.10
    address 172.29.236.11
    netmask 255.255.252.0
    gateway 172.29.236.1
    dns-nameservers 8.8.8.8 8.8.4.4

# OpenStack Networking VXLAN (tunnel/overlay) bridge
#
# The COMPUTE, NETWORK and INFRA nodes must have an IP address
# on this bridge.
#

auto br-vxlan
iface br-vxlan inet static
    bridge_stp off
    bridge_waitport 0
    bridge_fd 0
```

(continues on next page)

(continued from previous page)

```
    bridge_ports bond1.30
    address 172.29.240.16
    netmask 255.255.252.0

# OpenStack Networking VLAN bridge
auto br-vlan
iface br-vlan inet manual
    bridge_stp off
    bridge_waitport 0
    bridge_fd 0
    bridge_ports bond1

# compute1 Network VLAN bridge
#auto br-vlan
#iface br-vlan inet manual
#    bridge_stp off
#    bridge_waitport 0
#    bridge_fd 0
#
# For tenant vlan support, create a veth pair to be used when the neutron
# agent is not containerized on the compute hosts. 'eth12' is the value used on
# the host_bind_override parameter of the br-vlan network section of the
# openstack_user_config example file. The veth peer name must match the value
# specified on the host_bind_override parameter.
#
# When the neutron agent is containerized it will use the container_interface
# value of the br-vlan network, which is also the same 'eth12' value.
#
# Create veth pair, do not abort if already exists
#    pre-up ip link add br-vlan-veth type veth peer name eth12 || true
# Set both ends UP
#    pre-up ip link set br-vlan-veth up
#    pre-up ip link set eth12 up
# Delete veth pair on DOWN
#    post-down ip link del br-vlan-veth || true
#    bridge_ports bond1 br-vlan-veth

# Storage bridge (optional)
#
# Only the COMPUTE and STORAGE nodes must have an IP address
# on this bridge. When used by infrastructure nodes, the
# IP addresses are assigned to containers which use this
# bridge.
#
auto br-storage
iface br-storage inet manual
    bridge_stp off
    bridge_waitport 0
    bridge_fd 0
```

(continues on next page)

(continued from previous page)

```
    bridge_ports bond0.20

# compute1 Storage bridge
#auto br-storage
#iface br-storage inet static
#    bridge_stp off
#    bridge_waitport 0
#    bridge_fd 0
#    bridge_ports bond0.20
#    address 172.29.244.16
#    netmask 255.255.252.0
```

Deployment configuration

Environment layout

The `/etc/openstack_deploy/openstack_user_config.yml` file defines the environment layout.

The following configuration describes the layout for this environment.

```
---
cidr_networks: &cidr_networks
  management: 172.29.236.0/22
  tunnel: 172.29.240.0/22
  storage: 172.29.244.0/22

used_ips:
  - "172.29.236.1,172.29.236.50"
  - "172.29.240.1,172.29.240.50"
  - "172.29.244.1,172.29.244.50"
  - "172.29.248.1,172.29.248.50"

global_overrides:
  cidr_networks: *cidr_networks
  internal_lb_vip_address: 172.29.236.9
  #
  # The below domain name must resolve to an IP address
  # in the CIDR specified in haproxy_keepalived_external_vip_cidr.
  # If using different protocols (https/http) for the public/internal
  # endpoints the two addresses must be different.
  #
  external_lb_vip_address: openstack.example.com
  management_bridge: "br-mgmt"
  provider_networks:
    - network:
        container_bridge: "br-mgmt"
        container_type: "veth"
        container_interface: "eth1"
        ip_from_q: "management"
        type: "raw"
```

(continues on next page)

(continued from previous page)

```
        group_binds:
          - all_containers
          - hosts
        is_management_address: true
    - network:
        container_bridge: "br-vxlan"
        container_type: "veth"
        container_interface: "eth10"
        ip_from_q: "tunnel"
        type: "vxlan"
        range: "1:1000"
        net_name: "vxlan"
        group_binds:
          - neutron_openvswitch_agent
    - network:
        container_bridge: "br-vlan"
        container_type: "veth"
        container_interface: "eth12"
        host_bind_override: "eth12"
        type: "flat"
        net_name: "physnet1"
        group_binds:
          - neutron_openvswitch_agent
    - network:
        container_bridge: "br-vlan"
        container_type: "veth"
        container_interface: "eth11"
        type: "vlan"
        range: "101:200,301:400"
        net_name: "physnet2"
        group_binds:
          - neutron_openvswitch_agent
    - network:
        container_bridge: "br-storage"
        container_type: "veth"
        container_interface: "eth2"
        ip_from_q: "storage"
        type: "raw"
        group_binds:
          - glance_api
          - cinder_api
          - cinder_volume
          - manila_share
          - nova_compute
          - ceph-mon
          - ceph-osd

###
### Infrastructure
```

(continues on next page)

(continued from previous page)

```
###

_infrastructure_hosts: &infrastructure_hosts
  infra1:
    ip: 172.29.236.11
  infra2:
    ip: 172.29.236.12
  infra3:
    ip: 172.29.236.13

# nova hypervisors
compute_hosts: &compute_hosts
  compute1:
    ip: 172.29.236.16
  compute2:
    ip: 172.29.236.17

ceph-osd_hosts:
  osd1:
    ip: 172.29.236.18
  osd2:
    ip: 172.29.236.19
  osd3:
    ip: 172.29.236.20

# galera, memcache, rabbitmq, utility
shared-infra_hosts: *infrastructure_hosts

# zookeeper
coordination_hosts: *infrastructure_hosts

# ceph-mon containers
ceph-mon_hosts: *infrastructure_hosts

# ceph-mds containers
ceph-mds_hosts: *infrastructure_hosts

# ganesha-nfs hosts
ceph-nfs_hosts: *infrastructure_hosts

# repository (apt cache, python packages, etc)
repo-infra_hosts: *infrastructure_hosts

# load balancer
# Ideally the load balancer should not use the Infrastructure hosts.
# Dedicated hardware is best for improved performance and security.
load_balancer_hosts: *infrastructure_hosts

###
```

(continues on next page)

(continued from previous page)

```
### OpenStack
###

# keystone
identity_hosts: *infrastructure_hosts

# cinder api services
storage-infra_hosts: *infrastructure_hosts

# cinder volume hosts (Ceph RBD-backed)
storage_hosts: *infrastructure_hosts

# glance
image_hosts: *infrastructure_hosts

# placement
placement-infra_hosts: *infrastructure_hosts

# nova api, conductor, etc services
compute-infra_hosts: *infrastructure_hosts

# heat
orchestration_hosts: *infrastructure_hosts

# horizon
dashboard_hosts: *infrastructure_hosts

# neutron server, agents (L3, etc)
network_hosts: *infrastructure_hosts

# ceilometer (telemetry data collection)
metering-infra_hosts: *infrastructure_hosts

# aodh (telemetry alarm service)
metering-alarm_hosts: *infrastructure_hosts

# gnocchi (telemetry metrics storage)
metrics_hosts: *infrastructure_hosts

# manila (share service)
manila-infra_hosts: *infrastructure_hosts
manila-data_hosts: *infrastructure_hosts

# ceilometer compute agent (telemetry data collection)
metering-compute_hosts: *compute_hosts
```

Environment customizations

The optionally deployed files in `/etc/openstack_deploy/env.d` allow the customization of Ansible groups. This allows the deployer to set whether the services will run in a container (the default), or on the host (on metal).

For a ceph environment, you can run the `cinder-volume` in a container. To do this you will need to create a `/etc/openstack_deploy/env.d/cinder.yml` file with the following content:

```
---
# This file contains an example to show how to set
# the cinder-volume service to run in a container.
#
# Important note:
# When using LVM or any iSCSI-based cinder backends, such as NetApp with
# iSCSI protocol, the cinder-volume service *must* run on metal.
# Reference: https://bugs.launchpad.net/ubuntu/+source/lxc/+bug/1226855

container_skel:
  cinder_volumes_container:
    properties:
      is_metal: false
```

User variables

The `/etc/openstack_deploy/user_variables.yml` file defines the global overrides for the default variables.

For this example environment, we configure a HA load balancer. We implement the load balancer (HAProxy) with an HA layer (Keepalived) on the infrastructure hosts. Your `/etc/openstack_deploy/user_variables.yml` must have the following content to configure HAProxy, Keepalived and Ceph:

```
---
# Because we have three haproxy nodes, we need
# to one active LB IP, and we use keepalived for that.
# These variables must be defined when external_lb_vip_address or
# internal_lb_vip_address is set to FQDN.
## Load Balancer Configuration (haproxy/keepalived)
haproxy_keepalived_external_vip_cidr: "<external_ip_address>/<netmask>"
haproxy_keepalived_internal_vip_cidr: "172.29.236.9/32"
haproxy_keepalived_external_interface: ens2
haproxy_keepalived_internal_interface: br-mgmt

## Ceph cluster fsid (must be generated before first run)
## Generate a uuid using: python -c 'import uuid; print(str(uuid.uuid4()))'
generate_fsid: false
fsid: 116f14c4-7fe1-40e4-94eb-9240b63de5c1 # Replace with your generated UUID

## ceph-ansible settings
## See https://github.com/ceph/ceph-ansible/tree/master/group_vars for
## additional configuration options available.
monitor_address_block: "{{ cidr_networks.storage }}"
```

(continues on next page)

(continued from previous page)

```
public_network: "{{ cidr_networks.storage }}"
cluster_network: "{{ cidr_networks.storage }}"
journal_size: 10240 # size in MB
# ceph-ansible automatically creates pools & keys for OpenStack services
openstack_config: true
cinder_ceph_client: cinder
glance_ceph_client: glance
glance_default_store: rbd
glance_rbd_store_pool: images
nova_libvirt_images_rbd_pool: vms

cinder_backends:
  rbd_volumes:
    volume_driver: cinder.volume.drivers.rbd.RBDDriver
    rbd_pool: volumes
    rbd_ceph_conf: /etc/ceph/ceph.conf
    rbd_store_chunk_size: 8
    volume_backend_name: rbddriver
    rbd_user: "{{ cinder_ceph_client }}"
    rbd_secret_uuid: "{{ cinder_ceph_client_uuid }}"
    report_discard_supported: true
```

Using radosgw as a drop-in replacement for Swift

OpenStack-Ansible gives you the option of deploying radosgw as a drop-in replacement for native OpenStack Swift.

In particular, the `ceph-rgw-install.yml` playbook (which includes `ceph-rgw-keystone-setup.yml`) will deploy radosgw to any `ceph-rgw` hosts, and create a corresponding Keystone `object-store` service catalog entry. The service endpoints do contain the `AUTH_%(tenant_id)s` prefix just like in native Swift, so public read ACLs and temp URLs will work just like they do in Swift.

By default, OSA enables *only* the Swift API in radosgw.

Adding S3 API support

You may want to enable the default radosgw S3 API, in addition to the Swift API. In order to do so, you need to override the `ceph_conf_overrides_rgw` variable in `user_variables.yml`. Below is an example configuration snippet:

Note

Mentioned below overrides are default ones and will be applied to *ceph-rgw* group

```
---
ceph_rgw_client_name: "client.rgw.{{ rgw_zone | default('default') }}.{{␣
↪hostvars[inventory_hostname]['ansible_facts']['hostname']}}.rgw0"
ceph_conf_overrides_rgw: |-
  {{
```

(continues on next page)

(continued from previous page)

```
    {
      ceph_rgw_client_name: {
        'rgw_keystone_url': keystone_service_adminuri,
        'rgw_keystone_api_version': 3,
        'rgw_keystone_admin_user': radosgw_admin_user,
        'rgw_keystone_admin_password': radosgw_admin_password,
        'rgw_keystone_admin_project': radosgw_admin_tenant,
        'rgw_keystone_admin_domain': 'default',
        'rgw_keystone_accepted_roles': 'member, admin, swiftoperator',
        'rgw_keystone_implicit_tenants': 'true',
        'rgw_swift_account_in_url': 'true',
        'rgw_swift_versioning_enabled': 'true',
        'rgw_enable_apis': 'swift, s3',
        'rgw_s3_auth_use_keystone': 'true'
      }
    }
  }}

###
### Backend TLS
###

# Ceph configuration options to enable TLS on ceph-rgw
radosgw_frontend_ssl_certificate: "{{ ceph_rgw_backend_ssl is truthy |␣
↪ternary(ceph_rgw_ssl_cert, '') }}"
# Ceph-ansible requires to include private key in `radosgw_frontend_ssl_
↪certificate`
# which is not possible with ansible-role-pki.
# That is why `ssl_private_key` is defined in `radosgw_frontend_options`.
radosgw_frontend_options: "{{ ceph_rgw_backend_ssl is truthy | ternary('ssl_
↪private_key=' + ceph_rgw_ssl_key, '') }}"

# Define if communication between haproxy and service backends should be
# encrypted with TLS.
ceph_rgw_backend_ssl: "{{ openstack_service_backend_ssl | default(False) }}"

# Storage location for SSL certificate authority
ceph_rgw_pki_dir: "{{ openstack_pki_dir | default('/etc/openstack_deploy/pki
↪') }}"

# Delegated host for operating the certificate authority
ceph_rgw_pki_setup_host: "{{ openstack_pki_setup_host | default('localhost') }
↪}"

# ceph_rgw server certificate
ceph_rgw_pki_keys_path: "{{ ceph_rgw_pki_dir ~ '/certs/private/' }}"
ceph_rgw_pki_certs_path: "{{ ceph_rgw_pki_dir ~ '/certs/certs/' }}"
ceph_rgw_pki_intermediate_cert_name: "{{ openstack_pki_service_intermediate_
↪cert_name | default('ExampleCorpIntermediate') }}"
```

(continues on next page)

(continued from previous page)

```
ceph_rgw_pki_regen_cert: ''
ceph_rgw_pki_san: "{{ openstack_pki_san | default('DNS:' ~ ansible_facts[
↪'hostname'] ~ ',IP:' ~ management_address) }}"
ceph_rgw_pki_certificates:
  - name: "ceph_rgw_{{ ansible_facts['hostname'] }}"
    provider: ownca
    cn: "{{ ansible_facts['hostname'] }}"
    san: "{{ ceph_rgw_pki_san }}"
    signed_by: "{{ ceph_rgw_pki_intermediate_cert_name }}"

# ceph_rgw destination files for SSL certificates
ceph_rgw_ssl_cert: /etc/ceph/ceph-rgw.pem
ceph_rgw_ssl_key: /etc/ceph/ceph-rgw.key

# Installation details for SSL certificates
ceph_rgw_pki_install_certificates:
  - src: "{{ ceph_rgw_user_ssl_cert | default(ceph_rgw_pki_certs_path ~ 'ceph_
↪rgw_' ~ ansible_facts['hostname'] ~ '-chain.crt') }}"
    dest: "{{ ceph_rgw_ssl_cert }}"
    owner: "ceph"
    group: "ceph"
    mode: "0644"
  - src: "{{ ceph_rgw_user_ssl_key | default(ceph_rgw_pki_keys_path ~ 'ceph_
↪rgw_' ~ ansible_facts['hostname'] ~ '.key.pem') }}"
    dest: "{{ ceph_rgw_ssl_key }}"
    owner: "ceph"
    group: "ceph"
    mode: "0600"

# Define user-provided SSL certificates
#ceph_rgw_user_ssl_cert: <path to cert on ansible deployment host>
#ceph_rgw_user_ssl_key: <path to cert on ansible deployment host>
```

You may also want to add the `rgw_dns_name` option if you want to enable bucket hostnames with the S3 API.

Integrate radosgw into your Telemetry

The telemetry (and in consequence accounting) for radosgw as object-storage will not work out of the box. You need to change different parts of your OpenStack and Ceph setup to get it up and running.

Ceilometer Changes

Ceilometer needs additional pip packages to talk to Ceph Rados Gateway. To install it, edit the default ceilometer_pip_packages in your user_variables.yml file:

```
ceilometer_pip_packages:
    - ceilometer
    - ceilometermiddleware
    - cryptography
```

(continues on next page)

(continued from previous page)

```
    - gnocchiclient
    - libvirt-python
    - PyMySQL
    - pymongo
    - python-memcached
    - tooz
    - warlock
    - requests-aws>=0.1.4 #https://github.com/openstack/ceilometer/blob/
↪stable/pike/test-requirements.txt
```

You also have to configure Ceilometer to actually query radosgw. When your ceilometer isnt configured to poll everything, add these pollsters to your polling.yml file:

```
- name: radosgw_pollsters
  interval: 1200
  meters:
    -  radosgw.containers.objects
    -  radosgw.containers.objects.size
    -  radosgw.objects
    -  radosgw.objects.size
    -  radosgw.objects.containers
    -  radosgw.usage
```

Add them also to your pipeline:

```
- name: radosgw_source
  interval: 60
  meters:
    - "rgw.objects"
    - "rgw.objects.size"
    - "rgw.objects.containers"
    - "rgw.api.request"
    - "rgw.containers.objects"
    - "rgw.containers.objects.size"
  sinks:
    - meter_sink
```

Declare Ceph Rados Gateway as object-store in your ceilometer.conf file by adding this to your user_variables.yml file:

```
ceilometer_ceilometer_conf_overrides:
  service_types:
    radosgw: object-store
  rgw_admin_credentials:
    access_key: XXX
    secret_key: XXX
```

The required user and credentials is created by this command:

```
radosgw-admin user create --uid admin --display-name "admin user" --caps
↪"usage=read,write;metadata=read,write;users=read,write;buckets=read,write"
```

To get your credentials, execute:

```
radosgw-admin user info --uid admin | jq '.keys'
```

Ceph Changes

The required changes are described in the documentation of Ceilometer. This is just a sum up. In your ceph.conf add:

```
[client.radosgw.gateway]
rgw enable usage log = true
rgw usage log tick interval = 30
rgw usage log flush threshold = 1024
rgw usage max shards = 32
rgw usage max user shards = 1
```

Security settings and SSL certificates

This chapter contains information to configure specific security settings for your OpenStack-Ansible cloud.

For understanding security design, please see *Security*.

Securing services with SSL certificates

The OpenStack Security Guide recommends providing secure communication between various services in an OpenStack deployment. The OpenStack-Ansible project currently offers the ability to configure SSL certificates for secure communication between services:

All public endpoints reside behind HAProxy, resulting in the only certificate management for externally visible https services are those for HAProxy. Certain internal services such as RabbitMQ also require proper SSL configuration.

When deploying with OpenStack-Ansible, you can either use self-signed certificates that are generated during the deployment process or provide SSL certificates, keys, and CA certificates from your own trusted certificate authority. Highly secured environments use trusted, user-provided certificates for as many services as possible.

Note

Perform all SSL certificate configuration in `/etc/openstack_deploy/user_variables.yml` file. Do not edit the playbooks or roles themselves.

OpenStack-Ansible uses an ansible role ansible_role_pki as a general tool to manage and install self-signed and user provided certificates.

Note

The OpenStack-Ansible example configurations are designed to be minimal examples and in test or development use-cases will set `external_lb_vip_address` to the IP address of the HAProxy

external endpoint. For a production deployment it is advised to set `external_lb_vip_address` to be the FQDN which resolves via DNS to the IP of the external endpoint.

Self-signed certificates

Self-signed certificates enable you to start quickly and encrypt data in transit. However, they do not provide a high level of trust for public endpoints in highly secure environments. By default, self-signed certificates are used in OpenStack-Ansible. When self-signed certificates are used, certificate verification is automatically disabled.

Self-signed certificates can play an important role in securing internal services within the OpenStack-Ansible deployment, as they can only be issued by the private CA associated with the deployment. Using mutual TLS between backend services such as RabbitMQ and MariaDB with self-signed certificates and a robust CA setup can ensure that only correctly authenticated clients can connect to these internal services.

Generating and regenerating self-signed certificate authorities

A self-signed certificate authority is generated on the deploy host during the first run of the playbook.

To regenerate the certificate authority you must set the `openstack_pki_regen_ca` variable to either the name of the root CA or intermediate CA you wish or regenerate, or to `true` to regenerate all self-signed certificate authorities.

```
# openstack-ansible -e "openstack_pki_regen_
↪ca=ExampleCorpIntermediate" certificate-authority.yml
```

Take particular care not to regenerate Root or Intermediate certificate authorities in a way that may invalidate existing server certificates in the deployment. It may be preferable to create new Intermediate CA certificates rather than regenerate existing ones in order to maintain existing chains of trust.

Generating and regenerating self-signed certificates

Self-signed certificates are generated for each service during the first run of the playbook.

To regenerate a new self-signed certificate for a service, you must set the `<servicename>_pki_regen_cert` variable to true in one of the following ways:

- To force a self-signed certificate to regenerate, you can pass the variable to `openstack-ansible` on the command line:

```
# openstack-ansible -e "haproxy_pki_regen_cert=true" haproxy-install.yml
```

- To force a self-signed certificate to regenerate with every playbook run, set the appropriate regeneration option to `true`. For example, if you have already run the `haproxy` playbook, but you want to regenerate the self-signed certificate, set the `haproxy_pki_regen_cert` variable to `true` in the `/etc/openstack_deploy/user_variables.yml` file:

```
haproxy_pki_regen_cert: true
```

Generating and regenerating self-signed user certificates

Self-signed user certificates are generated but not installed for services outside of OpenStack-Ansible. These user certificates are signed by the same self-signed certificate authority as is used by OpenStack services but are intended to be used by user applications.

To generate user certificates, define a variable with the prefix `user_pki_certificates_` in the `/etc/openstack_deploy/user_variables.yml` file.

Example

```
user_pki_certificates_example:
  - name: "example"
    provider: ownca
    cn: "example.com"
    san: "DNS:example.com,IP:x.x.x.x"
    signed_by: "{{ openstack_pki_service_intermediate_cert_name }}"
    key_usage:
      - digitalSignature
      - keyAgreement
    extended_key_usage:
      - serverAuth
```

Generate the certificate with the following command:

```
# openstack-ansible certificate-generate.yml
```

To regenerate a new self-signed certificate for a service, you must set the `user_pki_regen_cert` variable to true in one of the following ways:

- To force a self-signed certificate to regenerate, you can pass the variable to `openstack-ansible` on the command line:

  ```
  # openstack-ansible -e "user_pki_regen_cert=true" certificate-generate.yml
  ```

- To force a self-signed certificate to regenerate with every playbook run, set the `user_pki_regen_cert` variable to `true` in the `/etc/openstack_deploy/user_variables.yml` file:

  ```
  user_pki_regen_cert: true
  ```

User-provided certificates

For added trust in highly secure environments, you can provide your own SSL certificates, keys, and CA certificates. Acquiring certificates from a trusted certificate authority is outside the scope of this document, but the Certificate Management section of the Linux Documentation Project explains how to create your own certificate authority and sign certificates.

Use the following process to deploy user-provided SSL certificates in OpenStack-Ansible:

1. Copy your SSL certificate, key, and CA certificate files to the deployment host.
2. Specify the path to your SSL certificate, key, and CA certificate in the `/etc/openstack_deploy/user_variables.yml` file.
3. Run the playbook for that service.

HAProxy example

The variables to set which provide the path on the deployment node to the certificates for HAProxy configuration are:

```
haproxy_user_ssl_cert: /etc/openstack_deploy/ssl/example.com.crt
haproxy_user_ssl_key: /etc/openstack_deploy/ssl/example.com.key
haproxy_user_ssl_ca_cert: /etc/openstack_deploy/ssl/ExampleCA.crt
```

RabbitMQ example

To deploy user-provided certificates for RabbitMQ, copy the certificates to the deployment host, edit the `/etc/openstack_deploy/user_variables.yml` file and set the following three variables:

```
rabbitmq_user_ssl_cert:    /etc/openstack_deploy/ssl/example.com.crt
rabbitmq_user_ssl_key:     /etc/openstack_deploy/ssl/example.com.key
rabbitmq_user_ssl_ca_cert: /etc/openstack_deploy/ssl/ExampleCA.crt
```

Then, run the playbook to apply the certificates:

```
# openstack-ansible rabbitmq-install.yml
```

The playbook deploys your user-provided SSL certificate, key, and CA certificate to each RabbitMQ container.

The process is identical for the other services. Replace *rabbitmq* in the preceding configuration variables with *horizon*, *haproxy*, or *keystone*, and then run the playbook for that service to deploy user-provided certificates to those services.

Certbot certificates

The HAProxy ansible role supports using Certbot to automatically deploy trusted SSL certificates for the public endpoint. Each HAProxy server will individually request a SSL certificate using Certbot.

Certbot defaults to using Lets Encrypt as the Certificate Authority, other Certificate Authorities can be used by setting the `haproxy_ssl_letsencrypt_certbot_server` variable in the `/etc/openstack_deploy/user_variables.yml` file:

```
haproxy_ssl_letsencrypt_certbot_server: "https://acme-staging-v02.api.
↪letsencrypt.org/directory"
```

The http-01 type challenge is used by Certbot to deploy certificates so it is required that the public endpoint is accessible directly by the Certificate Authority.

Deployment of certificates using Lets Encrypt has been validated for OpenStack-Ansible using Ubuntu 24.04 (Noble Numbat). Other distributions should work but are not tested.

To deploy certificates with Certbot, add the following to `/etc/openstack_deploy/user_variables.yml` to enable the Certbot function in the HAProxy ansible role, and to create a new backend service called `certbot` to service http-01 challenge requests.

```
haproxy_ssl: true
haproxy_ssl_letsencrypt_enable: true
haproxy_ssl_letsencrypt_email: "email.address@example.com"
```

TLS for HAProxy Internal Virtual IP (VIP)

As well as load balancing public endpoints, HAProxy is also used to load balance internal connections.

By default, OpenStack-Ansible does not secure connections to the internal VIP. To enable this you must set the following variables in the `/etc/openstack_deploy/user_variables.yml` file:

```
openstack_service_adminuri_proto: https
openstack_service_internaluri_proto: https

haproxy_ssl_all_vips: true
```

Run all playbooks to configure HAProxy and OpenStack services.

When enabled HAProxy will use the same TLS certificate on all interfaces (internal and external). It is not currently possible in OpenStack-Ansible to use different self-signed or user-provided TLS certificates on different HAProxy interfaces.

The only way to use a different TLS certificates on the internal and external VIP is to use Certbot.

Enabling TLS on the internal VIP for existing deployments will cause some downtime, this is because HAProxy only listens on a single well known port for each OpenStack service and OpenStack services are configured to use http or https. This means once HAProxy is updated to only accept HTTPS connections, the OpenStack services will stop working until they are updated to use HTTPS.

To avoid downtime, it is recommended to enable `openstack_service_accept_both_protocols` until all services are configured correctly. It allows HAProxy frontends to listen on both HTTP and HTTPS.

TLS for HAProxy Backends

Communication between HAProxy and service backends can be encrypted. Currently it is disabled by default. It can be enabled for all services by setting the following variable:

```
openstack_service_backend_ssl: true
```

There is also an option to enable it only for individual services:

```
keystone_backend_ssl: true
neutron_backend_ssl: true
```

By default, self-signed certificates will be used to secure traffic but user-provided certificates are also supported.

TLS for Live Migrations

Live migration of VMs using SSH is deprecated and the OpenStack Nova Docs recommends using the more secure native TLS method supported by QEMU. The default live migration method used by OpenStack-Ansible has been updated to use TLS migrations.

QEMU-native TLS requires all compute hosts to accept TCP connections on port 16514 and port range 49152 to 49261.

It is not possible to have a mixed state of some compute nodes using SSH and some using TLS for live migrations, as this would prevent live migrations between the compute nodes.

There are no issues enabling TLS live migration during an OpenStack upgrade, as long as you do not need to live migrate instances during the upgrade. If you you need to live migrate instances during an upgrade, enable TLS live migrations before or after the upgrade.

To force the use of SSH instead of TLS for live migrations you must set the `nova_libvirtd_listen_tls` variable to `0` in the `/etc/openstack_deploy/user_variables.yml` file:

```
nova_libvirtd_listen_tls: 0
```

TLS for VNC

When using VNC for console access there are 3 connections to secure, client to HAProxy, HAProxy to noVNC Proxy and noVNC Proxy to Compute nodes. The OpenStack Nova Docs for remote console access cover console security in much more detail.

In OpenStack-Ansible TLS to HAProxy is configured in HAProxy, TLS from HAProxy to noVNC is not currently enabled and TLS from nVNC to Compute nodes is enabled by default.

Changes will not apply to any existing running guests on the compute node, so this configuration should be done before launching any instances. For existing deployments it is recommended that you migrate instances off the compute node before enabling.

To help with the transition from unencrypted VNC to VeNCrypt, initially noVNC proxy auth scheme allows for both encrypted and unencrypted sessions using the variable *nova_vencrypt_auth_scheme*. This will be restricted to VeNCrypt only in future versions of OpenStack-Ansible.

```
nova_vencrypt_auth_scheme: "vencrypt,none"
```

To not encrypt data from noVNC proxy to Compute nodes you must set the `nova_qemu_vnc_tls` variable to `0` in the `/etc/openstack_deploy/user_variables.yml` file:

```
nova_qemu_vnc_tls: 0
```

Security Headers

Security headers are HTTP headers that can be used to increase the security of a web application by restricting what modern browsers are able to run.

In OpenStack-Ansible, security headers are implemented in HAProxy as all the public endpoints reside behind it.

The following headers are enabled by default on all the HAProxy interfaces that implement TLS, but only for the Horizon service. The security headers can be implemented on other HAProxy services, but only services used by browsers will make use of the headers.

HTTP Strict Transport Security

The OpenStack TLS Security Guide recommends that all production deployments use HTTP strict transport security (HSTS).

By design, this header is difficult to disable once set. It is recommended that during testing you set a short time of 1 day and after testing increase the time to 1 year.

To change the default max age to 1 day, override the variable `haproxy_security_headers_max_age` in the `/etc/openstack_deploy/user_variables.yml` file:

```
haproxy_security_headers_max_age: 86400
```

If you would like your domain included in the HSTS preload list, which is built into browsers, before submitting your request to be added to the HSTS preload list you must add the `preload` token to your response header. The `preload` token indicates to the maintainers of HSTS preload list that you are happy to have your site included.

```
- "http-response set-header Strict-Transport-Security \"max-age={{ haproxy_
↪security_headers_max_age }}; includeSubDomains; preload;\""
```

X-Content-Type-Options

The `X-Content-Type-Options` header prevents MIME type sniffing.

This functionality can be changed by overriding the list of headers in `haproxy_security_headers` variable in the `/etc/openstack_deploy/user_variables.yml` file.

Referrer Policy

The `Referrer-Policy` header controls how much referrer information is sent with requests. It defaults to `same-origin`, which does not send the origin path for cross-origin requests.

This functionality can be changed by overriding the list of headers in `haproxy_security_headers` variable in the `/etc/openstack_deploy/user_variables.yml` file.

Permission Policy

The `Permissions-Policy` header allows you to selectively enable, disable or modify the use of browser features and APIs, previously known as Feature Policy.

By default this header is set to block access to all features apart from the following from the same origin; fullscreen, clipboard read, clipboard write and spatial navigation.

This functionality can be changed by overriding the list of headers in `haproxy_security_headers` variable in the `/etc/openstack_deploy/user_variables.yml` file.

Content Security Policy (CSP)

The `Content-Security-Policy` header allows you to control what resources a browser is allowed to load for a given page, which helps to mitigate the risks from Cross-Site Scripting (XSS) and data injection attacks.

By default, the Content Security Policy (CSP) enables a minimum set of resources to allow Horizon to work, which includes access the Nova console. If you require access to other resources these can be set by overriding the `haproxy_security_headers_csp` variable in the `/etc/openstack_deploy/user_variables.yml` file.

Report Only

Implementing CSP could lead to broken content if a browser is blocked from accessing certain resources, therefore it is recommended that when testing CSP you use the `Content-Security-Policy-Report-Only` header, instead of `Content-Security-Policy`, this reports CSP violations to the browser console, but does not enforce the policy.

To set the CSP policy to report only by overriding the `haproxy_security_headers_csp_report_only` variable to `true` in the `/etc/openstack_deploy/user_variables.yml` file:

```
haproxy_security_headers_csp_report_only: true
```

Reporting Violations

It is recommended that you monitor attempted CSP violations in production, this is achieved by setting the `report-uri` and `report-to` tokens.

Federated Login

When using federated login you will need to override the default Content Security Policy to allow access to your authorisation server by overriding the `haproxy_horizon_csp` variable in the `/etc/openstack_deploy/user_variables.yml` file:

```
haproxy_horizon_csp: >
  http-response set-header Content-Security-Policy "
  default-src 'self';
  frame-ancestors 'self';
  form-action 'self' {{ external_lb_vip_address }}:5000 <YOUR-AUTHORISATION-
↪SERVER-ORIGIN>;
  upgrade-insecure-requests;
  style-src 'self' 'unsafe-inline';
  script-src 'self' 'unsafe-inline' 'unsafe-eval';
  child-src 'self' {{ external_lb_vip_address }}:{{ nova_spice_html5proxy_
↪base_port }} {{ external_lb_vip_address }}:{{ nova_novncproxy_port }} {{␣
↪external_lb_vip_address }}:{{ nova_serialconsoleproxy_port }};
  frame-src 'self' {{ external_lb_vip_address }}:{{ nova_spice_html5proxy_
↪base_port }} {{ external_lb_vip_address }}:{{ nova_novncproxy_port }} {{␣
↪external_lb_vip_address }}:{{ nova_serialconsoleproxy_port }};
  "
```

It is also possible to set specific security headers for Skyline.

```
haproxy_skyline_csp: ...
```

security.txt

security.txt is a proposed IETF standard to allow independent security researchers to easily report vulnerabilities. The standard defines that a text file called `security.txt` should be found at /.well-known/security.txt. For legacy compatibility reasons the file might also be placed at /security.txt.

In OpenStack-Ansible, `security.txt` is implemented in HAProxy as all public endpoints reside behind it. It defaults to directing any request paths that end with `/security.txt` to the text file using an ACL rule in HAProxy.

Enabling security.txt

Use the following process to add a `security.txt` file to your deployment using OpenStack-Ansible:

1. Write the contents of the `security.txt` file in accordance with the standard.
2. Define the contents of `security.txt` in the variable `haproxy_security_txt_content` in the `/etc/openstack_deploy/user_variables.yml` file:

```
haproxy_security_txt_content: |
  # This is my example security.txt file
  # Please see https://securitytxt.org/ for details of the␣
↪specification of this file
```

3. Update HAProxy

```
# openstack-ansible haproxy-install.yml
```

Advanced security.txt ACL

In some cases you may need to change the HAProxy ACL used to redirect requests to the `security.txt` file, such as adding extra domains.

The HAProxy ACL is updated by overriding the variable `haproxy_map_entries` inside `haproxy_security_txt_service`.

Apply ansible-hardening

The `ansible-hardening` role is applicable to physical hosts within an OpenStack-Ansible deployment that are operating as any type of node, infrastructure or compute. By default, the role is enabled. You can disable it by changing the value of the `apply_security_hardening` variable in the `user_variables.yml` file to `false`:

```
apply_security_hardening: false
```

You can apply security hardening configurations to an existing environment or audit an environment by using a playbook supplied with OpenStack-Ansible:

```
# Apply security hardening configurations
  openstack-ansible openstack.osa.security_hardening

# Perform a quick audit by using Ansible's check mode
  openstack-ansible --check openstack.osa.security_hardening
```

For more information about the security configurations, see the security hardening role documentation.

Deployment Host Hardening

You can extend security hardening to the deployment host by defining the `security_host_group` variable in your `openstack_user_variables` file. Include `localhost` along with your other hosts, like this:

```
security_host_group: localhost, hosts
```

Then apply the hardening with:

```
openstack-ansible openstack.osa.security_hardening
```

Or alternatively, you can also supply this variable as extra variable during runtime, for example:

```
openstack-ansible openstack.osa.security_hardening -e security_host_
↪group=localhost
```

Warning

After applying security hardening, root login via password will be disabled. Make sure you configure SSH key authentication or set up a non-root user with sudo privileges before applying the changes, otherwise you may lose access to the host.

Including the deployment host can be useful to reduce its attack surface and ensure that the host running OpenStack-Ansible follows the same security best practices as your other nodes.

Hiding Secrets in OpenStack-Ansible

OpenStack-Ansible roles use variables like `_oslodb_setup_nolog`, `_service_setup_nolog`, and `_oslomsg_nolog` to control whether task output is hidden in logs.

By default, this prevents sensitive values (such as passwords) from being written to log files. Disabling these variables can make debugging easier, but it also risks exposing secrets in plain text.

Warning

Use them with caution: keep logging enabled for troubleshooting, but remember that passwords may appear in the logs if protection is turned off.

Running as non-root user

Deployers do not have to use `root` user accounts on deploy or target hosts. This approach works out of the box by leveraging Ansible privilege escalation.

Deployment hosts

You can avoid usage of the `root` user on a deployment by following these guidelines:

1. Clone OpenStack-Ansible repository to home user directory. It means, that instead of `/opt/openstack-ansible` repository will be in `~/openstack-ansible`.
2. Use custom path for `/etc/openstack_deploy` directory. You can place OpenStack-Ansible configuration directory inside user home directory. For that you will need to define the following environment variable:

   ```
   export OSA_CONFIG_DIR="${HOME}/openstack_deploy"
   ```

3. If you want to keep basic ansible logging, you need either to create `/openstack/log/ansible-logging/` directory and allow user to write there, or define the following environment variable:

```
export ANSIBLE_LOG_PATH="${HOME}/ansible-logging/ansible.log"
```

Note

You can also add the environment variable to `user.rc` file inside openstack_deploy folder (`${OSA_CONFIG_DIR}/user.rc`). `user.rc` file is sourced each time you run `openstack-ansible` binary.

4. Initial bootstrap of OpenStack-Ansible using ./scripts/bootstrap-ansible.sh script still should be done either as the `root` user or escalate privileges using `sudo` or `su`.

Destination hosts

It is also possible to use non-root user for Ansible authentication on destination hosts. However, this user must be able to escalate privileges using Ansible privilege escalation.

Note

You can add environment variables from that section to `user.rc` file inside openstack_deploy folder (`${OSA_CONFIG_DIR}/user.rc`). `user.rc` file is sourced each time you run `openstack-ansible` binary.

There are also couple of additional things which you might want to consider:

1. Provide `--become` flag each time your run a playbook or ad-hoc command. Alternatively, you can define the following environment variable:

   ```
   export ANSIBLE_BECOME="true"
   ```

2. Override Ansible temporary path if LXC containers are used. The ansible connection from the physical host to the LXC container passes environment variables from the host. This means that Ansible attempts to use the same temporary folder in the LXC container as it would on the host, relative to the non-root user ${HOME} directory. This will not exist inside the container and another path must be used instead.

   You can do that following in multiple ways:

   a. Define `ansible_remote_tmp: /tmp` in user_variables.yml

   b. Define the following environment variable:

   ```
   export ANSIBLE_LOCAL_TEMP="/tmp"
   ```

3. Define the user that will be used for for connections from the deploy host to the ansible target hosts. In case the user is the same for all hosts in your deployment, you can do it in one of following ways:

   a. Define `ansible_user: <USER>` in user_variables.yml

   b. Define the following environment variable:

   ```
   export ANSIBLE_REMOTE_USER="<USER>"
   ```

If the user differs from host to host, you can leverage group_vars or host_vars. More information on how to use that can be found in the *overrides guide*

Password Rotation

Warning

The playbooks do not guarantee password rotation with zero downtime in an existing environment. Downtime for the service is expected between password is reset on backend and services are restarted to apply new value.

All service passwords are defined and stored as Ansible variables in OpenStack-Ansible. This allows the operator to store passwords in an encrypted format using Ansible Vault or define them as a lookup to SOPS or OpenBao.

Typical password change processes include the following steps:

1. Define a new password in Ansible variables (or where defined lookup is pointing to).
2. Change password on an infrastructure backend (i.e. MariaDB, RabbitMQ, etc.).
3. Update service configuration file with the new password.
4. Restart the service to apply new configuration.

Due to the variety of methods which can be used for storing and defining Ansible variables, we will leave the process of changing password definitions out of scope of this article, and will focus solely on the process of applying new passwords to the environment.

Applying new passwords to the service

A typical service has a set of passwords for authentication in infra backends, which include, but not limited to:

- Keystone
- MariaDB
- RabbitMQ

As a service downtime is expected after a password is changed (which generally happens at the very beginning of each role) until the service is restarted, it is important to ensure that playbook will execute as fast as possible. For that, we will use a set of specific tags and variables for each backend individually.

For following examples, we will take Nova as a sample service, as Nova may struggle from playbook runtime the most due to amount of hosts which needs to be updated.

As a common technique, we will disable `serial` execution, which is enabled by default, as the password for the backend will be reset during the runtime for the first host, so rest will not be able to communicate normally regardless of used `serial`.

Another common thing among all sections below is usage of `post-install` tag. It has been introduced in 2024.1 (Caracal) release and is applied only to `<service>_post_install.yml` tasks, where templating of config files is happening.

Changing Keystone password for the service

In order to trigger a password update in Keystone for the service, we need to supply `service_update_password` variable. To execute the rotation of the Keystone password for a service like Nova, you will need to execute playbook like this:

```
openstack-ansible openstack.osa.nova -e nova_conductor_serial=100% -e nova_
↪compute_serial=100% \
    -e service_update_password=true --tags common-service,post-install
```

Changing MariaDB password for the service

To execute the rotation of the MariaDB password for a service like Nova, you will need to execute playbook like this:

```
openstack-ansible openstack.osa.nova -e nova_conductor_serial=100% -e nova_
↪compute_serial=100% \
    --tags common-db,post-install
```

Changing RabbitMQ password for the service

To execute the rotation of the MariaDB password for a service like Nova, you will need to execute playbook like this:

```
openstack-ansible openstack.osa.nova -e nova_conductor_serial=100% -e nova_
↪compute_serial=100% \
    --tags common-mq,post-install
```

Changing all passwords for the service at once

It is worth to mention, that operator can combine them together to perform rotation of passwords for all backends at the same time. While it will increase playbook runtime (and thus a downtime), it will still be more efficient when all passwords need to be changed anyway.

To update all passwords mentioned in previous sections, we will simply combine all used tags and variables:

```
openstack-ansible openstack.osa.nova -e nova_conductor_serial=100% -e nova_
↪compute_serial=100% \
    -e service_update_password=true --tags common-service,common-db,common-mq,
↪post-install
```

Source overriding examples

There are situations where a deployer want to override sources with its own fork.

This chapter gives case-by-case examples on how to override the default sources.

Overriding Ansible version

Overriding the default Ansible version is not recommended, as each branch of OpenStack-Ansible has been built with the a specific Ansible version in mind, and many Ansible changes are neither backwards nor forward compatible.

The `bootstrap-ansible.sh` script installs Ansible, and uses a variable `ANSIBLE_PACKAGE` to describe which version to install.

For example to install ansible version 2.5.0:

```
$ export ANSIBLE_PACKAGE="ansible==2.5.0"
```

Installing directly from git is also supported. For example, from the tip of Ansible development branch:

```
$ export ANSIBLE_PACKAGE="git+https://github.com/ansible/ansible@devel
↪#egg=ansible"
```

Overriding the roles

Overriding the role file has been explained in the reference guide, on the *Adding new or overriding roles in your OpenStack-Ansible installation* section.

Overriding other upstream projects source code

All the upstream repositories used are defined in the `openstack-ansible` integrated repository, in the `inventory/group_vars/<service_group>/source_git.yml` file.

For example, if you want to override `glance` repository with your own, you need to define the following:

```
glance_git_repo: https://<your git repo>
glance_git_install_branch: <your git branch or commit SHA>
glance_git_project_group: glance_all
```

Please note, for this glance example, that you do not need to edit the `inventory/group_vars/glance_all/source_git.yml` file.

Instead, the usual overrides mechanism can take place, and you can define these 3 variables in a `user_*.yml` file. See also the *Overriding default configuration* page.

Note

These variables behave a little differently than standard ansible precedence, because they are also consumed by a custom lookup plugin.

The `py_pkgs lookup` will ignore all _git_ variables unless the `_git_repo` variable is present.

So even if you only want to override the `_git_install_branch` for a repository, you should also define the `_git_repo` variable in your user variables.

Telemetry with Gnocchi, Ceph and Redis example

The default OpenStack-Ansible installation configures gnocchi to use a file as storage backend. When you already have a pre-installed ceph, you can use this as backend for gnocchi. This documentation will guide you how to set up gnocchi to use your ceph as storage backend.

Ceph as metric storage

```
gnocchi_storage_driver: ceph
```

You have to add some pip packages to your gnocchi setup:

```
gnocchi_pip_packages:
- cryptography
- gnocchiclient
# this is what we want:
#  - "gnocchi[mysql,ceph,ceph_alternative_lib,redis,keystone]"
# but as there is no librados >=12.2 pip package we have to first install␣
↪ceph without alternative support
# after adding the ceph repo to gnocchi container, python-rados>=12.2.0 is␣
↪installed and linked automatically
# and gnocchi will automatically take up the features present in the used␣
↪rados lib.
- "gnocchi[mysql,ceph,redis,keystone]"
- keystonemiddleware
- python-memcached
```

But when your setup grows, gnocchi might slow down or block your ceph installation. You might experience slow requests and stuck PGs in your Ceph. As this might have multiple causes, take a look at the presentations linked in the *Performance Tests for Gnocchi* section. They also include various parameters which you might tune.

Redis as measure storage

One solution to possible performance problems is to use an incoming measure storage for your gnocchi installation. The supported storage systems are:

- File (default)
- Ceph (preferred)
- OpenStack Swift
- Amazon S3
- Redis

When your Swift API endpoint uses Ceph as a backend, the only one left for this setup is Redis.

So first of all setup a redis server/cluster, e.g. with this ansible role. Next, you have to configure Gnocchi with OpenStack-Ansible to use the Redis Cluster as incoming storage:

```
gnocchi_conf_overrides:
  incoming:
    driver: redis
```

(continues on next page)

(continued from previous page)

```
    redis_url: redis://{{ hostvars[groups['redis-master'][0]]['ansible_
↪default_ipv4']['address'] }}:{{ hostvars[groups['redis-master'][0]]['redis_
↪sentinel_port'] }}?sentinel=master01{% for host in groups['redis-slave'] %}&
↪sentinel_fallback={{ hostvars[host]['ansible_default_ipv4']['address'] }}:{
↪{ hostvars[host]['redis_sentinel_port'] }}{% endfor %}&db=0
```

You also have to install additional pip/distro packages to use the redis cluster:

```
gnocchi_distro_packages:
- apache2
- apache2-utils
- libapache2-mod-wsgi
- git
- build-essential
- python3-dev
- librados-dev
- libpq-dev
- python3-rados
# additional package for python redis client
- python3-redis
- libsystemd-dev
```

```
gnocchi_pip_packages:
- cryptography
- gnocchiclient
# this is what we want:
#  - "gnocchi[mysql,ceph,ceph_alternative_lib,redis,keystone]"
# but as there is no librados >=12.2 pip package we have to first install␣
↪ceph without alternative support
# after adding the ceph repo to gnocchi container, python-rados>=12.2.0 is␣
↪installed and linked automatically
# and gnocchi will automatically take up the features present in the used␣
↪rados lib.
- "gnocchi[mysql,ceph,redis,keystone]"
- keystonemiddleware
- python-memcached
- redis
- systemd-python
```

Note

A word of caution: the name of the Ceph alternative lib implementation (ceph_alternative_lib) varies between Gnocchi versions.

Zookeeper for coordination

When you deployed Gnocchi on multiple servers to distribute the work, add Zookeeper as coordination backend. To setup Zookeeper, you can use this ansible role.

Create containers for Zookeeper:

```
## conf.d
zookeeper_hosts:
{% for server in groups['control_nodes'] %}
{{ server }}:
  ip: {{ hostvars[server]['ansible_facts']['default_ipv4']['address'] }}
{% endfor%}
```

```
## env.d
component_skel:
  zookeeper_server:
    belongs_to:
      - zookeeper_all

container_skel:
  zookeeper_container:
    belongs_to:
      - infra_containers
      - shared-infra_containers
    contains:
      - zookeeper_server
    properties:
      service_name: zookeeper
```

Now you can set up Zookeeper as coordination backend for Gnocchi:

```
gnocchi_coordination_url: "zookeeper://{% for host in groups['zookeeper_all']
↪%}{{ hostvars[host]['management_address'] }}:2181{% if not loop.last %},{%␣
↪endif %}{% endfor %}"
```

You also have to install additional packages:

```
gnocchi_pip_packages:
- cryptography
- gnocchiclient
# this is what we want:
#  - "gnocchi[mysql,ceph,ceph_alternative_lib,redis,keystone]"
# but as there is no librados >=12.2 pip package we have to first install␣
↪ceph without alternative support
# after adding the ceph repo to gnocchi container, python-rados>=12.2.0 is␣
↪installed and linked automatically
# and gnocchi will automatically take up the features present in the used␣
↪rados lib.
- "gnocchi[mysql,ceph,redis,keystone]"
- keystonemiddleware
- python-memcached
```

(continues on next page)

(continued from previous page)

```
- redis
- systemd-python
# addiitional pip packages needed for zookeeper coordination backend
- tooz
- lz4
- kazoo
```

Performance Tests for Gnocchi

For more ideas how to tune your Gnocchi stack, take a look at these presentations:

- https://docs.openstack.org/performance-docs/latest/test_results/telemetry_gnocchi_with_ceph/index.html

HAProxy and Keepalived in LXC containers

There can be a usecase where you might want to run HAProxy and Keepalived inside LXC containers. For instance, running these services on bare metal assumes that a default route for hosts should be set towards a public network. This scenario might be un-preferable for some deployments, especially in cases where you do not have standalone Load-Balancing hosts, but theyre co-located with other infra services instead.

Inventory overrides

In order to tell dynamic_inventory to generate a set of containers for HAProxy, you need to create a file `/etc/openstack_deploy/env.d/haproxy.yml` with the following content:

```
---
# This file contains an example to show how to set
# the cinder-volume service to run in a container.
#
# Important note:
# In most cases you need to ensure that default route inside of the
# container doesn't go through eth0, which is part of lxcbr0 and
# SRC nat-ed. You need to pass "public" VIP interface inside of the
# container and ensure "default" route presence on it.

container_skel:
  haproxy_container:
    properties:
      is_metal: false
```

Defining host networking

In order to make a public network available, you need to ensure having a corresponsive bridge on your hosts to which HAProxy containers will be plugged in with one side of a veth pair. The bridge should also contain a VLAN interface providing public connectivity.

You can create a bridge manually or leverage our systemd_networkd role which is capable of configuring required networking on hosts.

For the example below, lets name our bridge `br-public-api` and public vlan with ID `40`. In your `user_variables.yml` define the following variables:

```
_systemd_networkd_generic_devices:
  - NetDev:
      Name: bond0
      Kind: bond
    Bond:
      Mode: 802.3ad
      TransmitHashPolicy: layer3+4
      LACPTransmitRate: fast
      MIIMonitorSec: 100
    filename: 05-generic-bond0

_systemd_networkd_public_api_devices:
  - NetDev:
      Name: vlan-public-api
      Kind: vlan
    VLAN:
      Id: 40
    filename: 10-openstack-vlan-public-api
  - NetDev:
      Name: br-public-api
      Kind: bridge
    Bridge:
      ForwardDelaySec: 0
      HelloTimeSec: 2
      MaxAgeSec: 12
      STP: off
    filename: 11-openstack-br-public-api

openstack_hosts_systemd_networkd_devices: |-
  {% set devices = [] %}
  {% if is_metal %}
  {%   set _ = devices.extend(_systemd_networkd_generic_devices) %}
  {%   if inventory_hostname in groups['haproxy_hosts'] %}
  {%     set _ = devices.extend(_systemd_networkd_public_api_devices) %}
  {%   endif %}
  {% endif %}
  {{ devices }}

_systemd_networkd_bonded_networks:
  - interface: ens3
    filename: 05-generic-ens3
    bond: bond0
    link_config_overrides:
      Match:
        MACAddress: df:25:83:e1:77:c8
  - interface: ens6
    filename: 05-generic-ens6
    bond: bond0
```

(continues on next page)

(continued from previous page)

```
    link_config_overrides:
      Match:
        MACAddress: df:25:83:e1:77:c9
  - interface: bond0
    filename: 05-general-bond0
    vlan:
      - vlan-public-api

_systemd_networkd_public_api_networks:
  - interface: "vlan-public-api"
    bridge: "br-public-api"
    filename: 10-openstack-vlan-public-api
  - interface: "br-public-api"
    filename: "11-openstack-br-public-api"

openstack_hosts_systemd_networkd_networks: |-
  {% set networks = [] %}
  {% if is_metal %}
  {%   set _ = networks.extend(_systemd_networkd_bonded_networks) %}
  {%   if inventory_hostname in groups['haproxy_hosts'] %}
  {%     set _ = networks.extend(_systemd_networkd_public_api_networks) %}
  {%   endif %}
  {% endif %}
  {{ networks }}
```

Defining container networking

In case of deploying HAProxy inside LXC you need to ensure connectivity with a public network and that `haproxy_bind_external_lb_vip_address` will be present inside the container as well as `external_lb_vip_address` is reachable.

For that we need to do the following series of changes in the `openstack_user_config.yml` file.

1. In `cidr_networks` add a network which should be used as public network for accessing APIs. For example we will be using *203.0.113.128/28*:

   ```
   cidr_networks:
     ...
     public_api: 203.0.113.128/28
   ```

2. In `used_ips` you need to reserve IP address for your gateway and `haproxy_keepalived_external_vip_cidr/external_lb_vip_address`

   ```
   used_ips:
     ...
     - "203.0.113.129"
     - "203.0.113.140-203.0.113.142"
   ```

3. In `provider_networks` you need to define a new container network and assign it to HAProxy group.

```
global_overrides:
  ...
  provider_networks:
    ...
    - network:
      group_binds:
        - haproxy
      type: "raw"
      container_bridge: "br-public-api"
      container_interface: "eth20"
      container_type: "veth"
      ip_from_q: public_api
      static_routes:
        - cidr: 0.0.0.0/0
          gateway: 203.0.113.129
```

While these are all changes, that need to be done in `openstack_user_config.yml`, there is one more override that needs to be applied.

As you might have spotted, we are defining a default route for the container through eth20. However, by default all containers have their default route through eth0, which is a local LXC bridge where address is recieved through DHCP. In order to avoid a conflict, you need to ensure that the default route will not be set for eth0 inside the container. For that, create a file */etc/openstack_deploy/group_vars/haproxy* with the following content:

```
---
lxc_container_networks:
  lxcbr0_address:
    bridge: "{{ lxc_net_bridge | default('lxcbr0') }}"
    bridge_type: "{{ lxc_net_bridge_type | default('linuxbridge') }}"
    interface: eth0
    type: veth
    dhcp_use_routes: False
```

Configuring HAProxy binding inside containers

As IP provisioning is quite random inside containers, it may not always be handy to bind HAProxy to a specific IP address. If thats the case, you can bind HAProxy to an interface instead, since we always know the interface names inside containers. With that Keepalived public/internal VIPs are supposed to be added in `used_ips`, so you still can define them freely.

Example bellow shows a possible content in `user_variables.yml`:

```
haproxy_bind_external_lb_vip_interface: eth20
haproxy_bind_internal_lb_vip_interface: eth1
haproxy_bind_external_lb_vip_address: "*"
haproxy_bind_internal_lb_vip_address: "*"
haproxy_keepalived_external_vip_cidr: 203.0.113.140/32
haproxy_keepalived_internal_vip_cidr: 172.29.236.9/32
haproxy_keepalived_external_interface: "{{ haproxy_bind_external_lb_vip_
↪interface }}"
```

(continues on next page)

(continued from previous page)

```
haproxy_keepalived_internal_interface: "{{ haproxy_bind_internal_lb_vip_
↪interface }}"
```

Alternatively, you can detect IPs used inside your containers to configure HAProxy binds. This can be done by reffering to `container_networks` mapping:

```
haproxy_bind_external_lb_vip_address: "{{ container_networks['public_api_
↪address']['address'] }}"
haproxy_bind_internal_lb_vip_address: "{{ container_networks['management_
↪address']['address'] }}"
```

Creating containers

Once all steps above are accomplished, its time to create our new HAProxy containers. For that run the following command:

```
# openstack-ansible playbooks/lxc-containers-create.yml --limit haproxy,lxc_
↪hosts
```

Using domain (or path) based endpoints instead of port-based

By default, OpenStack-Ansible uses port-based endpoints. This means, that each service will be served on its own unique port for both public and internal endpoints. For example, Keystone will be added as `https://domain.com:5000/v3`, Nova as `https://domain.com:8774/v2.1` and so on.

While this is the simplest approach, as it does not require any extra configuration and is easy to start with, it also has some disadvantages. For example, some clients or organizations might not be allowed to connect to custom ports which completely disables the ability to use them in such deployments.

In order to work around such limitations, starting from 2023.1 (Antelope) release, it is possible to have domain-based or path-based endpoints instead.

Warning

After switching to domain or path-based endpoints, all internal and public endpoints must be covered by valid TLS certificates. Otherwise communication between OpenStack components may break. If TLS is not used for internal traffic in your deployment, ensure that HAProxy is configured so that internal requests are not redirected to HTTPS. This must be adjusted manually by the operator.

Configuring domain-based endpoints (recommended)

Domain-based endpoints do separate direct requests to specific services based on FQDNs. Usually for this purpose subdomains are used. For example, Keystone endpoint may look like `https://identity.domain.com` while Nova endpoint can be like `https://compute.domain.com`.

As a prerequisite for this type of setup you need to ensure that corresponding *A* or *CNAME* records are present for your domain. Also, you need to ensure having a valid wildcard or SAN certificates for public/internal endpoints.

HAProxy configuration

In order for HAProxy to pass specific FQDN to its own backend we will leverage map files functionality.

We need to make adjustments to each HAProxy service definition to:

- Prevent creation of a front-end per service. As we are now expecting traffic to come only on default *80* and *443* ports there is no need to have a separate frontend per service. A HAProxy map file is attached to a base frontend which is deployed with the `haproxy_server` role and is independent of any service definitions. The map file can be used to direct incoming requests to specific backends by using rules defined in the map file to match against host request headers.

  Note

  In case of any changes to `haproxy_base_service_overrides` variable you need to re-run `openstack-ansible openstack.osa.haproxy --tags haproxy-service-config`.

```
haproxy_base_service_overrides:
  haproxy_maps:
    - 'use_backend %[req.hdr(host),map_dom(/etc/haproxy/base_domain.map)]'
  haproxy_map_entries:
    - name: base_domain
      entries:
        - "# Domain map file - this comment is defined in the base␣
↪frontend config"
```

- Populate a base map file with search patterns per service backend. As each service is going to use its own FQDN we need to inform HAProxy which backend should be used when request is coming to the FQDN.

  Sample configuration for Keystone and Nova will look like this:

  Note

  With changes made to `haproxy_<service>_service_overrides` variable you need to re-run a service-specific playbook with *haproxy-service-config* tag, for example `openstack-ansible openstack.osa.keystone --tags haproxy-service-config`.

```
haproxy_keystone_service_overrides:
  haproxy_backend_only: true
  haproxy_map_entries:
    - name: base_domain
      entries:
        - "identity.{{ external_lb_vip_address }} keystone_service-back"
        - "identity.{{ internal_lb_vip_address }} keystone_service-back"

haproxy_nova_api_compute_service_overrides:
  haproxy_backend_only: true
  haproxy_map_entries:
    - name: base_domain
```

(continues on next page)

(continued from previous page)

```
      entries:
        - "compute.{{ external_lb_vip_address }} nova_api_os_compute-back"
        - "compute.{{ internal_lb_vip_address }} nova_api_os_compute-back"

haproxy_nova_novnc_console_service_overrides:
  haproxy_backend_only: true
  haproxy_map_entries:
    - name: base_domain
      entries:
        - "novnc.{{ external_lb_vip_address }} nova_novnc_console-back"
```

Because the base frontend will use the domain map, ensure incoming requests to the external VIP are also routed properly to Horizon backend.

Add the following overrides:

```
haproxy_horizon_service_overrides:
  haproxy_backend_only: true
  haproxy_map_entries:
    - name: base_domain
      entries:
        - "{{ external_lb_vip_address }} horizon-back"
```

Then reconfigure the HAProxy mappings:

```
# openstack-ansible openstack.osa.horizon --tags haproxy-service-config
```

When configuring domain-based endpoints, it is important to ensure that domain map entries are evaluated in the correct order. HAProxy processes map entries sequentially, and the first matching entry wins. Because of this, a generic domain such as `{{ external_lb_vip_address }}` can override more specific subdomains like `identity.<domain>` or `compute.<domain>` if it appears earlier in the map file.

To guarantee correct routing, assign an `order` value to each map entry block so that more specific service domains are processed before generic ones. Lower values take precedence (Horizons default order is 98).

Example:

```
haproxy_keystone_service_overrides:
  haproxy_backend_only: true
  haproxy_map_entries:
    - name: base_domain
      order: 10
      entries:
        - "identity.{{ external_lb_vip_address }} keystone_service-back"
        - "identity.{{ internal_lb_vip_address }} keystone_service-back"

haproxy_horizon_service_overrides:
  haproxy_backend_only: true
  haproxy_map_entries:
    - name: base_domain
      order: 20
```

(continues on next page)

(continued from previous page)

```
    entries:
      - "{{ external_lb_vip_address }} horizon-back"
```

In this configuration, Keystone map entries (order 10) are evaluated before the Horizon domain (order 20), ensuring that subdomains route to the correct backend.

Service configuration

Along with HAProxy configuration we also need to ensure that the endpoint catalog will be populated with correct URIs. Each service has a set of variables that needs to be overridden. Usually such variables have the following format:

- *<service>_service_publicuri*
- *<service>_service_internaluri*
- *<service>_service_adminuri*

Below you can find an example for defining endpoints for Keystone and Nova:

```
keystone_service_publicuri: "{{ openstack_service_publicuri_proto }}://
↪identity.{{ external_lb_vip_address }}"
keystone_service_internaluri: "{{ openstack_service_internaluri_proto }}://
↪identity.{{ internal_lb_vip_address }}"
keystone_service_adminuri: "{{ openstack_service_adminuri_proto }}://identity.
↪{{ internal_lb_vip_address }}"

nova_service_publicuri: "{{ openstack_service_publicuri_proto }}://compute.{{␣
↪external_lb_vip_address }}"
nova_service_internaluri: "{{ openstack_service_internaluri_proto }}://
↪compute.{{ internal_lb_vip_address }}"
nova_service_adminuri: "{{ openstack_service_adminuri_proto }}://compute.{{␣
↪internal_lb_vip_address }}"
nova_novncproxy_base_uri: "{{ nova_novncproxy_proto }}://novnc.{{ external_lb_
↪vip_address }}"
```

After changing endpoint definitions for new domains, refresh the endpoint data across components starting with the utility host playbook:

```
# openstack-ansible openstack.osa.utility_host
# openstack-ansible openstack.osa.setup_openstack --tags <service>-config
```

Under `<service>` include all deployed services such as neutron, cinder, glance, nova, horizon, etc.

Using Lets Encrypt

While you can consider having a wildcard or SAN TLS certificate for the domain to cover all service endpoints in this setup, it is still possible to use Lets Encrypt certificates with dns-01 authentication or by supplying a list of subdomains which issued certificate will cover.

So your Lets Encrypt configuration may look like this:

```
haproxy_ssl_letsencrypt_enable: true
haproxy_ssl_letsencrypt_email: "root@{{ external_lb_vip_address }}"
haproxy_ssl_letsencrypt_domains:
  - "{{ external_lb_vip_address }}"
  - "identity.{{ external_lb_vip_address }}"
  - "compute.{{ external_lb_vip_address }}"
```

Note

Please mention, that Internal FQDNs are still going to be covered with self-signed certificates as in most use-cases Lets Encrypt should not be able to verify domain ownership for internal VIPs, unless dns-01 auth is used.

You also might need to take care of expanding CN names for issued SAN certificate by the PKI role. For that you will have to override `haproxy_vip_binds` variable like in example below:

```
haproxy_vip_binds:
  - address: "{{ haproxy_bind_external_lb_vip_address }}"
    interface: "{{ haproxy_bind_external_lb_vip_interface }}"
    type: external
  - address: "{{ haproxy_bind_internal_lb_vip_address }}"
    interface: "{{ haproxy_bind_internal_lb_vip_interface }}"
    type: internal
    pki_san_records:
      - "{{ internal_lb_vip_address }}"
      - "identity.{{ internal_lb_vip_address }}"
      - "compute.{{ internal_lb_vip_address }}"
```

You also might want to adjust HSTS headers defined by `haproxy_security_headers_csp` variable. While default rules do allow subdomains out of the box, you might want to restrict records a bit more to disallow access on arbitrary ports.

Note

Variables `haproxy_security_child_src_records` and `haproxy_security_connect_src_records` are only available staring with 2024.2 (Dalmatian) version. You need to override `haproxy_security_headers_csp` as a whole for earlier releases

```
haproxy_security_child_src_records:
  - "novnc.{{ external_lb_vip_address }}"
haproxy_security_connect_src_records:
  - "{{ external_lb_vip_address }}
haproxy_security_frame_ancestors_records:
  - "{{ external_lb_vip_address }}
```

Configuring path-based endpoints

Path-based endpoints imply serving services on the same FQDN but differentiating them based on URI.

For example, Keystone can be configured as `https://domain.com/identity/v3` while Nova as `https://domain.com/compute/v2.1`

Warning

Please note, that Horizon does utilize */identity* for its Keystone panel, so if youre serving Horizon on */* (default) and using */identity* to forward traffic to Keystone backend, management of users, roles, projects inside the Horizon will be broken due to a conflict.

While path-based endpoints might look tempting due to using FQDN and thus not having the need for wildcard TLS, they are harder to maintain and more complex to set up. Also worth mentioning, that not all services are ready to support path-based endpoints, despite this approach being used in devstack.

Good example of exceptions which do not support path-based endpoints at the moment are VNC consoles for VMs (to be implemented with blueprint), Magnum (*bug report <https://launchpad.net/bugs/2083168>*) and Ceph Rados Gateway.

HAProxy configuration

Similar to domain-based endpoints we rely on HAProxy maps functionality. But instead of `map_dom` we will be using `map_reg`.

So we need to define a map file to be used and a way to parse it. For that we need to apply an override for the *base* service.

```
haproxy_base_service_overrides:
  haproxy_maps:
    - 'use_backend %[path,map_reg(/etc/haproxy/base_regex.map)]'
```

In case you do need to have a Ceph RGW or want to combine domain-based with path-based approach - you can do that by defining two map files:

Note

In case of any changes to `haproxy_base_service_overrides` variable you need to re-run `openstack-ansible openstack.osa.haproxy --tags haproxy-service-config`.

```
haproxy_base_service_overrides:
  haproxy_maps:
    - 'use_backend %[req.hdr(host),map_dom(/etc/haproxy/base_domain.map)] if
↪{ req.hdr(host),map_dom(/etc/haproxy/base_domain.map) -m found }'
    - 'use_backend %[path,map_reg(/etc/haproxy/base_regex.map)]'
```

If no domain will be matched HAProxy will proceed with path-based endpoints.

Next, we need to ensure a HAProxy configuration for each service does contain HAProxy map population with a respective condition, for example:

Note

With changes made to `haproxy_<service>_service_overrides` variable you need to re-run a service-specific playbook with *haproxy-service-config* tag, for example `openstack-ansible openstack.osa.keystone --tags haproxy-service-config`.

```
haproxy_keystone_service_overrides:
  haproxy_backend_only: true
  haproxy_map_entries:
    - name: base_regex
      entries:
        - "^/identity keystone_service-back"

haproxy_nova_api_compute_service_overrides:
  haproxy_backend_only: true
  haproxy_map_entries:
    - name: base_regex
      entries:
        - "^/compute nova_api_os_compute-back"
```

Service configuration

Similar to the domain-based endpoints we need to override endpoints definition for each service. Endpoints are usually defined with following variables:

- *<service>_service_publicuri*
- *<service>_service_internaluri*
- *<service>_service_adminuri*

Below you can find an example for defining endpoints for Keystone and Nova:

```
keystone_service_publicuri: "{{ openstack_service_publicuri_proto }}://{{␣
↪external_lb_vip_address }}/identity"
keystone_service_internaluri: "{{ openstack_service_internaluri_proto }}://{{␣
↪internal_lb_vip_address }}/identity"
keystone_service_adminuri: "{{ openstack_service_adminuri_proto }}://{{␣
↪internal_lb_vip_address }}/identity"

nova_service_publicuri: "{{ openstack_service_publicuri_proto }}://{{␣
↪external_lb_vip_address }}/compute"
nova_service_internaluri: "{{ openstack_service_internaluri_proto }}://{{␣
↪internal_lb_vip_address }}/compute"
nova_service_adminuri: "{{ openstack_service_adminuri_proto }}://{{ internal_
↪lb_vip_address }}/compute"
```

However, there is another important part of the configuration required per service which is not a case for domain-based setup. All services assume that theyve been served on root path (i.e. `/`) while in path-based approach we use a unique path for each service.

So we now need to make service respect the path and respond correctly on it. One way of doing that could be using rewrite mechanism in uWSGI, for example:

Warning

Example below does not represent a correct approach on how to configure path-based endpoint for most services

```
keystone_uwsgi_ini_overrides:
  uwsgi:
    route: '^/identity(.*)$ rewrite:$1'
```

But this approach is not correct and will result in issues in some clients or use cases, despite the service appearing completely functional. The problem with the approach above is related to how services return the *self* URL when its asked for. Most services will reply with their current micro-version and URI to this micro-version in reply.

If you are to use uWSGI rewrites like shown above, you will result in response like that:

```
curl https://cloud.com/identity/ | jq
{
"versions": {
    "values": [
    {
        "id": "v3.14",
        "status": "stable",
        "updated": "2020-04-07T00:00:00Z",
        "links": [
        {
            "rel": "self",
            "href": "https://cloud.com/v3/"
        }
        ],
        "media-types": [
        {
            "base": "application/json",
            "type": "application/vnd.openstack.identity-v3+json"
        }
        ]
    }
    ]
}
}
```

As you might see, *href* is pointing not to the expected location. While some clients may not refer to href link provided by service, others might use it as source of truth and which will result in failures.

Some services, like keystone, have a configuration options which may control how *href* is being defined. For instance, keystone does have *[DEFAULT]/public_endpoint* option, but this approach is not consistent across services. Moreover, keystone will return provided *public_endpoint* for all endpoints, including admin and internal.

With that, the only correct approach here would be to adjust `api-paste.ini` for each respective service. But, Keystone specifically, does not support api-paste.ini files. So the only way around it is actually a uWSGI rewrite and to define a *public_endpoint* in *keystone.conf*:

```
keystone_keystone_conf_overrides:
  DEFAULT:
    public_endpoint: "{{ keystone_service_publicuri }}"
```

For other services applying `api-paste.ini` can be done with variables, but each service have quite a unique content there, so approach cant be easily generalized. Below you can find overrides made for some services as an example:

```
_glance_api_paste_struct:
    /: {}
    /healthcheck: {}
    /image: api
    /image/healthcheck: healthcheck
glance_glance_api_paste_ini_overrides:
  composite:glance-api: "{{ _glance_api_paste_struct }}"
  composite:glance-api-caching: "{{ _glance_api_paste_struct }}"
  composite:glance-api-cachemanagement: "{{ _glance_api_paste_struct }}"
  composite:glance-api-keystone: "{{ _glance_api_paste_struct }}"
  composite:glance-api-keystone+caching: "{{ _glance_api_paste_struct }}"
  composite:glance-api-keystone+cachemanagement: "{{ _glance_api_paste_struct␣
↪}}"

neutron_api_paste_ini_overrides:
  composite:neutron:
    /: {}
    /v2.0: {}
    /network/: neutronversions_composite
    /network/v2.0: neutronapi_v2_0

nova_api_paste_ini_overrides:
  composite:osapi_compute:
    /: {}
    /v2: {}
    /v2.1: {}
    /v2/+: {}
    /v2.1/+: {}
    /compute: oscomputeversions
    /compute/v2: oscomputeversion_legacy_v2
    /compute/v2.1: oscomputeversion_v2
    /compute/v2/+: openstack_compute_api_v21_legacy_v2_compatible
    /compute/v2.1/+: openstack_compute_api_v21
```

We suggest referring to each service api-paste.ini for more details on how to properly configure overrides.

Messaging configuration

This section provides an overview of hybrid messaging deployment concepts and describes the necessary steps for a working OpenStack-Ansible (OSA) deployment where RPC and Notify communications are separated and integrated with different messaging server backends.

oslo.messaging library

The oslo.messaging library is part of the OpenStack Oslo project that provides intra-service messaging capabilities. The library supports two communication patterns (RPC and Notify) and provides an abstraction that hides the details of the messaging bus operation from the OpenStack services.

Notifications

Notify communications are an asynchronous exchange from notifier to listener. The messages transferred typically correspond to information updates or event occurrences that are published by an OpenStack service. The listener need not be present when the notification is sent as notify communications are temporally decoupled. This decoupling between notifier and listener requires that the messaging backend deployed for notifications provide message persistence such as a broker queue or log store. It is noteworthy that the message transfer is unidirectional from notifier to listener and there is no message flow back to the notifier.

RPC

The RPC is intended as a synchronous exchange between a client and server that is temporally bracketed. The information transferred typically corresponds to a request-response pattern for service command invocation. If the server is not present at the time the command is invoked, the call should fail. The temporal coupling requires that the messaging backend deployed support the bi-directional transfer of the request from caller to server and the associated reply sent from the server back to the caller. This requirement can be satisfied by a broker queue or a direct messaging backend server.

Messaging transport

The oslo.messaging library supports a messaging transport plugin capability such that RPC and Notify communications can be separated and different messaging backend servers can be deployed.

The oslo.messaging drivers provide the transport integration for the selected protocol and backend server. The following table summarizes the supported oslo.messaging drivers and the communication services they support.

Oslo.Messaging Driver	Transport Protocol	Backend Server	RPC	Notify	Messaging Type
rabbit	AMQP V0.9	rabbitmq	yes	yes	queue
kafka (experimental)	kafka binary	kafka		yes	queue (stream)

Standard deployment of RabbitMQ server

A single RabbitMQ server backend (e.g. server or cluster) is the default deployment for OpenStack-Ansible (OSA). This broker messaging backend provides the queue services for both RPC and Notification communications through its integration with the oslo.messaging rabbit driver. The oslo-messaging.yml file provides the default configuration to associate the oslo.messaging RPC and Notify services to the RabbitMQ server backend.

```
# Quorum Queues
oslomsg_rabbit_quorum_queues: "{{ rabbitmq_queue_replication }}"

# NOTE(noonedeadpunk): Disabled due to missing oslo.concurrency lock_path␣
↪defenition
#                      for services
oslomsg_rabbit_queue_manager: False

# RPC
oslomsg_rpc_transport: 'rabbit'
oslomsg_rpc_port: "{{ rabbitmq_port }}"
oslomsg_rpc_servers: "{{ rabbitmq_servers }}"
oslomsg_rpc_use_ssl: "{{ rabbitmq_use_ssl }}"
oslomsg_rpc_host_group: "{{ rabbitmq_host_group }}"
oslomsg_rpc_policies: "{{ rabbitmq_policies }}"

# Notify
oslomsg_notify_transport: "{{ (groups[rabbitmq_host_group] | length > 0) |␣
↪ternary('rabbit', 'none') }}"
oslomsg_notify_port: "{{ rabbitmq_port }}"
oslomsg_notify_servers: "{{ rabbitmq_servers }}"
oslomsg_notify_use_ssl: "{{ rabbitmq_use_ssl }}"
oslomsg_notify_host_group: "{{ rabbitmq_host_group }}"
oslomsg_notify_policies: "{{ rabbitmq_policies }}"
```

Managing RabbitMQ stream policy

When deploying RabbitMQ with support for quorum and stream queues, the retention behaviour for messages changes. Stream queues maintain an append only log on disk of all messages received until a retention policy indicates they should be disposed of. By default, this policy is set with a per-stream *x-max-age* of 1800 seconds. However, as noted in the RabbitMQ docs, this only comes into effect ones a stream has accumulated enough messages to fill a segment, which has a default size of 500MB.

If you would like to reduce disk usage, an additional policy can be applied via OpenStack-Ansible as shown below:

```
rabbitmq_policies:
  - name: CQv2
    pattern: '.*'
    priority: 0
    tags:
      queue-version: 2
    state: >-
      {{
        ((oslomsg_rabbit_quorum_queues | default(True) or not rabbitmq_queue_
↪replication) and rabbitmq_install_method | default('') != 'distro'
          ) | ternary('present', 'absent')
      }}
# The following is an example of an additional policy which applies to fanout/
↪stream queues only
```

(continues on next page)

(continued from previous page)

```
# By default, each stream uses RabbitMQ's 500MB segment size, and no messages␣
↪will be discarded
# until that size is reached. This may result in undesirable disk usage.
# If using this policy, it must be applied BEFORE any stream queues are␣
↪created.
# See also https://bugs.launchpad.net/oslo.messaging/+bug/2089845 and https://
↪www.rabbitmq.com/docs/streams#retention
#  - name: CQv2F
#    pattern: '^.*_fanout'
#    priority: 1
#    tags:
#      queue-version: 2
#      stream-max-segment-size-bytes: 1000000
#    state: >-
#      {{
#        ((oslomsg_rabbit_quorum_queues | default(True) or not rabbitmq_queue_
↪replication) and rabbitmq_install_method | default('') != 'distro'
#          ) | ternary('present', 'absent')
#      }}
```

Note however, that this policy will only apply if it is in place before any stream queues are created. If these already exist, they will need to be manually deleted and re-created by the relevant OpenStack service.

This issue is being tracked in an oslo.messaging bug.

Multi-Architecture Deployments

OpenStack-Ansible supports deployments where either the control plane or compute nodes may comprise of several different CPU architectures.

Mixed CPU architectures for compute nodes

OpenStack-Ansible supports having compute nodes of multiple architectures deployed in the same environment.

Deployments consisting entirely of x86_64 or aarch64 nodes do not need any special consideration and will work according to the normal OpenStack-Ansible documentation.

A deployment with a mixture of architectures, or adding a new architecture to an existing single architecure deployment requires some additional steps to be taken by both the deployer and end users to ensure that the behaviour is as desired.

Example - adding `aarch64` nodes to an `x86_64` deployment

1) Install the operating system onto all the new compute nodes.
2) Add the new compute nodes to `openstack_user_config.yml`.
3) Ensure a host of each compute architecture is present in `repo-infra_hosts` in `openstack_user_config.yml`.

   This host will build python wheels for its own architecture which will speed up the deployment of many hosts. If you do not make a repository server for each architecture, ensure that measures

are taken not to overload the opendev.org git servers, such as using local mirrors of all OpenStack service repos.

4) Run the OpenStack-Ansible playbooks to deploy the required services.

5) Add HW_ARCH_XXXX Trait to Every Compute Host in OpenStack.

   Although most CPU hardware traits such as instruction set extensions are detected and handled automatically in OpenStack, CPU architecture is not. It is necessary to manually add an architecture trait to the resource provider corresponding to every compute host. The required traits are:

   HW_ARCH_X86_64 for x86_64 Intel and AMD CPUs

   HW_ARCH_AARCH64 for aarch64 architecture CPUs

   (see: https://docs.openstack.org/os-traits/latest/reference/traits.html)

```
openstack resource provider list
openstack resource provider trait list <uuid-of-compute-host>
openstack resource provider trait set --trait <existing-trait-1>␣
↪--trait <existing-trait-2> ... --trait HW_ARCH_xxxxx <uuid-of-
↪compute-host>
```

   Note

   The trait set command replaces all existing traits with the set provided, so you must specify all existing traits as well as the new trait.

6) Configure Nova Scheduler to Check Architecture.

   Two additional settings in /etc/nova/nova.conf in all Nova API instances:

```
[scheduler]
image_metadata_prefilter = true

[filter_scheduler]
image_properties_default_architecture = x86_64
```

   The `image_metadata_prefilter` setting forces the Nova scheduler to match the `hw_architecture` property on Glance images with the corresponding HW_ARCH_XXX trait on compute host resource providers. This ensures that images explicitly tagged with a target architecture get scheduled hosts with a matching architecture.

   The `image_properties_default_architecture` setting would apply in an existing `x86_64` architecture cloud where previously `hw_architecture` was not set on all Glance images. This avoids the need to retrospectively apply the property for all existing images which may be difficult as users may have their own tooling to create and upload images without applying the required property.

   Warning

   Undocumented Behaviour Alert!

   Note that the image metadata prefilter and ImagePropertiesFilter are different and unrelated

steps in the process Nova scheduler uses to determine candidate compute hosts. This section explains how to use them together.

The `image_metadata_prefilter` only looks at the HW_ARCH_XXX traits on compute hosts and finds hardware that matches the required architecture. This only happens when the `hw_architecture` property is present on an image, and only if the required traits are manually added to compute hosts.

The `image_properties_default_architecture` is used by the ImagePropertiesFilter which examines all the architectures supported by QEMU on each compute host; this includes software emulations of non-native architectures.

If the full QEMU suite is installed on a compute host, that host will offer to run all architectures supported by the available `qemu-system-*` binaries. In this situation images without the `hw_architecture` property could be scheduled to a non native architecture host and emulated.

7) Disable QEMU Emulation.

Note

This step applies particularly to existing `x86_64` environments when new `aarch64` compute nodes are added and it cannot be assumed that the `hw_architecure` property is applied to all Glance images as the operator may not be in control of all image uploads.

To avoid unwanted QEMU emulation of non native architectures it is necessary to ensure that only the native `qemu-system-*` binary is present on all compute nodes. The simplest way to do this for existing deployments is to use the system package manager to ensure that the unwanted binaries are removed.

OpenStack-Ansible releases including 2023.1 and later will only install the native architecture *qemu-system-*‘ binary so this step should not be required on newer releases.

8) Upload images to Glance.

- Ideally the `hw_architecture` property is set for all uploaded images. It is mandatory to set this property for all architectures that do not match `image_properties_default_architecture`
- It is recommended to set the property `hw_firmware_type='uefi'` for any images which require UEFI boot, even when this implicit with the `aarch64` architecture. This is to avoid issues with NVRAM files in libvirt when deleting an instance.

Architecture emulation by Nova

Nova has the capability to allow emulation of one CPU architecture on a host with a different native CPU architecure, see https://docs.openstack.org/nova/latest/admin/hw-emulation-architecture.html for more details.

This OpenStack-Ansible documentation currently assumes that a deployer wishes to run images on a compute host with a native CPU architecure, and does not give an example configuration involving emulation.

1.1.3 OpenStack-Ansible Reference

This chapter contains all the additional reference information needed to deploy, configure, or upgrade an OpenStack-Ansible cloud.

For information on how to deploy your OpenStack-Ansible cloud, refer to the Deployment Guide for step-by-step instructions on how to deploy the OpenStack packages and dependencies on your cloud using OpenStack-Ansible.

For user guides, see the *User Guide*.

For information on how to manage and operate OpenStack-Ansible, see the *Operations Guide*.

For information on how to contribute, extend or develop OpenStack-Ansible, see the Developer Documentation.

Releases

What is the OpenStack-Ansible release model?

OpenStack-Ansible uses the cycle-trailing release model as specified in the OpenStack release model reference.

How are release tags decided?

In order to ensure a common understanding of what release versions mean, we use Semantic Versioning 2.0.0 for versioning as a basis. The exception to the rule is for milestone releases during a development cycle, where releases are tagged <MAJOR>.0.0b<MILESTONE> where <MAJOR> is the next major release number, and <MILESTONE> is the milestone number.

The OpenStack series names are alphabetical, with each letter matched to a number (e.g., Austin = 1, Bexar = 2, Newton = 14, Pike = 16, etc.). OpenStack-Ansible adopted the same <MAJOR> release numbering as the Nova project to match the overall OpenStack series version numbering.

How frequently does OpenStack-Ansible release?

Major releases are done every six months according to the OpenStack release schedule. Each major release is consistent with an OpenStack series.

Minor/patch releases are requested for stable branches on the second and last Friday of every month. The releases are typically completed within a few days of the request.

What version of OpenStack is deployed by OpenStack-Ansible?

For each OpenStack-Ansible release, the OpenStack version that is deployed is set to a specific OpenStack git SHA-1 hash (SHA). These are updated after every OpenStack-Ansible release. The intent is to ensure that OpenStack-Ansible users are able to enjoy an updated OpenStack environment with smaller increments of change than the typical upstream service releases allow for as they are usually very infrequent.

This does mean that a stable OpenStack-Ansible deployment will include a version of a service (e.g.: nova-17.0.3dev4) which does not match a tag exactly as you may expect (e.g.: nova-17.0.3).

If you wish to change the SHA to a specific SHA/tag/branch, or wish to use your own fork of an OpenStack service, please see the section titled *Overriding other upstream projects source code* in the user guide.

When does a patch to an OpenStack-Ansible role get into a release?

For each OpenStack-Ansible release, the Ansible roles that form that release are set to a specific git SHA-1 hash (SHA). These are updated after every OpenStack-Ansible release.

OpenStack-Ansible frequently does proactive bugfix backports. In order to reduce the risk of these backports introducing any destabilization, OpenStack-Ansible implements a soak period for any patches implemented in the stable branches for roles, but also provides for circumventing this in exceptional circumstances.

A patch merged into a role is immediately tested by other role tests, ensuring that any major breaking change is caught. Once a minor/patch release is requested, the integrated build receives a SHA bump patch to update the integrated build to using the latest available roles including that new patch. This new set is available for testing to anyone wanting to use the head of the stable branch, and is tested in periodic tests until the next release. In total, that means that the cycle time for a patch from merge to release is anywhere from two weeks to one month.

If there is a requirement to rush a role patch into the next release, then anyone may propose a change to the `ansible-role-requirements.yml` file in the `openstack/openstack-ansible` repository with the appropriate justification.

We believe that this approach brings a balance of both reasonable stability, while still being able to do pro-active backports.

The only exception to this process is for the `master` branch, which intentionally consumes the `master` branch from all roles between releases so that any changes are immediately integration tested.

Conventions

To avoid extra configuration, a series of conventions are set into code.

Default folders locations

Ansible roles

The ansible roles are stored under `/etc/ansible/roles`.

OpenStack-Ansible directory checkout

The code is generally located into `/opt/openstack-ansible`.

OpenStack-Ansible wrapper

Our `openstack-ansible` cli is located in `/usr/local/bin/openstack-ansible`. It sources an environment variable file located in: `/usr/local/bin/openstack-ansible.rc`.

Userspace configurations

All the userspace configurations are expected to be in `/etc/openstack_deploy/`.

Ansible configuration

ansible.cfg

There is no `ansible.cfg` provided with OpenStack-Ansible. Environment variables are used to alter the default Ansible behavior if necessary.

Ansible roles fetching

Any roles defined in `openstack-ansible/ansible-role-requirements.yml` will be installed by the `openstack-ansible/scripts/bootstrap-ansible.sh` script, and fetched into the ansible roles folder.

Inventory conventions

Please refer to the *Inventory* section of this reference.

Inventory

OpenStack-Ansible uses an included script to generate the inventory of hosts and containers within the environment. This script is called by Ansible through its dynamic inventory functionality.

In this section, you will find documentation relevant to the inventory for OpenStack-Ansible.

Generating the Inventory

The script that creates the inventory is located at `inventory/dynamic_inventory.py` and installed into the ansible-runtime virtualenv as `openstack-ansible-inventory`.

This section explains how ansible runs the inventory, and how you can run it manually to see its behavior.

Executing the dynamic_inventory.py script manually

When running an Ansible command (such as `ansible`, `ansible-playbook` or `openstack-ansible`) Ansible automatically executes the `dynamic_inventory.py` script and use its output as inventory.

Run the following command:

```
# from the root folder of cloned OpenStack-Ansible repository
inventory/dynamic_inventory.py --config /etc/openstack_deploy/
```

Dynamic inventory script is also installed inside virtualenv as a script. So alternatively you can run following:

```
source /opt/ansible-runtime/bin/activate
openstack-ansible-inventory --config /etc/openstack_deploy/
```

This invocation is useful when testing changes to the dynamic inventory script.

Inputs

The `openstack-ansible-inventory` takes the `--config` argument for the directory holding configuration from which to create the inventory. If not specified, the default is `/etc/openstack_deploy/`.

In addition to this argument, the base environment skeleton is provided in the `inventory/env.d` directory of the OpenStack-Ansible codebase.

Should an `env.d` directory be found in the directory specified by `--config`, its contents will be added to the base environment, overriding any previous contents in the event of conflicts.

The following file must be present in the configuration directory:

- `openstack_user_config.yml`

Additionally, the configuration or environment could be spread between two additional sub-directories:

- `conf.d`
- `env.d` (for environment customization)

The dynamic inventory script does the following:

- Generates the names of each container that runs a service
- Creates container and IP address mappings
- Assigns containers to physical hosts

As an example, consider the following excerpt from `openstack_user_config.yml`:

```
identity_hosts:
  infra01:
    ip: 10.0.0.10
  infra02:
    ip: 10.0.0.11
  infra03:
    ip: 10.0.0.12
```

The `identity_hosts` dictionary defines an Ansible inventory group named `identity_hosts` containing the three infra hosts. The configuration file `inventory/env.d/keystone.yml` defines additional Ansible inventory groups for the containers that are deployed onto the three hosts named with the prefix *infra*.

Note that any services marked with `is_metal: true` will run on the allocated physical host and not in a container. For an example of `is_metal: true` being used refer to `inventory/env.d/cinder.yml` in the `container_skel` section.

For more details, see *Configuring the inventory*.

Outputs

Once executed, the script will output an `openstack_inventory.json` file into the directory specified with the `--config` argument. This is used as the source of truth for repeated runs.

Warning

The `openstack_inventory.json` file is the source of truth for the environment. Deleting this in a production environment means that the UUID portion of container names will be regenerated, which then results in new containers being created. Containers generated under the previous version will no longer be recognized by Ansible, even if reachable via SSH.

The same JSON structure is printed to stdout, which is consumed by Ansible as the inventory for the playbooks.

Checking inventory configuration for errors

Using the `--check` flag when running `openstack-ansible-inventory` will run the inventory build process and look for known errors, but not write any files to disk.

If any groups defined in the `openstack_user_config.yml` or `conf.d` files are not found in the environment, a warning will be raised.

This check does not do YAML syntax validation, though it will fail if there are unparseable errors.

Writing debug logs

The `--debug/-d` parameter allows writing of a detailed log file for debugging the inventory scripts behavior. The output is written to `inventory.log` in the current working directory.

The `inventory.log` file is appended to, not overwritten.

Like `--check`, this flag is not invoked when running from ansible.

Running with tox

In some cases you might want to generate inventory on operator local machines after altering openstack_user_config.yml or env.d/conf.d files. Given that you already have `openstack_deploy` directory on such machine, you can create tox.ini file in that directory with following content:

```
[tox]
envlist = generate_inventory

[testenv]
skip_install = true
usedevelop = true
allowlist_externals =
    bash

[testenv:generate_inventory]
basepython = python3
deps = -rhttps://opendev.org/openstack/openstack-ansible/raw/branch/master/
↪requirements.txt
install_command =
    pip install -c https://releases.openstack.org/constraints/upper/master
↪{packages} -e git+https://opendev.org/openstack/openstack-ansible@master\
↪#egg=openstack-ansible
commands =
    openstack-ansible-inventory --config {toxinidir}/openstack_deploy
```

Then you can run a command to generate inventory using tox:

```
tox -e generate_inventory
```

As a result you will get your openstack_user_config.json updated. You can use this method also to verify validity of the inventory.

Configuring the inventory

In this chapter, you can find the information on how to configure the OpenStack-Ansible dynamic inventory to your needs.

Introduction

Common OpenStack services and their configuration are defined by OpenStack-Ansible in the `/etc/openstack_deploy/openstack_user_config.yml` settings file.

Additional services should be defined with a YAML file in `/etc/openstack_deploy/conf.d`, in order to manage file size.

The `/etc/openstack_deploy/env.d` directory sources all YAML files into the deployed environment, allowing a deployer to define additional group mappings. This directory is used to extend the environment skeleton, or modify the defaults defined in the `inventory/env.d` directory.

To understand how the dynamic inventory works, see *Understanding the inventory*.

Warning

Never edit or delete the file `/etc/openstack_deploy/openstack_inventory.json`. This can lead to problems with the inventory: existng hosts and containers will be unmanaged and new ones will be generated instead, breaking your existing deployment.

Configuration constraints

Group memberships

When adding groups, keep the following in mind:

- A group can contain hosts
- A group can contain child groups

However, groups cannot contain child groups and hosts.

The lxc_hosts Group

When the dynamic inventory script creates a container name, the host on which the container resides is added to the `lxc_hosts` inventory group.

Using this name for a group in the configuration will result in a runtime error.

Deploying directly on hosts

To deploy a component directly on the host instead of within a container, set the `is_metal` property to `true` for the container group in the `container_skel` section in the appropriate file.

The use of `container_vars` and mapping from container groups to host groups is the same for a service deployed directly onto the host.

You can also use the `no_containers` option to specify a host that will have all services deployed on metal inside of it.

Note

The `cinder-volume` component is deployed directly on the host by default. See the `env.d/cinder.yml` file for this example.

Example: Running all controllers on metal

For example, if youd like to run all your controllers on metal, you would have the following inside your `openstack_user_config.yml`.

```
infra_hosts:
  infra1:
    ip: 172.39.123.11
    no_containers: true
  infra2:
    ip: 172.39.123.12
    no_containers: true
  infra3:
    ip: 172.39.123.13
    no_containers: true
```

Example: Running Galera on dedicated hosts

For example, to run Galera directly on dedicated hosts, you would perform the following steps:

1. Modify the `container_skel` section of the `env.d/galera.yml` file. For example:

```
container_skel:
  galera_container:
    belongs_to:
      - db_containers
    contains:
      - galera
    properties:
      is_metal: true
```

Note

To deploy within containers on these dedicated hosts, omit the `is_metal: true` property.

2. Assign the `db_containers` container group (from the preceding step) to a host group by providing a `physical_skel` section for the host group in a new or existing file, such as `env.d/galera.yml`. For example:

```
physical_skel:
  db_containers:
    belongs_to:
      - all_containers
  db_hosts:
```

(continues on next page)

(continued from previous page)

```
    belongs_to:
      - hosts
```

3. Define the host group (`db_hosts`) in a `conf.d/` file (such as `galera.yml`). For example:

```
db_hosts:
  db-host1:
    ip: 172.39.123.11
  db-host2:
    ip: 172.39.123.12
  db-host3:
    ip: 172.39.123.13
```

Note

Each of the custom group names in this example (`db_containers` and `db_hosts`) are arbitrary. Choose your own group names, but ensure the references are consistent among all relevant files.

Adding virtual nest groups

If you want to create a custom group for arbitrary grouping of hosts and containers within these hosts but skip the generation of any new containers, you should use `is_nest` property under container_skel and skip defining `belongs_to` structure. `is_nest` property will add host-containers as children to such a group.

Example: Defining Availability Zones

A good example of how `is_nest` property can be used is describing Availability Zones. As when operating multiple AZs its handy to define AZ-specific variables, like AZ name, for all hosts in this AZ. And leveraging `group_vars` is best way of ensuring that all hosts that belong to same AZ have same configuration applied.

Lets assume you have 3 controllers and each of them is placed in different Availability Zones. There is also a compute node in each Availability Zone. And we want each host or container that is placed physically in a specific AZ be part of its own group (ie `azN_all`)

In order to achieve that we need:

1. Define host groups in `conf.d` or `openstack_user_config.yml` to assign hosts accordingly to their Availability Zones:

```
az1-infra_hosts: &infra_az1
  az1-infra1:
    ip: 172.39.123.11

az2-infra_hosts: &infra_az2
  az2-infra2:
    ip: 172.39.123.12
```

(continues on next page)

(continued from previous page)

```
az3-infra_hosts: &infra_az3
  az3-infra3:
    ip: 172.39.123.13

shared-infra_hosts: &controllers
  <<: *infra_az1
  <<: *infra_az2
  <<: *infra_az3

az1-compute_hosts: &computes_az1
  az1-compute01:
    ip: 172.39.123.100

az2-compute_hosts: &computes_az2
  az2-compute01:
    ip: 172.39.123.150

az3-compute_hosts: &computes_az3
  az3-compute01:
    ip: 172.39.123.200

compute_hosts:
  <<: *computes_az1
  <<: *computes_az2
  <<: *computes_az3

az1_hosts:
  <<: *computes_az1
  <<: *infra_az1

az2_hosts:
  <<: *computes_az2
  <<: *infra_az2

az3_hosts:
  <<: *computes_az3
  <<: *infra_az3
```

2. Create `env.d/az.yml` file that will leverage `is_nest` property and allow all infra containers to be part of the AZ group as well

```
component_skel:
  az1_containers:
    belongs_to:
      - az1_all
  az1_hosts:
    belongs_to:
      - az1_all

  az2_containers:
```

(continues on next page)

(continued from previous page)

```
    belongs_to:
      - az2_all
  az2_hosts:
    belongs_to:
      - az2_all

  az3_containers:
    belongs_to:
      - az3_all
  az3_hosts:
    belongs_to:
      - az3_all

container_skel:
  az1_containers:
    properties:
      is_nest: true
  az2_containers:
    properties:
      is_nest: true
  az3_containers:
    properties:
      is_nest: true
```

3. Now you can leverage `group_vars` file to apply a variable to all containers and bare metal hosts in AZ. For example `/etc/openstack_deploy/group_vars/az1_all.yml`:

```
---
az_name: az1
cinder_storage_availability_zone: "{{ az_name }}"
```

Deploying with no component type per host (or more than one)

When OpenStack-Ansible generates its dynamic inventory, the affinity setting determines how many containers of a similar type are deployed on a single physical host.

Using `shared-infra_hosts` as an example, consider this `openstack_user_config.yml` configuration:

```
shared-infra_hosts:
  infra1:
    ip: 172.29.236.101
  infra2:
    ip: 172.29.236.102
  infra3:
    ip: 172.29.236.103
```

Three hosts are assigned to the `shared-infra_hosts` group, OpenStack-Ansible ensures that each host runs a single database container, a single Memcached container, and a single RabbitMQ container. Each host has an affinity of 1 by default, which means that each host runs one of each container type.

If you are deploying a stand-alone Object Storage (swift) environment, you can skip the deployment of RabbitMQ. If you use this configuration, your `openstack_user_config.yml` file would look as follows:

```
shared-infra_hosts:
  infra1:
    affinity:
      rabbit_mq_container: 0
    ip: 172.29.236.101
  infra2:
    affinity:
      rabbit_mq_container: 0
    ip: 172.29.236.102
  infra3:
    affinity:
      rabbit_mq_container: 0
    ip: 172.29.236.103
```

This configuration deploys a Memcached container and a database container on each host, but no RabbitMQ containers.

Omit a service or component from the deployment

To omit a component from a deployment, you can use one of several options:

- Remove the `physical_skel` link between the container group and the host group by deleting the related file located in the `env.d/` directory.
- Do not run the playbook that installs the component. Unless you specify the component to run directly on a host by using the `is_metal` property, a container is created for this component.
- Adjust the *Adding virtual nest groups* to 0 for the host group. Similar to the second option listed here, Unless you specify the component to run directly on a host by using the `is_metal` property, a container is created for this component.

Having SSH network different from OpenStack Management network

In some environments SSH network that is used to access nodes from deploy host and management network are different. In this case its important that services were listening on correct network while ensure that Ansible use SSH network for accessing managed hosts. In these cases you can define `management_ip` key while defining hosts in your `openstack_user_config.yml` file.

`management_ip` will be used as `management_address` for the node, while `ip` will be used as `ansible_host` for accessing node by SSH.

Example:

```
shared-infra_hosts:
  infra1:
    ip: 192.168.0.101
    management_ip: 172.29.236.101
```

Understanding the inventory

The default layout of containers and services in OpenStack-Ansible (OSA) is determined by the `/etc/openstack_deploy/openstack_user_config.yml` file and the contents of both the `/etc/openstack_deploy/conf.d/` and `/etc/openstack_deploy/env.d/` directories. You use these sources to define the *group* mappings that the playbooks use to target hosts and containers for roles used in the deploy.

- You define host groups, which gather the target hosts into *inventory groups*, through the `/etc/openstack_deploy/openstack_user_config.yml` file and the contents of the `/etc/openstack_deploy/conf.d/` directory.
- You define *container groups*, which can map from the service components to be deployed up to host groups, through files in the `/etc/openstack_deploy/env.d/` directory.

To customize the layout of the components for your deployment, modify the host groups and container groups appropriately before running the installation playbooks.

Understanding host groups (conf.d structure)

As part of the initial configuration, each target host appears either in the `/etc/openstack_deploy/openstack_user_config.yml` file or in files within the `/etc/openstack_deploy/conf.d/` directory. The format used for files in the `conf.d/` directory is identical to the syntax used in the `openstack_user_config.yml` file.

In these files, the target hosts are listed under one or more headings, such as `shared-infra_hosts` or `storage_hosts`, which serve as Ansible group mappings. These groups map to the physical hosts.

The `haproxy.yml.example` file in the `conf.d/` directory provides a simple example of defining a host group (`load_balancer_hosts`) with two hosts (`infra1` and `infra2`).

The `swift.yml.example` file provides a more complex example. Here, host variables for a target host are specified by using the `container_vars` key. OpenStack-Ansible applies all entries under this key as host-specific variables to any component containers on the specific host.

Note

To manage file size, we recommend that you define new inventory groups, particularly for new services, by using a new file in the `conf.d/` directory.

Understanding container groups (env.d structure)

Additional group mappings are located within files in the `/etc/openstack_deploy/env.d/` directory. These groups are treated as virtual mappings from the host groups (described above) onto the container groups, that define where each service deploys. By reviewing files within the `env.d/` directory, you can begin to see the nesting of groups represented in the default layout.

For example, the `shared-infra.yml` file defines a container group, `shared-infra_containers`, as a subset of the `all_containers` inventory group. The `shared- infra_containers` container group is mapped to the `shared-infra_hosts` host group. All of the service components in the `shared-infra_containers` container group are deployed to each target host in the `shared-infra_hosts host` group.

Within a `physical_skel` section, the OpenStack-Ansible dynamic inventory expects to find a pair of keys. The first key maps to items in the `container_skel` section, and the second key maps to the target

host groups (described above) that are responsible for hosting the service component.

To continue the example, the `memcache.yml` file defines the `memcache_container` container group. This group is a subset of the `shared-infra_containers` group, which is itself a subset of the `all_containers` inventory group.

Note

The `all_containers` group is automatically defined by OpenStack-Ansible. Any service component managed by OpenStack-Ansible maps to a subset of the `all_containers` inventory group, directly or indirectly through another intermediate container group.

The default layout does not rely exclusively on groups being subsets of other groups. The `memcache` component group is part of the `memcache_container` group, as well as the `memcache_all` group and also contains a `memcached` component group. If you review the `playbooks/memcached-install.yml` playbook, you see that the playbook applies to hosts in the `memcached` group. Other services might have more complex deployment needs. They define and consume inventory container groups differently. Mapping components to several groups in this way allows flexible targeting of roles and tasks.

openstack_user_config settings reference

The `openstack_user_config.yml.example` file is heavily commented with the details of how to do more advanced container networking configuration. The contents of the file are shown here for reference.

```
#
# Overview
# ========
#
# This file contains the configuration for OpenStack Ansible Deployment
# (OSA) core services. Optional service configuration resides in the
# conf.d directory.
#
# You can customize the options in this file and copy it to
# /etc/openstack_deploy/openstack_user_config.yml or create a new
# file containing only necessary options for your environment
# before deployment.
#
# OSA implements PyYAML to parse YAML files and therefore supports structure
# and formatting options that augment traditional YAML. For example, aliases
# or references. For more information on PyYAML, see the documentation at
#
# http://pyyaml.org/wiki/PyYAMLDocumentation
#
# Configuration reference
# =======================
#
# Level: cidr_networks (required)
# Contains an arbitrary list of networks for the deployment. For each network,
# the inventory generator uses the IP address range to create a pool of IP
# addresses for network interfaces inside containers. A deployment requires
# at least one network for management.
```

(continues on next page)

(continued from previous page)

```
#
#   Option: <value> (required, string)
#   Name of network and IP address range in CIDR notation. This IP address
#   range coincides with the IP address range of the bridge for this network
#   on the target host.
#
# Example:
#
# Define networks for a typical deployment.
#
#   - Management network on 172.29.236.0/22. Control plane for infrastructure
#     services, OpenStack APIs, and horizon.
#   - Tunnel network on 172.29.240.0/22. Data plane for project (tenant) VXLAN
#     networks.
#   - Storage network on 172.29.244.0/22. Data plane for storage services such
#     as cinder and swift.
#
# cidr_networks:
#   management: 172.29.236.0/22
#   tunnel: 172.29.240.0/22
#   storage: 172.29.244.0/22
#
# Example:
#
# Define additional service network on 172.29.248.0/22 for deployment in a
# Rackspace data center.
#
#   snet: 172.29.248.0/22
#
# --------
#
# Level: used_ips (optional)
# For each network in the 'cidr_networks' level, specify a list of IP addresses
# or a range of IP addresses that the inventory generator should exclude from
# the pools of IP addresses for network interfaces inside containers. To use a
# range, specify the lower and upper IP addresses (inclusive) with a comma
# separator.
#
# Example:
#
# The management network includes a router (gateway) on 172.29.236.1 and
# DNS servers on 172.29.236.11-12. The deployment includes seven target
# servers on 172.29.236.101-103, 172.29.236.111, 172.29.236.121, and
# 172.29.236.131. However, the inventory generator automatically excludes
# these IP addresses. The deployment host itself isn't automatically
# excluded. Network policy at this particular example organization
# also reserves 231-254 in the last octet at the high end of the range for
# network device management.
#
```

(continues on next page)

(continued from previous page)

```
# used_ips:
#   - 172.29.236.1
#   - "172.29.236.100,172.29.236.200"
#   - "172.29.240.100,172.29.240.200"
#   - "172.29.244.100,172.29.244.200"
#
# --------
#
# Level: global_overrides (required)
# Contains global options that require customization for a deployment. For
# example, load balancer virtual IP addresses (VIP). This level also provides
# a mechanism to override other options defined in the playbook structure.
#
#   Option: internal_lb_vip_address (required, string)
#   Load balancer VIP (or valid FQDN) for the following items:
#
#   - Local package repository
#   - Galera SQL database cluster
#   - Administrative and internal API endpoints for all OpenStack services
#   - Glance registry
#   - Nova compute source of images
#   - Cinder source of images
#   - Instance metadata
#
#   Option: external_lb_vip_address (required, string)
#   Load balancer VIP (or valid FQDN) for the following items:
#
#   - Public API endpoints for all OpenStack services
#   - Horizon
#
#   Option: management_bridge (required, string)
#   Name of management network bridge on target hosts. Typically 'br-mgmt'.
#
#   Level: provider_networks (required)
#   List of container and bare metal networks on target hosts.
#
#     Level: network (required)
#     Defines a container or bare metal network. Create a level for each
#     network.
#
#       Option: type (required, string)
#       Type of network. Networks other than those for neutron such as
#       management and storage typically use 'raw'. Neutron networks use
#       'flat', 'vlan', or 'vxlan'. Coincides with ML2 plug-in configuration
#       options.
#
#       Option: container_bridge (required, string)
#       Name of unique bridge on target hosts to use for this network. Typical
#       values include 'br-mgmt', 'br-storage', 'br-vlan', 'br-vxlan', etc.
```

(continues on next page)

(continued from previous page)

```
#
#       Option: container_bridge_type (optional, string)
#       Type of container_bridge on target hosts. This option should only set
#       to "openvswitch" when the container_bridge is set up with openvswitch.
#       The default value is undefined, which means bridge type is linux␣
↪bridge.
#
#       Option: container_interface (required, string)
#       Name of unique interface in containers to use for this network.
#       Typical values include 'eth1', 'eth2', etc. This option is OPTIONAL
#       for Neutron provider network definitions when Neutron agents are
#       deployed on bare metal (default), but REQUIRED when agents are
#       deployed in containers and for all other non-Neutron use-cases.
#       NOTE: Container interface is different from host interfaces.
#
#       Option: container_type (required, string)
#       Name of mechanism that connects interfaces in containers to the bridge
#       on target hosts for this network. Typically 'veth'.
#
#       Option: host_bind_override (optional, string)
#       Name of the physical network interface on the same L2 network being
#       used with the br-vlan device. This host_bind_override should only
#       be set for the ' container_bridge: "br-vlan" '.
#       This interface is optional but highly recommended for vlan based
#       OpenStack networking.
#       If no additional network interface is available, a deployer can create
#       a veth pair, and plug it into the br-vlan bridge to provide
#       this interface. An example could be found in the aio_interfaces.cfg
#       file.
#
#       Option: container_mtu (optional, string)
#       Sets the MTU within LXC for a given network type.
#
#       Option: ip_from_q (optional, string)
#       Name of network in 'cidr_networks' level to use for IP address pool.␣
↪Only
#       valid for 'raw' and 'vxlan' types.
#
#       Option: address_prefix (option, string)
#       Override for the prefix of the key added to each host that contains IP
#       address information for this network. By default, this will be the␣
↪name
#       given in 'ip_from_q' with a fallback of the name of the interface given␣
↪in
#       'container_interface'.
#       (e.g., 'ip_from_q'_address and 'container_interface'_address)
#
#       Option: is_management_address (required, boolean)
#       If true, the load balancer uses this IP address to access services
```

(continues on next page)

(continued from previous page)

```
#       in the container. Only valid for networks with 'ip_from_q' option.
#
#       Option: group_binds (required, string)
#       List of one or more Ansible groups that contain this
#       network. For more information, see the env.d YAML files.
#
#       Option: reference_group (optional, string)
#       An Ansible group that a host must be a member of, in addition to any␣
↪of the
#       groups within 'group_binds', for this network to apply.
#
#       Option: net_name (optional, string)
#       Name of network for 'flat' or 'vlan' types. Only valid for these
#       types. Coincides with ML2 plug-in configuration options.
#
#       Option: range (optional, string)
#       For 'vxlan' type neutron networks, range of VXLAN network identifiers
#       (VNI). For 'vlan' type neutron networks, range of VLAN tags. Supports
#       more than one range of VLANs on a particular network. Coincides with
#       ML2 plug-in configuration options.
#
#       Option: static_routes (optional, list)
#       List of additional routes to give to the container interface.
#       Each item is composed of cidr and gateway. The items will be
#       translated into the container network interfaces configuration
#       as a `post-up ip route add <cidr> via <gateway> || true`.
#
#       Option: gateway (optional, string)
#       String containing the IP of the default gateway used by the
#       container. Generally not needed: the containers will have
#       their default gateway set with dnsmasq, pointing to the host
#       which does natting for container connectivity.
#
#   Example:
#
#   Define a typical network architecture:
#
#   - Network of type 'raw' that uses the 'br-mgmt' bridge and 'management'
#     IP address pool. Maps to the 'eth1' device in containers. Applies to all
#     containers and bare metal hosts. Both the load balancer and Ansible
#     use this network to access containers and services.
#   - Network of type 'raw' that uses the 'br-storage' bridge and 'storage'
#     IP address pool. Maps to the 'eth2' device in containers. Applies to
#     nova compute and all storage service containers. Optionally applies to
#     to the swift proxy service.
#   - Network of type 'vxlan' that contains neutron VXLAN tenant networks
#     1 to 1000 and uses 'br-vxlan' bridge on target hosts. Applies to all
#     neutron agents on bare metal hosts.
#   - Network of type 'vlan' that contains neutron VLAN networks 101 to 200
```

(continues on next page)

(continued from previous page)

```
#     and 301 to 400 and uses the 'br-vlan' bridge on target hosts. Applies
#     to all neutron agent containers and neutron agents on bare metal hosts.
#   - Network of type 'flat' that contains one neutron flat network and uses
#     the 'br-vlan' bridge on target hosts. Maps to the 'eth12' device in
#     containers. Applies to all neutron agent containers and neutron agents
#     on bare metal hosts.
#
#   Note: A pair of 'vlan' and 'flat' networks can use the same bridge because
#   one only handles tagged frames and the other only handles untagged frames
#   (the native VLAN in some parlance). However, additional 'vlan' or 'flat'
#   networks require additional bridges.
#
#   provider_networks:
#     - network:
#         group_binds:
#           - all_containers
#           - hosts
#         type: "raw"
#         container_bridge: "br-mgmt"
#         container_interface: "eth1"
#         container_type: "veth"
#         ip_from_q: "management"
#         is_management_address: true
#     - network:
#         group_binds:
#           - glance_api
#           - cinder_api
#           - cinder_volume
#           - nova_compute
#           # Uncomment the next line if using swift with a storage network.
#           # - swift_proxy
#         type: "raw"
#         container_bridge: "br-storage"
#         container_type: "veth"
#         container_interface: "eth2"
#         container_mtu: "9000"
#         ip_from_q: "storage"
#     - network:
#         group_binds:
#           - neutron_openvswitch_agent
#         container_bridge: "br-vxlan"
#         container_type: "veth"
#         container_interface: "eth10"
#         container_mtu: "9000"
#         ip_from_q: "tunnel"
#         type: "vxlan"
#         range: "1:1000"
#         net_name: "vxlan"
#     - network:
```

(continues on next page)

(continued from previous page)

```
#         group_binds:
#           - neutron_openvswitch_agent
#         container_bridge: "br-vlan"
#         container_type: "veth"
#         container_interface: "eth11"
#         type: "vlan"
#         range: "101:200,301:400"
#         net_name: "physnet1"
#     - network:
#         group_binds:
#           - neutron_openvswitch_agent
#         container_bridge: "br-vlan"
#         container_type: "veth"
#         container_interface: "eth12"
#         host_bind_override: "eth12"
#         type: "flat"
#         net_name: "physnet2"
#
# --------
#
# Level: shared-infra_hosts (required)
# List of target hosts on which to deploy shared infrastructure services
# including the Galera SQL database cluster, RabbitMQ, and Memcached. Recommend
# three minimum target hosts for these services.
#
#   Level: <value> (required, string)
#   Hostname of a target host.
#
#     Option: ip (required, string)
#     IP address of this target host, typically the IP address assigned to
#     the management bridge.
#
# Example:
#
# Define three shared infrastructure hosts:
#
# shared-infra_hosts:
#   infra1:
#     ip: 172.29.236.101
#   infra2:
#     ip: 172.29.236.102
#   infra3:
#     ip: 172.29.236.103
#
# --------
#
# Level: repo-infra_hosts (required)
# List of target hosts on which to deploy the package repository. Recommend
```

(continues on next page)

(continued from previous page)

```
# minimum three target hosts for this service. Typically contains the same
# target hosts as the 'shared-infra_hosts' level.
#
#   Level: <value> (required, string)
#   Hostname of a target host.
#
#     Option: ip (required, string)
#     IP address of this target host, typically the IP address assigned to
#     the management bridge.
#
# Example:
#
# Define three package repository hosts:
#
# repo-infra_hosts:
#   infra1:
#     ip: 172.29.236.101
#   infra2:
#     ip: 172.29.236.102
#   infra3:
#     ip: 172.29.236.103
#
# --------
#
# Level: coordination_hosts (optional)
# Some services, like cinder , octavia, gnocchi or designate require
# coordination service to be present for some use-cases. As such service
# Zookeeper is being used.
#
# Example:
#
# Define three hosts for zookeeper cluster:
#
# coordination_hosts:
#   infra1:
#     ip: 172.29.236.101
#   infra2:
#     ip: 172.29.236.102
#   infra3:
#     ip: 172.29.236.103
#
# --------
#
# Level: os-infra_hosts (required)
# List of target hosts on which to deploy the glance API, nova API, heat API,
# and horizon. Recommend three minimum target hosts for these services.
# Typically contains the same target hosts as 'shared-infra_hosts' level.
#
#   Level: <value> (required, string)
```

(continues on next page)

(continued from previous page)

```
#   Hostname of a target host.
#
#     Option: ip (required, string)
#     IP address of this target host, typically the IP address assigned to
#     the management bridge.
#
# Example:
#
# Define three OpenStack infrastructure hosts:
#
# os-infra_hosts:
#   infra1:
#     ip: 172.29.236.101
#   infra2:
#     ip: 172.29.236.102
#   infra3:
#     ip: 172.29.236.103
#
# --------
#
# Level: identity_hosts (required)
# List of target hosts on which to deploy the keystone service. Recommend
# three minimum target hosts for this service. Typically contains the same
# target hosts as the 'shared-infra_hosts' level.
#
#   Level: <value> (required, string)
#   Hostname of a target host.
#
#     Option: ip (required, string)
#     IP address of this target host, typically the IP address assigned to
#     the management bridge.
#
# Example:
#
# Define three OpenStack identity hosts:
#
# identity_hosts:
#   infra1:
#     ip: 172.29.236.101
#   infra2:
#     ip: 172.29.236.102
#   infra3:
#     ip: 172.29.236.103
#
# --------
#
# Level: network_hosts (required)
# List of target hosts on which to deploy neutron services. Recommend three
# minimum target hosts for this service. Typically contains the same target
```

(continues on next page)

(continued from previous page)

```
# hosts as the 'shared-infra_hosts' level.
#
#   Level: <value> (required, string)
#   Hostname of a target host.
#
#     Option: ip (required, string)
#     IP address of this target host, typically the IP address assigned to
#     the management bridge.
#
# Example:
#
# Define three OpenStack network hosts:
#
# network_hosts:
#   infra1:
#     ip: 172.29.236.101
#   infra2:
#     ip: 172.29.236.102
#   infra3:
#     ip: 172.29.236.103
#
# If you want to deploy neutron agents to different set of hosts (net nodes)
# rather then neutron API, you should define 'network-infra_hosts' and
# 'network-agent_hosts' instead.
#
# Example:
#
# network-infra_hosts:
#   infra1:
#     ip: 172.29.236.101
#   infra2:
#     ip: 172.29.236.102
#   infra3:
#     ip: 172.29.236.103
#
# network-agent_hosts:
#   net1:
#     ip: 172.29.236.201
#   net2:
#     ip: 172.29.236.202
#   net3:
#     ip: 172.29.236.203
#
# --------
#
# Level: compute_hosts (optional)
# List of target hosts on which to deploy the nova compute service. Recommend
# one minimum target host for this service. Typically contains target hosts
# that do not reside in other levels.
```

(continues on next page)

(continued from previous page)

```
#
#   Level: <value> (required, string)
#   Hostname of a target host.
#
#     Option: ip (required, string)
#     IP address of this target host, typically the IP address assigned to
#     the management bridge.
#
# Example:
#
# Define an OpenStack compute host:
#
# compute_hosts:
#   compute1:
#     ip: 172.29.236.121
#
# --------
#
# Level: ironic-compute_hosts (optional)
# List of target hosts on which to deploy the nova compute service for Ironic.
# Recommend one minimum target host for this service. Typically contains␣
↪target
# hosts that do not reside in other levels.
#
#   Level: <value> (required, string)
#   Hostname of a target host.
#
#     Option: ip (required, string)
#     IP address of this target host, typically the IP address assigned to
#     the management bridge.
#
# Example:
#
# Define an OpenStack compute host:
#
# ironic-compute_hosts:
#   ironic-infra1:
#     ip: 172.29.236.121
#
# --------
#
# Level: storage-infra_hosts (required)
# List of target hosts on which to deploy the cinder API. Recommend three
# minimum target hosts for this service. Typically contains the same target
# hosts as the 'shared-infra_hosts' level.
#
#   Level: <value> (required, string)
#   Hostname of a target host.
#
```

(continues on next page)

(continued from previous page)

```
#     Option: ip (required, string)
#     IP address of this target host, typically the IP address assigned to
#     the management bridge.
#
# Example:
#
# Define three OpenStack storage infrastructure hosts:
#
# storage-infra_hosts:
#   infra1:
#     ip: 172.29.236.101
#   infra2:
#     ip: 172.29.236.102
#   infra3:
#     ip: 172.29.236.103
#
# --------
#
# Level: storage_hosts (required)
# List of target hosts on which to deploy the cinder volume service. Recommend
# one minimum target host for this service. Typically contains target hosts
# that do not reside in other levels.
#
#   Level: <value> (required, string)
#   Hostname of a target host.
#
#     Option: ip (required, string)
#     IP address of this target host, typically the IP address assigned to
#     the management bridge.
#
#     Level: container_vars (required)
#     Contains storage options for this target host.
#
#       Option: cinder_storage_availability_zone (optional, string)
#       Cinder availability zone.
#
#       Option: cinder_default_availability_zone (optional, string)
#       If the deployment contains more than one cinder availability zone,
#       specify a default availability zone.
#
#       Level: cinder_backends (required)
#       Contains cinder backends.
#
#         Option: limit_container_types (optional, string)
#         Container name string in which to apply these options. Typically
#         any container with 'cinder_volume' in the name.
#
#         Level: <value> (required, string)
#         Arbitrary name of the backend. Each backend contains one or more
```

(continues on next page)

(continued from previous page)

```
#          options for the particular backend driver. The template for the
#          cinder.conf file can generate configuration for any backend
#          providing that it includes the necessary driver options.
#
#            Option: volume_backend_name (required, string)
#            Name of backend, arbitrary.
#
#            The following options apply to the LVM backend driver:
#
#            Option: volume_driver (required, string)
#            Name of volume driver, typically
#            'cinder.volume.drivers.lvm.LVMVolumeDriver'.
#
#            Option: volume_group (required, string)
#            Name of LVM volume group, typically 'cinder-volumes'.
#
#            The following options apply to the NFS backend driver:
#
#            Option: volume_driver (required, string)
#            Name of volume driver,
#            'cinder.volume.drivers.nfs.NfsDriver'.
#            NB. When using NFS driver you may want to adjust your
#            env.d/cinder.yml file to run cinder-volumes in containers.
#
#            Option: nfs_shares_config (optional, string)
#            File containing list of NFS shares available to cinder, typically
#            '/etc/cinder/nfs_shares'.
#
#            Option: nfs_mount_point_base (optional, string)
#            Location in which to mount NFS shares, typically
#            '$state_path/mnt'.
#
#            Option: nfs_mount_options (optional, string)
#            Mount options used for the NFS mount points.
#
#            Option: shares (required)
#            List of shares to populate the 'nfs_shares_config' file. Each share
#            uses the following format:
#            - { ip: "{{ ip_nfs_server }}", share "/vol/cinder" }
#
#            The following options apply to the NetApp backend driver:
#
#            Option: volume_driver (required, string)
#            Name of volume driver,
#            'cinder.volume.drivers.netapp.common.NetAppDriver'.
#            NB. When using NetApp drivers you may want to adjust your
#            env.d/cinder.yml file to run cinder-volumes in containers.
#
#            Option: netapp_storage_family (required, string)
```

(continues on next page)

(continued from previous page)

```
#           Access method, typically 'ontap_7mode' or 'ontap_cluster'.
#
#           Option: netapp_storage_protocol (required, string)
#           Transport method, typically 'scsi' or 'nfs'. NFS transport also
#           requires the 'nfs_shares_config' option.
#
#           Option: nfs_shares_config (required, string)
#           For NFS transport, name of the file containing shares. Typically
#           '/etc/cinder/nfs_shares'.
#
#           Option: netapp_server_hostname (required, string)
#           NetApp server hostname.
#
#           Option: netapp_server_port (required, integer)
#           NetApp server port, typically 80 or 443.
#
#           Option: netapp_login (required, string)
#           NetApp server username.
#
#           Option: netapp_password (required, string)
#           NetApp server password.
#
# Example:
#
# Define an OpenStack storage host:
#
# storage_hosts:
#   lvm-storage1:
#     ip: 172.29.236.131
#
# Example:
#
# Use the LVM iSCSI backend in availability zone 'cinderAZ_1':
#
#     container_vars:
#       cinder_storage_availability_zone: cinderAZ_1
#       cinder_default_availability_zone: cinderAZ_1
#       cinder_backends:
#         lvm:
#           volume_backend_name: LVM_iSCSI
#           volume_driver: cinder.volume.drivers.lvm.LVMVolumeDriver
#           volume_group: cinder-volumes
#           iscsi_ip_address: "{{ cinder_storage_address }}"
#         limit_container_types: cinder_volume
#
# Example:
#
# Use the NetApp iSCSI backend via Data ONTAP 7-mode in availability zone
# 'cinderAZ_2':
```

(continues on next page)

(continued from previous page)

```
#
#     container_vars:
#       cinder_storage_availability_zone: cinderAZ_2
#       cinder_default_availability_zone: cinderAZ_1
#       cinder_backends:
#         netapp:
#           volume_backend_name: NETAPP_iSCSI
#           volume_driver: cinder.volume.drivers.netapp.common.NetAppDriver
#           netapp_storage_family: ontap_7mode
#           netapp_storage_protocol: iscsi
#           netapp_server_hostname: hostname
#           netapp_server_port: 443
#           netapp_login: username
#           netapp_password: password
#
# Example
#
# Use the QNAP iSCSI backend in availability zone
# 'cinderAZ_2':
#
#    container_vars:
#      cinder_storage_availability_zone: cinderAZ_2
#      cinder_default_availability_zone: cinderAZ_1
#      cinder_backends:
#        limit_container_types: cinder_volume
#        qnap:
#          volume_backend_name: "QNAP 1 VOLUME"
#          volume_driver: cinder.volume.drivers.qnap.QnapISCSIDriver
#          qnap_management_url : http://10.10.10.5:8080
#          qnap_poolname: "Storage Pool 1"
#          qnap_storage_protocol: iscsi
#          qnap_server_port: 8080
#          iscsi_ip_address: 172.29.244.5
#          san_login: username
#          san_password: password
#          san_thin_provision: true
#
#
# Example:
#
# Use the ceph RBD backend in availability zone 'cinderAZ_3':
#
#     container_vars:
#       cinder_storage_availability_zone: cinderAZ_3
#       cinder_default_availability_zone: cinderAZ_1
#       cinder_backends:
#         limit_container_types: cinder_volume
#         volumes_hdd:
#           volume_driver: cinder.volume.drivers.rbd.RBDDriver
```

(continues on next page)

(continued from previous page)

```
#           rbd_pool: volumes_hdd
#           rbd_ceph_conf: /etc/ceph/ceph.conf
#           rbd_flatten_volume_from_snapshot: 'false'
#           rbd_max_clone_depth: 5
#           rbd_store_chunk_size: 4
#           rados_connect_timeout: -1
#           volume_backend_name: volumes_hdd
#           rbd_user: "{{ cinder_ceph_client }}"
#           rbd_secret_uuid: "{{ cinder_ceph_client_uuid }}"
#
#
# Example:
#
# Use the cephfs (NATIVE) backend with manila:
#
#     container_vars:
#       manila_default_share_type: cephfs1
#       manila_backends:
#         cephfs1:
#           driver_handles_share_servers: False
#           share_backend_name: CEPHFS1
#           share_driver: manila.share.drivers.cephfs.driver.CephFSDriver
#           cephfs_conf_path: /etc/ceph/ceph.conf
#           cephfs_auth_id: manila
#           cephfs_cluster_name: ceph
#           cephfs_enable_snapshots: False
#           filter_function: "share.size >= 0"
#           goodness_function: "share.size >= 0"
#
#
# Use the cephfs + NFS backend with manila:
#
#     container_vars:
#       manila_default_share_type: cephfsnfs1
#       manila_backends:
#         cephfsnfs1:
#           driver_handles_share_servers: False
#           share_backend_name: CEPHFSNFS1
#           share_driver: manila.share.drivers.cephfs.driver.CephFSDriver
#           cephfs_ganesha_server_ip: 172.16.24.200
#           cephfs_protocol_helper_type: NFS
#           cephfs_conf_path: /etc/ceph/ceph.conf
#           cephfs_auth_id: manila
#           filter_function: "share.size >= 0"
#           goodness_function: "share.size >= 0"
#
#
# Example:
#
```

(continues on next page)

(continued from previous page)

```
# Use the lvm backend with manila:
#
#     container_vars:
#       manila_default_share_type: nfs-share1
#       manila_backends:
#         nfs-share1:
#           share_backend_name: NFS_SHARE1
#           share_driver: manila.share.drivers.lvm.LVMShareDriver
#           driver_handles_share_servers: False
#           lvm_share_volume_group: manila-shares
#           lvm_share_export_ips: 10.1.1.1
#           filter_function: "share.size >= 0"
#           goodness_function: "share.size >= 0"
#
#
#
# Example:
#
# Use the generic backend with manila:
#
#     container_vars:
#       manila_default_share_type: generic
#       manila_backends:
#         generic:
#           share_backend_name: GENERIC
#           share_driver: manila.share.drivers.generic.GenericShareDriver
#           driver_handles_share_servers: true
#           service_instance_flavor_id: 100
#           service_image_name: manila-service-image
#           service_instance_user: manila
#           service_instance_password: manila
#           interface_driver: manila.network.linux.interface.
↪BridgeInterfaceDriver
#           filter_function: "share.size >= 0"
#           goodness_function: "share.size >= 0"
#
#
# --------
#
# Level: load_balancer_hosts (optional)
# List of target hosts on which to deploy HAProxy. Recommend at least one
# target host for this service if hardware load balancers are not being
# used.
#
#   Level: <value> (required, string)
#   Hostname of a target host.
#
#     Option: ip (required, string)
#     IP address of this target host, typically the IP address assigned to
```

(continues on next page)

(continued from previous page)

```
#     the management bridge.
#
#
# Example:
#
# Define a virtual load balancer (HAProxy):
#
# While HAProxy can be used as a virtual load balancer, it is recommended to␣
↪use
# a physical load balancer in a production environment.
#
# load_balancer_hosts:
#   lb1:
#     ip: 172.29.236.100
#   lb2:
#     ip: 172.29.236.101
#
# In case of the above scenario(multiple hosts),HAProxy can be deployed in a
# highly-available manner by installing keepalived.
#
# To make keepalived work, edit at least the following variables
# in ``user_variables.yml``:
#  haproxy_keepalived_external_vip_cidr: 192.168.0.4/25
#  haproxy_keepalived_internal_vip_cidr: 172.29.236.54/16
#  haproxy_keepalived_external_interface: br-flat
#  haproxy_keepalived_internal_interface: br-mgmt
#
# To always deploy (or upgrade to) the latest stable version of keepalived.
# Edit the ``/etc/openstack_deploy/user_variables.yml``:
#   keepalived_package_state: latest
#
# The group_vars/all/keepalived.yml contains the keepalived
# variables that are fed into the keepalived role during
# the haproxy playbook.
# You can change the keepalived behavior for your
# deployment. Refer to the ``user_variables.yml`` file for
# more information.
#
# Keepalived can ping a public and private IP address to check its status.  To
# enable this feature, set the ``keepalived_external_ping_address`` and
# ``keepalived_internal_ping_address`` variables in the ``user_variables.yml``
# file.
```

Inspecting and manipulating the inventory

Warning

Never edit or delete the file `/etc/openstack_deploy/openstack_inventory.json`. This can

lead to problems with the inventory: existng hosts and containers will be unmanaged and new ones will be generated instead, breaking your existing deployment.

The file `scripts/inventory-manage.py` is used to produce human readable output based on the `/etc/openstack_deploy/openstack_inventory.json` file.

The same script can be used to safely remove hosts from the inventory, export the inventory based on hosts, and clear IP addresses from containers within the inventory files.

Operations taken by this script only affect the `/etc/opentstack_deploy/openstack_inventory.json` file; any new or removed information must be set by running playbooks.

Viewing the inventory

The `/etc/openstack_deploy/openstack_inventory.json` file is read by default. An alternative file can be specified with `--file`.

A list of all hosts can be seen with the `--list-host/-l` argument

To see a listing of hosts and containers by their group, use `--list-groups/-g`.

To see all of the containers, use `--list-containers/-G`.

Removing a host

A host can be removed with the `--remove-item/-r` parameter.

Use the hosts name as an argument.

Removing a group

A host group can be removed with the `--remove-group/-d` parameter.

Use the groupss name as an argument. You can repeat argument multiple times to remove several groups at once.

Exporting host information

Information on a per-host basis can be obtained with the `--export/-e` parameter.

This JSON output has two top-level keys: `hosts` and `all`.

`hosts` contains a map of a hosts name to its variable and group data.

`all` contains global network information such as the load balancer IPs and provider network metadata.

Clearing existing container IP addresses

The `--clear-ips` parameter can be used to remove all container IP address information from the `openstack_inventory.json` file. Baremetal hosts will not be changed.

This will *not* change the LXC configuration until the associated playbooks are run and the containers restarted, which will result in API downtime.

Any changes to the containers must also be reflected in the deployments load balancer.

Advanced inventory topics

Changing the base environment directory

The `--environment/-e` argument will take the path to a directory containing an `env.d` directory. This defaults to `inventory/` in the OpenStack-Ansible codebase.

Contents of this directory are populated into the environment *before* the `env.d` found in the directory specified by `--config`.

Dynamic Inventory API documentation

Advanced configuration

The OpenStack-Ansible project provides a basic OpenStack environment, but many deployers will wish to extend the environment based on their needs. This could include installing extra services, changing package versions, or overriding existing variables.

Using these extension points, deployers can provide a more opinionated installation of OpenStack that may include their own software.

Overriding default configuration

Variable precedence

Role defaults

Every role has a file, `defaults/main.yml` which holds the usual variables overridable by a deployer, like a regular Ansible role. This defaults are the closest possible to OpenStack standards.

They can be overridden at multiple levels.

Group vars and host vars

OpenStack-Ansible provides safe defaults for deployers in its group_vars folder. They take care of the wiring between different roles, like for example storing information on how to reach RabbitMQ from nova role.

You can override the existing group vars (and host vars) by creating your own folder in `/etc/openstack_deploy/group_vars` (and `/etc/openstack_deploy/host_vars` respectively).

If you want to change the location of the override folder, you can adapt your `user.rc` file (see details in *Defining environment variables for deployment*), or export `GROUP_VARS_PATH` and `HOST_VARS_PATH` during your shell session.

Role vars

Every role makes use of additional variables in `vars/` which take precedence over group vars.

These variables are typically internal to the role and are not designed to be overridden. However, deployers may choose to override them using extra-vars by placing the overrides into the user variables file.

User variables

If you want to globally override variable, you can define the variable you want to override in a `/etc/openstack_deploy/user_*.yml` file. It will apply on all hosts.

user_*.yml files in more details

Files in `/etc/openstack_deploy` beginning with `user_` will be automatically sourced in any `openstack-ansible` command. Alternatively, the files can be sourced with the `-e` parameter of the `ansible-playbook` command.

`user_variables.yml` and `user_secrets.yml` are used directly by OpenStack-Ansible. Adding custom variables used by your own roles and playbooks to these files is not recommended. Doing so will complicate your upgrade path by making comparison of your existing files with later versions of these files more arduous. Rather, recommended practice is to place your own variables in files named following the `user_*.yml` pattern so they will be sourced alongside those used exclusively by OpenStack-Ansible.

`user_*.yml` files contain YAML variables which are applied as extra-vars when executing `openstack-ansible` to run playbooks. They will be sourced in alphanumeric order by `openstack-ansible`. If duplicate variables occur in the `user_*.yml` files, the variable in the last file read will take precedence.

Setting overrides in configuration files with config_template

All of the services that use YAML, JSON, or INI for configuration can receive overrides through the use of a Ansible action plugin named `config_template`. The configuration template engine allows a deployer to use a simple dictionary to modify or add items into configuration files at run time that may not have a preset template option. All OpenStack-Ansible roles allow for this functionality where applicable. Files available to receive overrides can be seen in the `defaults/main.yml` file as standard empty dictionaries (hashes).

This module was not accepted into Ansible Core (see PR1 and PR2), and will never be.

config_template documentation

These are the options available as found within the virtual module documentation section.

```
module: config_template
version_added: 1.9.2
short_description: >
  Renders template files providing a create/update override interface
description:
  - The module contains the template functionality with the ability to
    override items in config, in transit, through the use of a simple
    dictionary without having to write out various temp files on target
    machines. The module renders all of the potential jinja a user could
    provide in both the template file and in the override dictionary which
    is ideal for deployers who may have lots of different configs using a
    similar code base.
  - The module is an extension of the **copy** module and all of attributes
    that can be set there are available to be set here.
options:
  src:
```

(continues on next page)

(continued from previous page)

```
    description:
      - Path of a Jinja2 formatted template on the local server. This can
        be a relative or absolute path.
    required: true
    default: null
  dest:
    description:
      - Location to render the template to on the remote machine.
    required: true
    default: null
  config_overrides:
    description:
      - A dictionary used to update or override items within a configuration
        template. The dictionary data structure may be nested. If the target
        config file is an ini file the nested keys in the ``config_overrides``
        will be used as section headers.
  config_type:
    description:
      - A string value describing the target config type.
    choices:
      - ini
      - json
      - yaml
```

Example task using the config_template module

In this task the `test.ini.j2` file is a template which will be rendered and written to disk at `/tmp/test.ini`. The **config_overrides** entry is a dictionary (hash) which allows a deployer to set arbitrary data as overrides to be written into the configuration file at run time. The **config_type** entry specifies the type of configuration file the module will be interacting with; available options are yaml, json, and ini.

```
- name: Run config template ini
  config_template:
    src: test.ini.j2
    dest: /tmp/test.ini
    config_overrides: "{{ test_overrides }}"
    config_type: ini
```

Here is an example override dictionary (hash):

```
test_overrides:
  DEFAULT:
    new_item: 12345
```

And here is the template file:

```
[DEFAULT]
value1 = abc
value2 = 123
```

The rendered file on disk, namely `/tmp/test.ini` looks like this:

```
[DEFAULT]
value1 = abc
value2 = 123
new_item = 12345
```

Discovering available overrides

All of these options can be specified in any way that suits your deployment. In terms of ease of use and flexibility its recommended that you define your overrides in a user variable file such as `/etc/openstack_deploy/user_variables.yml`.

The list of overrides available may be found by executing:

```
ls /etc/ansible/roles/*/defaults/main.yml -1 \
    | xargs -I {} grep '_.*_overrides:' {} \
    | grep -Ev "^#|^\s" \
    | sort -u
```

Note

Possible additional overrides can be found in the Tunable Section of each roles `main.yml` file, such as `/etc/ansible/roles/role_name/defaults/main.yml`.

Overriding OpenStack configuration defaults

OpenStack has many configuration options available in `.conf` files (in a standard `INI` file format), policy files (in a standard `JSON` format) and `YAML` files, and can therefore use the `config_template` module described above.

OpenStack-Ansible enables you to reference any options in the OpenStack Configuration Reference through the use of a simple set of configuration entries in the `/etc/openstack_deploy/user_variables.yml`.

Overriding .conf files

Most often, overrides are implemented for the `<service>.conf` files (for example, `nova.conf`). These files use a standard INI file format.

For example, you might want to add the following parameters to the `nova.conf` file:

```
[DEFAULT]
remove_unused_original_minimum_age_seconds = 43200

[libvirt]
cpu_mode = host-model
disk_cachemodes = file=directsync,block=none

[database]
idle_timeout = 300
max_pool_size = 10
```

To do this, you use the following configuration entry in the `/etc/openstack_deploy/user_variables.yml` file:

```
nova_nova_conf_overrides:
  DEFAULT:
    remove_unused_original_minimum_age_seconds: 43200
  libvirt:
    cpu_mode: host-model
    disk_cachemodes: file=directsync,block=none
  database:
    idle_timeout: 300
    max_pool_size: 10
```

Note

The general format for the variable names used for overrides is `<service>_<filename>_<file extension>_overrides`. For example, the variable name used in these examples to add parameters to the `nova.conf` file is `nova_nova_conf_overrides`.

The same way you can apply overrides to the uwsgi services. For example:

```
nova_api_os_compute_uwsgi_ini_overrides:
  uwsgi:
    limit: 1024
```

Note

Some roles, like uwsgi, are used for lot of roles, and have special overrides, (like *uwsgi_ini_overrides*) which can be defined to impact all services which are using uwsgi. These variables are special as they will have precedence over role defined **_uwsgi_ini_overrides*.

You can also apply overrides on a per-host basis with the following configuration in the `/etc/openstack_deploy/openstack_user_config.yml` file:

```
compute_hosts:
  900089-compute001:
    ip: 192.0.2.10
    host_vars:
      nova_nova_conf_overrides:
        DEFAULT:
          remove_unused_original_minimum_age_seconds: 43200
        libvirt:
          cpu_mode: host-model
          disk_cachemodes: file=directsync,block=none
        database:
          idle_timeout: 300
          max_pool_size: 10
```

Use this method for any files with the `INI` format for in OpenStack projects deployed in OpenStack-Ansible.

Overriding .json files

To implement access controls that are different from the ones in a standard OpenStack environment, you can adjust the default policies applied by services. Policy files are in a `JSON` format.

For example, you might want to add the following policy in the `policy.json` file for the Identity service (keystone):

```
{
    "identity:foo": "rule:admin_required",
    "identity:bar": "rule:admin_required"
}
```

To do this, you use the following configuration entry in the `/etc/openstack_deploy/user_variables.yml` file:

```
keystone_policy_overrides:
  identity:foo: "rule:admin_required"
  identity:bar: "rule:admin_required"
```

Note

The general format for the variable names used for overrides is `<service>_policy_overrides`. For example, the variable name used in this example to add a policy to the Identity service (keystone) `policy.json` file is `keystone_policy_overrides`.

Use this method for any files with the `JSON` format in OpenStack projects deployed in OpenStack-Ansible.

To assist you in finding the appropriate variable name to use for overrides, the general format for the variable name is `<service>_policy_overrides`.

Overriding .yml files

You can override `.yml` file values by supplying replacement YAML content.

Note

All default YAML file content is completely overwritten by the overrides, so the entire YAML source (both the existing content and your changes) must be provided.

For example, you might want to define a meter exclusion for all hardware items in the default content of the `pipeline.yml` file for the Telemetry service (ceilometer):

```
sources:
    - name: meter_source
    interval: 600
    meters:
        - "!hardware.*"
    sinks:
        - meter_sink
```

(continues on next page)

(continued from previous page)

```
    - name: foo_source
      value: foo
```

To do this, you use the following configuration entry in the `/etc/openstack_deploy/user_variables.yml` file:

```
ceilometer_pipeline_yaml_overrides:
  sources:
      - name: meter_source
        interval: 600
        meters:
            - "!hardware.*"
        sinks:
            - meter_sink
        - name: source_foo
        value: foo
```

Note

The general format for the variable names used for overrides is `<service>_<filename>_<file extension>_overrides`. For example, the variable name used in this example to define a meter exclusion in the `pipeline.yml` file for the Telemetry service (ceilometer) is `ceilometer_pipeline_yaml_overrides`.

Overriding OpenStack upper constraints

Each OpenStack release uses an `upper-constraints.txt` file to define the maximum permitted version of each Python package. In some cases it may be necessary to override this file, for example when your local deployment needs to take advantage of a bug fix. Care should be taken when modifying this file as OpenStack services may not have been tested against more recent package versions.

To override the upper constraints for a deployment, clone the OpenStack requirements git repository, either storing it as a fork at a URL of your choice, or within the local filesystem of the host you are using to deploy OpenStack Ansible from. Once cloned, switch to the branch which matches the name of your deployed OpenStack version, and modify the upper constraints as required.

Next, edit your `/etc/openstack_deploy/user_variables.yml` file to indicate the path to the requirements git repository, and the git hash of the commit containing your changes using the `requirements_git_repo` and `requirements_git_install_branch` variables. When using the local filesystem, the `requirements_git_repo` should start with `file://`.

Finally, run the `repo-install.yml` playbook to upload these modified constraints to your repo host(s).

Extending OpenStack-Ansible with additional Ansible content

Including OpenStack-Ansible in your project

Including the OpenStack-Ansible repository within another project can be done in several ways:

- A git submodule pointed to a released tag.
- A script to automatically perform a git checkout of OpenStack-Ansible.

When including OpenStack-Ansible in a project, consider using a parallel directory structure as shown in the `ansible.cfg` files section.

Also note that copying files into directories such as `env.d` or `conf.d` should be handled via some sort of script within the extension project.

Including OpenStack-Ansible with your Ansible structure

You can create your own playbook, variable, and role structure while still including the OpenStack-Ansible roles and libraries by setting environment variables or by adjusting `/usr/local/bin/openstack-ansible.rc`.

The relevant environment variables for OpenStack-Ansible are as follows:

ANSIBLE_LIBRARY
: This variable should point to `/etc/ansible/plugins/library`. Doing so allows roles and playbooks to access OpenStack-Ansibles included Ansible modules.

ANSIBLE_ROLES_PATH
: This variable should point to `/etc/ansible/roles` by default. This allows Ansible to properly look up any OpenStack-Ansible roles that extension roles may reference.

ANSIBLE_INVENTORY
: This variable should point to `openstack-ansible/inventory/dynamic_inventory.py`. With this setting, extensions have access to the same dynamic inventory that OpenStack-Ansible uses.

The paths to the `openstack-ansible` top level directory can be relative in this file.

Consider this directory structure:

```
my_project
|
|- custom_stuff
|  |
|  |- playbooks
|- openstack-ansible
|  |
|  |- playbooks
```

The environment variables set would use `../openstack-ansible/playbooks/<directory>`.

Adding new or overriding roles in your OpenStack-Ansible installation

By default OpenStack-Ansible uses its ansible-role-requirements file to fetch the roles it requires for the installation process.

The roles will be fetched into the standard `ANSIBLE_ROLES_PATH`, which defaults to `/etc/ansible/roles`.

`ANSIBLE_ROLE_FILE` is an environment variable pointing to the location of a YAML file which ansible-galaxy can consume, specifying which roles to download and install. The default value for this is `ansible-role-requirements.yml`.

To completely override the `ansible-role-requirement.yml` file you can define the environment variable `ANSIBLE_ROLE_FILE` before running the `bootstrap-ansible.sh` script. With this approach it

is now the responsibility of the deployer to maintain appropriate versions pins of the Ansible roles if an upgrade is required.

If you want to extend or just partially override content of the `ansible-role-requirements.yml` file you can create a new config file which path defaults to `/etc/openstack_deploy/user-role-requirements.yml`. This path can be overriden with another environment variable `USER_ROLE_FILE` which is expected to be relative to `OSA_CONFIG_DIR` (/etc/openstack_deploy) folder.

This file is in the same format as `ansible-role-requirements.yml` and can be used to add new roles or selectively override existing ones. New roles listed in `user-role-requirements.yml` will be merged with those in `ansible-role-requirements.yml`, and roles with matching `name` key will override those in `ansible-role-requirements.yml`. In case when `src` key is not defined bootstrap script will skip cloning such roles.

It is easy for a deployer to keep this file under their own version control and out of the OpenStack-Ansible tree.

Adding new or overriding collections in your OpenStack-Ansible installation

Alike to roles, collections for installation are stored in ansible-collection-requirements file. Path to this file can be overriden through `ANSIBLE_COLLECTION_FILE` environmental variable.

The Victoria release of OpenStack-Ansible adds an optional new config file which defaults to `/etc/openstack_deploy/user-collection-requirements.yml`.

It should be in the native format of the ansible-galaxy requirements file and can be used to add new collections to the deploy host or override versions or source for collections defined in `ansible-collection-requirements`.

`user-collection-requirements` will be merged with `ansible-collection-requirements` using collection `name` as a key. In case `source` is not defined in `user-collection-requirements`, collection installation will be skipped. This way you can skip installation of unwanted collections.

You can override location of the `user-collection-requirements.yml` by setting `USER_COLLECTION_FILE` environment variable before running the `bootstrap-ansible.sh` script. Though it is expected to be relative to `OSA_CONFIG_DIR` (/etc/openstack_deploy) folder.

Calling extra playbooks during the deployment

If you install some additional deployment functionality as either a collection or a git repository on the deploy host, it is possible to automatically include extra playbooks at certain points during the deployment.

The points where a hook exists to call an external playbook are as follows:

- `pre_setup_hosts_hook`
- `post_setup_hosts_hook`
- `pre_setup_infrastructure_hook`
- `post_setup_infrastructure_hook`
- `pre_setup_openstack_hook`
- `post_setup_openstack_hook`

The hook variables should be configured in a suitable `user_variables.yml` file. An example calling a playbook from a collection (installed using `user-collection-requirements.yml`):

```
pre_setup_hosts_hook: custom.collection.playbook
```

Installing extra playbooks using collections, and referencing the playbook with its FQCN is the most robust approach to including additional user defined playbooks.

Installing extra Python packages inside Ansible virtualenv

Some Ansible collections may require presence of specific Python libraries inside execution environment. In order to accomplish that deployer can create `/etc/openstack_deploy/user-ansible-venv-requirements.txt` file with a list of Python libraries that should be installed inside virtual environment along with Ansible during `bootstrap-ansible.sh` execution.

You can override the default path to `user-ansible-venv-requirements.txt` file with `USER_ANSIBLE_REQUIREMENTS_FILE` environment variable before running the `bootstrap-ansible.sh` script.

Defining environment variables for deployment

Throughout the documentation we talk a lot about different environment variables that control behaviour of OpenStack-Ansible and Ansible iteself.

Starting with the Zed release a `user.rc` file can be placed in `OSA_CONFIG_DIR` (/etc/openstack_deploy) folder and contain any environment variable definitions that might be needed to change the default behaviour or any arbitrary ansible-configuration parameter. These environment variables are general purpose and are not limited to those understood by Ansible.

The path to this file can be changed by setting the `OSA_USER_RC` variable, but the `OSA_CONFIG_DIR` and `OSA_USER_RC` variables cannot re-defined or controlled through the `user.rc` file.

Adding extra python software

The system will allow you to install and build any package that is a python installable. The repository infrastructure will look for and create any git based or PyPi installable package. When the package is built the repo-build role will create the sources as Python wheels to extend the base system and requirements.

While the pre-built packages in the repository-infrastructure are comprehensive, it may be needed to change the source locations and versions of packages to suit different deployment needs. Adding additional repositories as overrides is as simple as listing entries within the variable file of your choice. Any `user_.*.yml` file within the /etc/openstack_deployment directory will work to facilitate the addition of a new packages.

```
swift_git_repo: https://private-git.example.org/example-org/swift
swift_git_install_branch: master
```

Additional lists of python packages can also be overridden using a `user_.*.yml` variable file.

```
swift_requires_pip_packages:
  - virtualenv
  - python-keystoneclient
  - NEW-SPECIAL-PACKAGE
```

Once the variables are set call the play `repo-build.yml` to build all of the wheels within the repository infrastructure. When ready run the target plays to deploy your overridden source code.

Adding extra network to container

In some cases it may be useful to have an ability to add extra network interface for some container group (or just a single container). As an example this can be used for applying known fixed IP address from another network for Designate service. We will show futher configuration based on this example. Lets assume, that this network is 10.0.20.0/24 which is reachable through *br-dns* interface.

To add new interface with that network into dessignate containers, we need to do several actions in `openstack_user_config.yml`.

Note

You may find detailed example of *openstack_user_config.yml* configuration in section *openstack_user_config settings reference*.

- Add this network in `cidr_networks`:

```
cidr_networks:
  container: 172.29.236.0/22
  tunnel: 172.29.240.0/22
  storage: 172.29.244.0/22
  designate: 10.0.20.0/24
```

- Describe network in `provider_networks`:

```
global_overrides:
  provider_networks:
    - network:
      container_bridge: "br-dns"
      container_type: "veth"
      container_interface: "eth5"
      ip_from_q: "designate"
      type: "veth"
      group_binds:
        - dnsaas_hosts
```

- Define override for containers

Note

Adding gateway key will create default route inside container through it

```
dnsaas_hosts:
  aio1:
    ip: 172.29.236.100
    container_vars:
      container_extra_networks:
        dns_address:
          bridge: br-dns
          interface: eth5
```

(continues on next page)

(continued from previous page)

```
        address: 10.0.20.100
        netmask: 255.255.255.0
        gateway: 10.0.20.1
```

Using SR-IOV interfaces in containers

For some deployments it might be required to passthrough devices directly to containers, for example, when SR-IOV is used or devices cant be bridged (ie with *IPoIB <https://www.kernel.org/doc/html/latest/infiniband/ipoib.html>*)

You would need to manually map physical interfaces to specific containers. This also assumes, that same interface name is present on all containers and it is consistent and present before LXC startup.

Below as an example we will try using IB interfaces for storage network and pass them inside containers that require storage connectivity. For that you need describe connections in `provider_networks` inside *openstack_user_config.yml* configuration:

```
global_overrides:
  provider_networks:
    - network:
        container_bridge: "ib1"
        container_type: "phys"
        container_interface: "ib1"
        ip_from_q: "storage"
        type: "raw"
        group_binds:
          - cinder_volume
    - network:
        container_bridge: "ib3"
        container_type: "phys"
        container_interface: "ib3"
        ip_from_q: "storage"
        type: "raw"
        group_binds:
          - glance_api
    - network:
        container_bridge: "ib5"
        container_type: "phys"
        container_interface: "ib5"
        ip_from_q: "storage"
        type: "raw"
        group_binds:
          - gnocchi_api
```

Architecture

Many operational requirements have been taken into consideration for the design of the OpenStack-Ansible project.

In this chapter, you can find details about *why* OpenStack-Ansible was architected in this way.

OpenStack-Ansible Manifesto

Project scope

This will be a **Batteries included** project. Which means deployer can expect that deploying from any of the named feature branches or tags should provide an OpenStack cloud built for production which will be available at the successful completion of the deployment.

However, this project solely focuses on the deployment of OpenStack and its requirements.

This project does **not** PXE boot hosts. Host setup and lifecycle management is left to the deployer. This project also requires that bridges are setup within the hosts to allow the containers to attach to a local bridge for network access. See also the *Container networking*.

Ansible Usage

Ansible provides an automation platform to simplify system and application deployment. Ansible manages systems by using Secure Shell (SSH) instead of unique protocols that require remote daemons or agents.

Ansible uses playbooks written in the YAML language for orchestration. For more information, see Ansible playbooks.

Ansible is a simple yet powerful orchestration tool that is ideally equipped for deploying OpenStack-powered clouds. The declarative nature of Ansible allows the deployer to turn an entire deployment into a rather simple set of instructions.

Roles within the OpenStack-Ansible umbrella are built using Ansible best practices and contain namespaced variables that are *human* understandable. All roles are independant of each other and testable separately.

All roles are built as Galaxy compatible roles even when the given role is not intended for standalone use. While the project will offer a lot of built-in roles the deployer will be able to pull down or override roles with external ones using the built-in Ansible capabilities. This allows extreme flexibility for deployers.

Source based deployments

When the OpenStack-Ansible project was created, it was required to provide a system able to override any OpenStack upstream source code.

This means that OpenStack services and their python dependencies are built and installed from source code as found within the OpenStack Git repositories by default, but allow deployers to point to their own repositories.

This also allows developers to point to their own code for their work.

A source based deployment, for Python-built parts of OpenStack, makes sense when dealing with scale and wanting consistency over long periods of time. A deployer should have the ability to deploy the same OpenStack release on every node throughout the life cycle of the cloud, even when some components are end of life. By providing a repository of the sources, the deployment can be re-created even years after the initial deployment, assuming the underlying operating systems and packages stay the same.

This means that there will never be a time where OpenStack specific packages, as provided by the distributions, are being used for OpenStack services. Third party repositories like *CloudArchive* and or *RDO* may still be required within a given deployment but only as a means to meet application dependencies.

Containerized deployments

This project introduces containers as a means to abstract services from one another.

The use of containers allows for additional abstractions of entire application stacks to be run all within the same physical host machines.

The containerized applications are sometimes grouped within a single container where it makes sense, or distributed in multiple containers based on application and or architectural needs.

The default container architecture has been built in such a way to allow for scalability and highly available deployments.

The simple nature of machine containers allows the deployer to treat containers as physical machines. The same concepts apply for machine containers and physical machines: This will allow deployers to use existing operational tool sets to troubleshoot issues within a deployment and the ability to revert an application or service within inventory to a known working state without having to re-kick a physical host.

Not all services are containerized: some dont make sense to run within a container. Logic needs to be applied in regards on how services are containerized. If their requirements cant be met due to system limitations, (kernel, application maturity, etc), then the service is not set to run within a container.

Example of un-containerized services:

- Nova compute (for direct access to virtualization devices)
- Swift storage (for direct access to drive)

The containers are not a mean of securing a system. The containers were not chosen for any eventual security safe guards. The machine containers were chosen because of their practicality with regard to providing a more uniform OpenStack deployment. Even if the abstractions that the containers provides do improve overall deployment security these potential benefits are not the intention of the containerization of services.

Security

Security is one of the top priorities within OpenStack-Ansible (OSA), and many security enhancements for OpenStack clouds are available in deployments by default. This section provides a detailed overview of the most important security enhancements.

Note

Every deployer has different security requirements. The OpenStack Security Guide has instructions and advice on how to operate and consume an OpenStack cloud by using the most secure methods.

Encrypted communication

Any OpenStack cloud has sensitive information transmitted between services, including user credentials, service credentials or information about resources being created. Encrypting this traffic is critical in environments where the network cannot be trusted. (For more information about securing the network, see the *Securing network access to OpenStack services* section.)

Many of the services deployed with OpenStack-Ansible are encrypted by default or offer encryption as an option. The playbooks generate self-signed certificates by default, but deployers have the option to use their existing certificates, keys, and CA certificates.

To learn more about how to customize the deployment of encrypted communications, see *Securing services with SSL certificates*.

Host security hardening

OpenStack-Ansible provides a comprehensive security hardening role that applies over 200 security configurations as recommended by the Security Technical Implementation Guide (STIG) provided by the Defense Information Systems Agency (DISA). These security configurations are widely used and are distributed in the public domain by the United States government.

Host security hardening is required by several compliance and regulatory programs, such as the Payment Card Industry Data Security Standard (PCI DSS) (Requirement 2.2).

By default, OpenStack-Ansible automatically applies the ansible-hardening role to all deployments. The role has been carefully designed to perform as follows:

- Apply nondisruptively to a production OpenStack environment
- Balance security with OpenStack performance and functionality
- Run as quickly as possible

For more information about the security configurations, see the security hardening role documentation.

Isolation

By default, OpenStack-Ansible provides isolation by default between the containers that run the OpenStack infrastructure (control plane) services and also between the virtual machines that end users spawn within the deployment. This isolation is critical because it can prevent container or virtual machine breakouts, or at least reduce the damage that breakouts might cause.

The Linux Security Modules (LSM) framework allows administrators to set mandatory access controls (MAC) on a Linux system. MAC is different than discretionary access controls (DAC) because the kernel enforces strict policies that no user can bypass. Although any user might be able to change a DAC policy (such as `chown bob secret.txt`), only the `root` user can alter a MAC policy.

OpenStack-Ansible currently uses AppArmor to provide MAC policies on infrastructure servers and hypervisors. The AppArmor configuration sets the access policies to prevent one container from accessing the data of another container. For virtual machines, `libvirtd` uses the sVirt extensions to ensure that one virtual machine cannot access the data or devices from another virtual machine.

These policies are applied and governed at the kernel level. Any process that violates a policy is denied access to the resource. All denials are logged in `auditd` and are available at `/var/log/audit/audit.log`.

Least privilege

The principle of least privilege is used throughout OpenStack-Ansible to limit the damage that could be caused if an attacker gains access to any credentials.

OpenStack-Ansible configures unique username and password combinations for each service that interacts with RabbitMQ and Galera/MariaDB. Each service that connects to RabbitMQ uses a separate virtual host for publishing and consuming messages. The MariaDB users for each service are only granted access only to the databases that they need to query.

You can also run OpenStack-Ansible with non-root user by leveraging the Ansible privilege escalation method. For more details, please reffer to the *running as non-root*.

Securing network access to OpenStack services

OpenStack clouds provide many services to end users, that enable them to build instances, provision storage, and create networks. Each of these services exposes one or more service ports and API endpoints to the network.

However, some of the services within an OpenStack cloud are accessible to all end users, while others are accessible only to administrators or operators on a secured network.

- Services that *all end users* can access
  - These services include Compute (nova), Object Storage (swift), Networking (neutron), and Image (glance).
  - These services should be offered on a sufficiently restricted network that still allows all end users to access the services.
  - A firewall must be used to restrict access to the network.
- Services that *only administrators or operators* can access
  - These services include MariaDB, Memcached, RabbitMQ, and the admin API endpoint for the Identity (keystone) service.
  - These services *must* be offered on a highly restricted network that is available only to administrative users.
  - A firewall must be used to restrict access to the network.

Limiting access to these networks has several benefits:

- Allows for network monitoring and alerting
- Prevents unauthorized network surveillance
- Reduces the chance of credential theft
- Reduces damage from unknown or unpatched service vulnerabilities

OpenStack-Ansible deploys HAProxy back ends for each service and restricts access for highly sensitive services by making them available only on the management network. Deployers with external load balancers must ensure that the back ends are configured securely and that firewalls prevent traffic from crossing between networks.

For more information about recommended network policies for OpenStack clouds, see the API endpoint process isolation and policy section of the OpenStack Security Guide

Service architecture

Introduction

OpenStack-Ansible (OSA) has a flexible deployment configuration model that can deploy all services in separate machine containers or on designated hosts without using containers, and all network traffic either on a single network interface or on many network interfaces.

This flexibility enables deployers to choose how to deploy OpenStack in the appropriate way for the specific use case.

The following sections describe the services that OpenStack-Ansible deploys.

Infrastructure services

OpenStack-Ansible deploys the following infrastructure components:

- MariaDB with Galera

  All OpenStack services require an underlying database. MariaDB with Galera implements a multi-master database configuration, which simplifies its use as a highly available database with a simple failover model.

- RabbitMQ

  OpenStack services use RabbitMQ for Advanced Message Queuing Protocol (AMQP). OpenStack-Ansible deploys RabbitMQ in a clustered configuration with all queues mirrored between the cluster nodes. Because Telemetry (ceilometer) message queue traffic is quite heavy, for large environments we recommend separating Telemetry notifications into a separate RabbitMQ cluster.

- Memcached

  OpenStack services use Memcached for in-memory caching, which accelerates transactions. For example, the OpenStack Identity service (Keystone) uses Memcached for caching authentication tokens, which ensures that token validation does not have to complete a disk or database transaction every time the service is asked to validate a token.

- Repository

  The repository holds the reference set of artifacts that are used for the installation of the environment. The artifacts include:

  - A Git repository that contains a copy of the source code that is used to prepare the packages for all OpenStack services
  - Python wheels for all services that are deployed in the environment
  - An apt/yum proxy cache that is used to cache distribution packages installed in the environment

- Load balancer

  At least one load balancer is required for a deployment. OpenStack-Ansible provides a deployment of HAProxy, but we recommend using a physical load balancing appliance for production environments.

- Utility container

  If a tool or object does not require a dedicated container, or if it is impractical to create a new container for a single tool or object, it is installed in the utility container. The utility container is also used when tools cannot be installed directly on a host. The utility container is prepared with the appropriate credentials and clients to administer the OpenStack environment. It is set to automatically use the internal service endpoints.

- Unbound DNS container

  Containers running an Unbound DNS caching service can optionally be deployed to cache DNS lookups and to handle internal DNS name resolution. We recommend using this service for large-scale production environments because the deployment will be significantly faster. If this service is not used, OpenStack-Ansible modifies `/etc/hosts` entries for all hosts in the environment.

OpenStack services

OpenStack-Ansible is able to deploy a multitude of services. Have a look at the role maturity matrix to know the status of the service you want to deploy.

Storage architecture

OpenStack has multiple storage realms to consider:

- Block Storage (cinder)
- Object Storage (swift)
- Image storage (glance)
- Ephemeral storage (nova)
- Shared Filesystems storage (manila)

Block Storage (cinder)

The Block Storage (cinder) service manages volumes on storage devices in an environment. In a production environment, the device presents storage via a storage protocol (for example, NFS, iSCSI, or Ceph RBD) to a storage network (`br-storage`) and a storage management API to the management network (`br-mgmt`). Instances are connected to the volumes via the storage network by the hypervisor on the Compute host.

The following diagram illustrates how Block Storage is connected to instances.

Important

The LVMVolumeDriver is designed as a reference driver implementation, which we do not recommend for production usage. The LVM storage back-end is a single-server solution that provides no high-availability options. If the server becomes unavailable, then all volumes managed by the `cinder-volume` service running on that server become unavailable. Upgrading the operating system packages (for example, kernel or iSCSI) on the server causes storage connectivity outages because the iSCSI service (or the host) restarts.

Because of a limitation with container iSCSI connectivity, you must deploy the `cinder-volume` service directly on a physical host (not into a container) when using storage back ends that connect via iSCSI. This includes the LVMVolumeDriver and many of the drivers for commercial storage devices.

Note

The `cinder-volume` service does not run in a highly available configuration. When the `cinder-volume` service is configured to manage volumes on the same back end from multiple hosts or containers, one service is scheduled to manage the life cycle of the volume until an alternative service is assigned to do so. This assignment can be made through the cinder-manage CLI tool. This configuration might change if cinder volume active-active support spec is implemented.

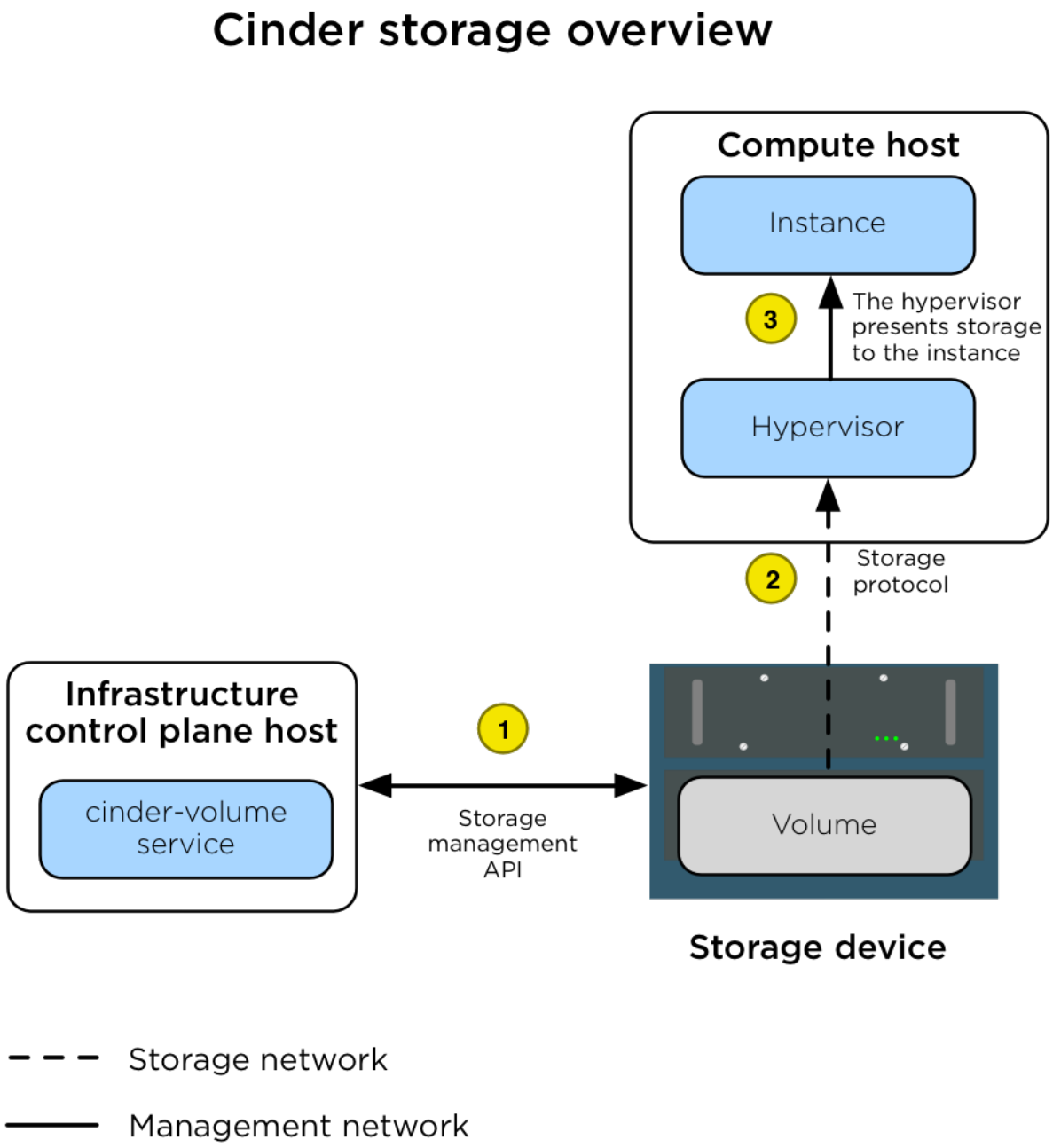


Fig. 6: The diagram shows the following steps.

1.	A volume is created by the assigned `cinder-volume` service using the appropriate cinder driver. The volume is created by using an API that is presented to the management network.
2.	After the volume is created, the `nova-compute` service connects the Compute host hypervisor to the volume via the storage network.
3.	After the hypervisor is connected to the volume, it presents the volume as a local hardware device to the instance.

Object Storage (swift)

The Object Storage (swift) service implements a highly available, distributed, eventually consistent object/blob store that is accessible via HTTP/HTTPS.

The following diagram illustrates how data is accessed and replicated.

Image storage (glance)

The Image service (glance) can be configured to store images on a variety of storage back ends supported by the glance_store drivers.

Important

When the File System store is used, the Image service has no mechanism of its own to replicate the image between Image service hosts. We recommend using a shared storage back end (via a file system mount) to ensure that all`glance-api` services have access to all images. Doing so prevents losing access to images when an infrastructure (control plane) host is lost.

The following diagram illustrates the interactions between the Image service, the storage device, and the `nova-compute` service when an instance is created.

Ephemeral storage (nova)

When the flavors in the Compute service are configured to provide instances with root or ephemeral disks, the `nova-compute` service manages these allocations using its ephemeral disk storage location.

In many environments, the ephemeral disks are stored on the Compute hosts local disks, but for production environments we recommend that the Compute hosts be configured to use a shared storage subsystem instead. A shared storage subsystem allows quick, live instance migration between Compute hosts, which is useful when the administrator needs to perform maintenance on the Compute host and wants to evacuate it. Using a shared storage subsystem also allows the recovery of instances when a Compute host goes offline. The administrator is able to evacuate the instance to another Compute host and boot it up again. The following diagram illustrates the interactions between the storage device, the Compute host, the hypervisor, and the instance.

Shared Filesystems storage (manila)

The shared filesystem service (manila) can be configured to provide file systems on a variety of storage back ends as supported by the manila_store drivers.

Container networking

OpenStack-Ansible deploys Linux containers (LXC) and uses Linux or Open vSwitch-based bridging between the container and the host interfaces to ensure that all traffic from containers flows over multiple host interfaces. All services in this deployment model use a *unique* IP address.

This appendix describes how the interfaces are connected and how traffic flows.

For more information about how the OpenStack Networking service (Neutron) uses the interfaces for instance traffic, please see the OpenStack Networking Guide.

For details on the configuration of networking for your environment, please have a look at *openstack_user_config settings reference*.

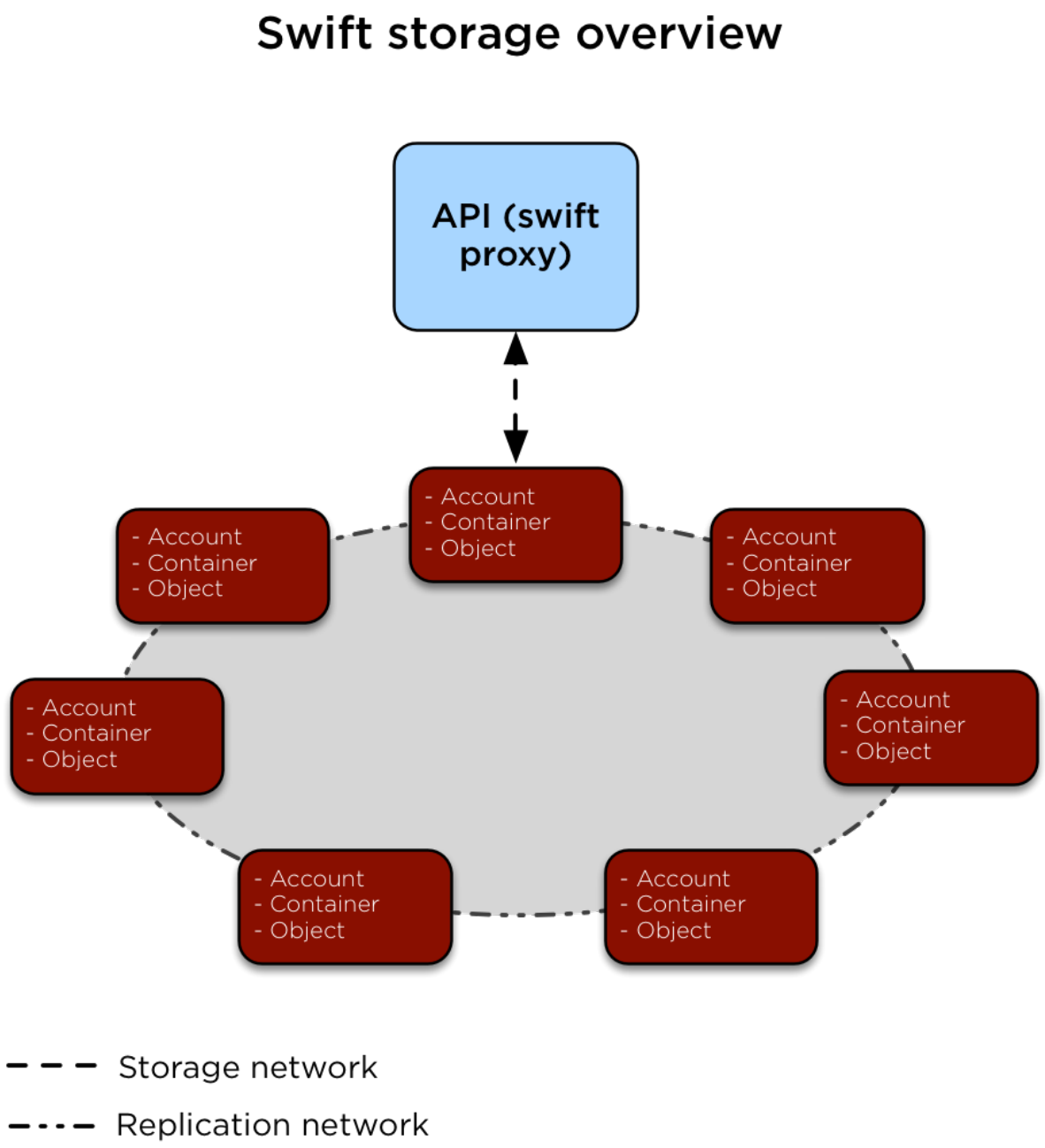


Fig. 7: The `swift-proxy` service is accessed by clients via the load balancer on the management network (`br-mgmt`). The `swift-proxy` service communicates with the Account, Container, and Object services on the Object Storage hosts via the storage network(`br-storage`). Replication between the Object Storage hosts is done via the replication network (`br-repl`).

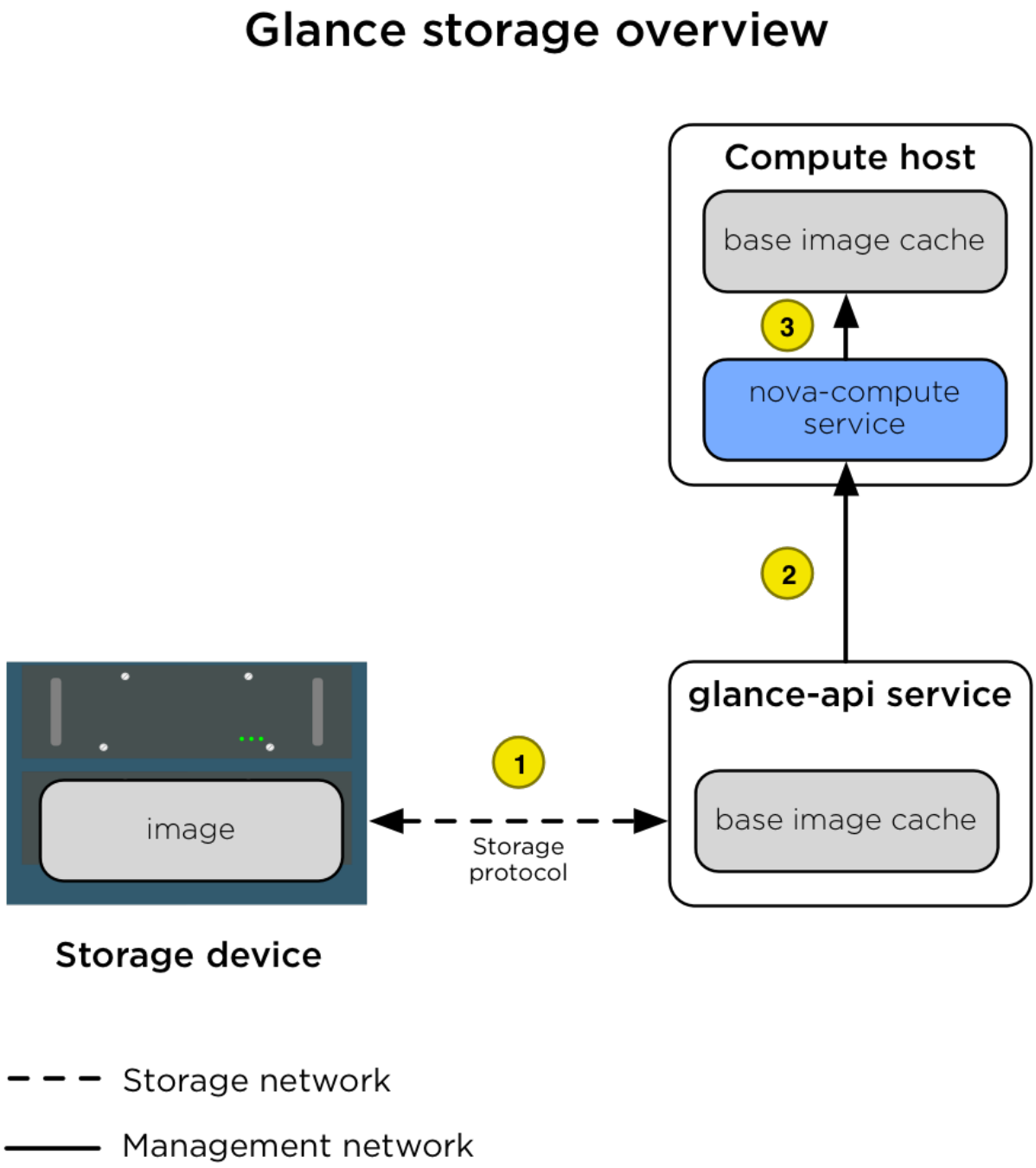


Fig. 8: The diagram shows the following steps.

1	When a client requests an image, the `glance-api` service accesses the appropriate store on the storage device over the storage network (`br-storage`) and pulls it into its cache. When the same image is requested again, it is given to the client directly from the cache.
2	When an instance is scheduled for creation on a Compute host, the `nova-compute` service requests the image from the `glance-api` service over the management network (`br-mgmt`).
3	After the image is retrieved, the `nova-compute` service stores the image in its own image cache. When another instance is created with the same image, the image is retrieved from the local base image cache.

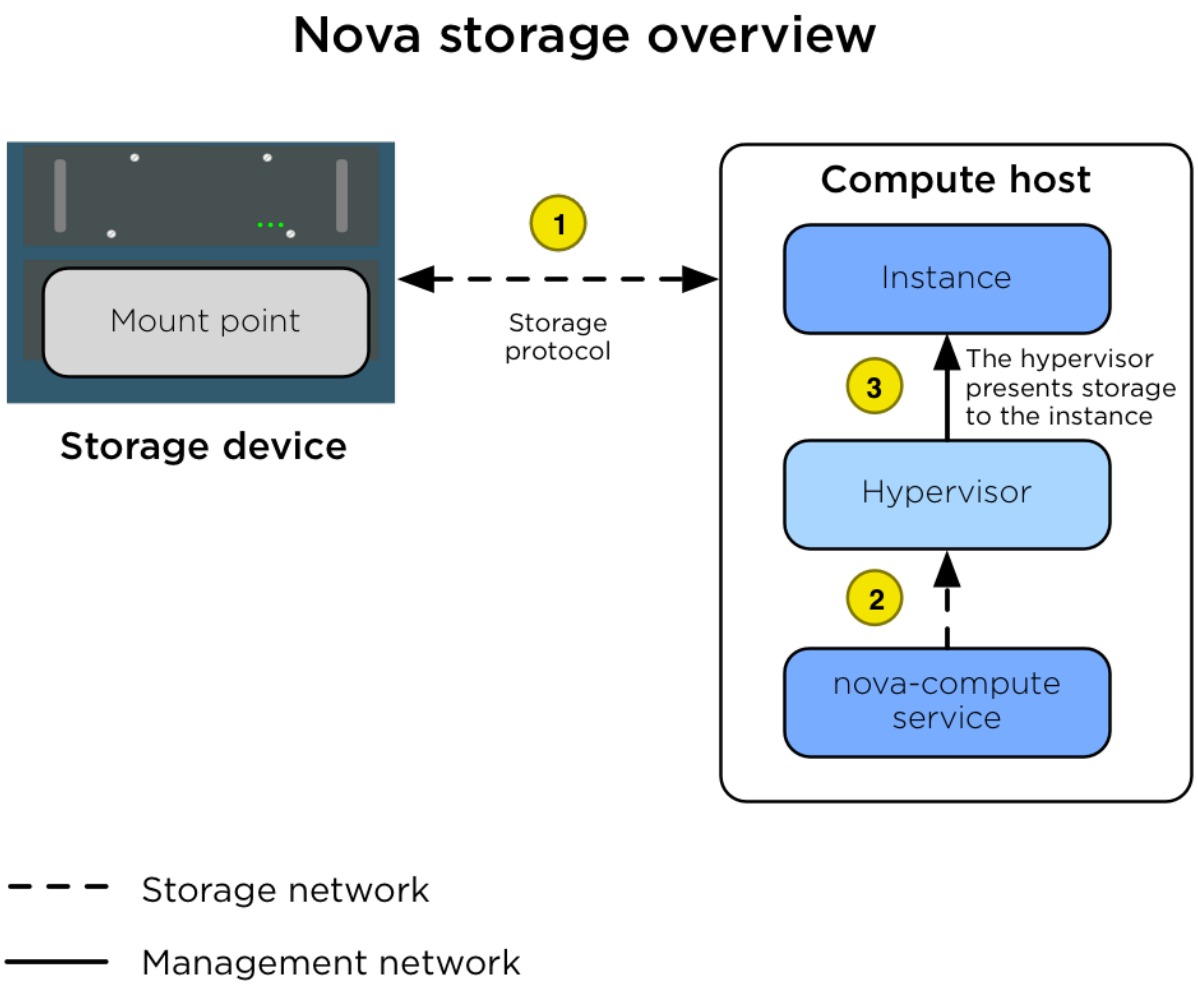


Fig. 9: The diagram shows the following steps.

1	The Compute host is configured with access to the storage device. The Compute host accesses the storage space via the storage network (`br-storage`) by using a storage protocol (for example, NFS, iSCSI, or Ceph RBD).
2	The `nova-compute` service configures the hypervisor to present the allocated instance disk as a device to the instance.
3	The hypervisor presents the disk as a device to the instance.

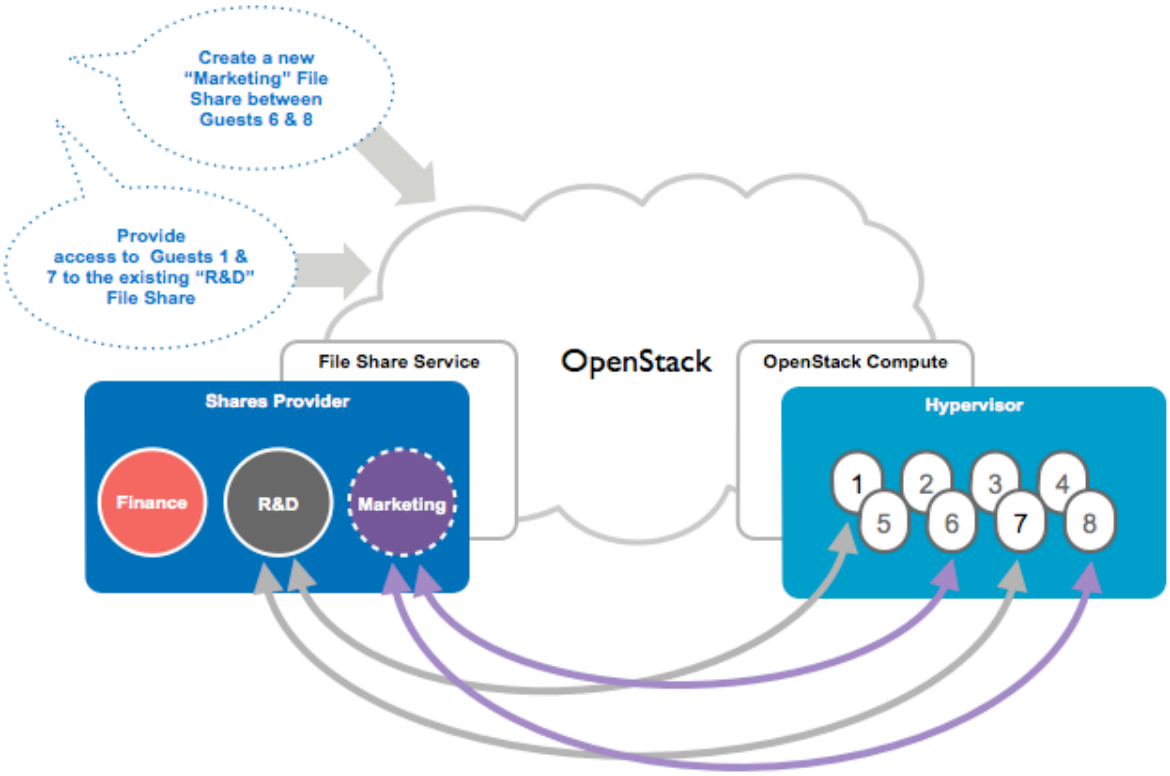


Fig. 10: The diagram shows a basic overview of the manila service (Shared Filesystems service).

Physical host interfaces

In a typical production environment, physical network interfaces are combined in bonded pairs for better redundancy and throughput. Avoid using two ports on the same multiport network card for the same bonded interface, because a network card failure affects both of the physical network interfaces used by the bond. Single (bonded) interfaces are also a supported configuration, but will require the use of VLAN subinterfaces.

Linux Bridges/Switches

The combination of containers and flexible deployment options requires implementation of advanced Linux networking features, such as bridges, switches, and namespaces.

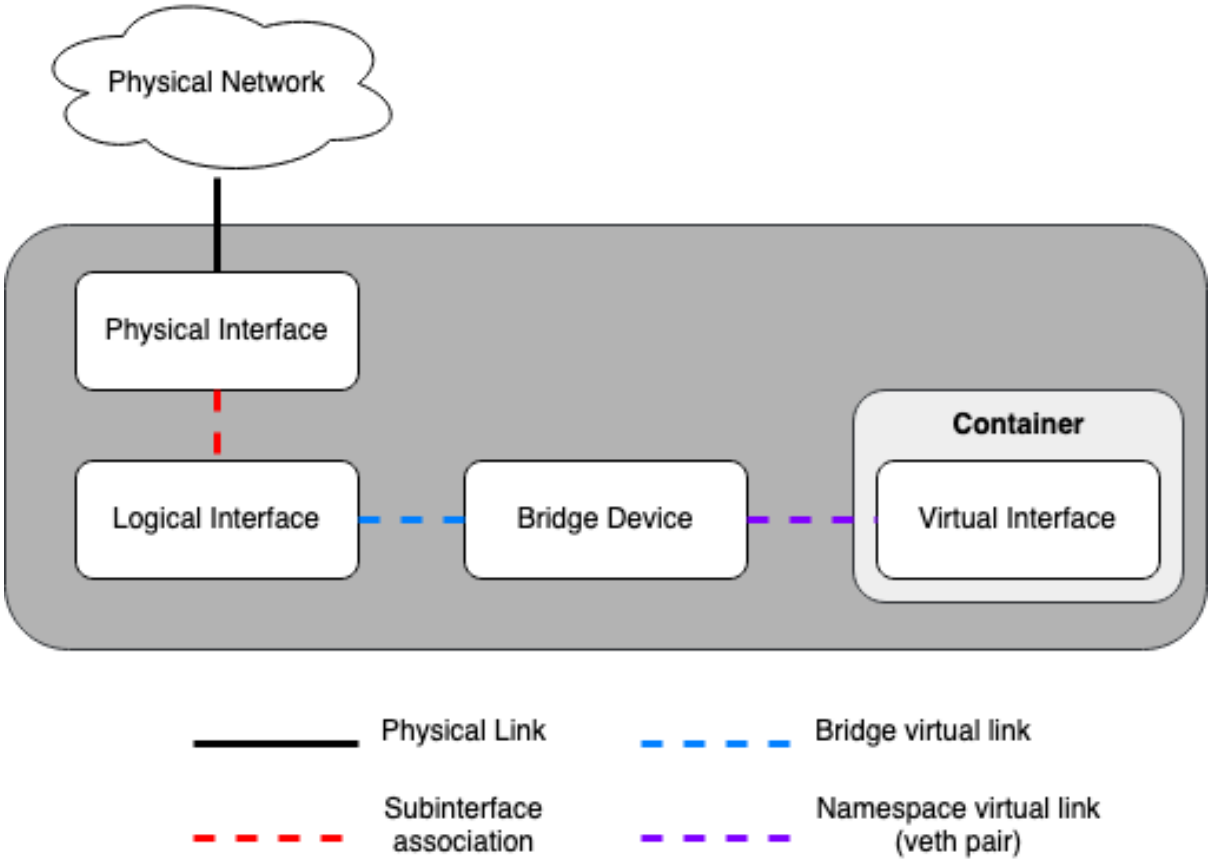
- Bridges provide layer 2 connectivity (similar to physical switches) among physical, logical, and virtual network interfaces within a host. After a bridge/switch is created, the network interfaces are virtually plugged in to it.

  OpenStack-Ansible uses Linux bridges for control plane connections to LXC containers, and can use Linux bridges or Open vSwitch-based bridges for data plane connections that connect virtual machine instances to the physical network infrastructure.

- Network namespaces provide logically separate layer 3 environments (similar to VRFs) within a host. Namespaces use virtual interfaces to connect with other namespaces, including the host namespace. These interfaces, often called `veth` pairs, are virtually plugged in between namespaces similar to patch cables connecting physical devices such as switches and routers.

  Each container has a namespace that connects to the host namespace with one or more `veth` pairs. Unless specified, the system generates random names for `veth` pairs.

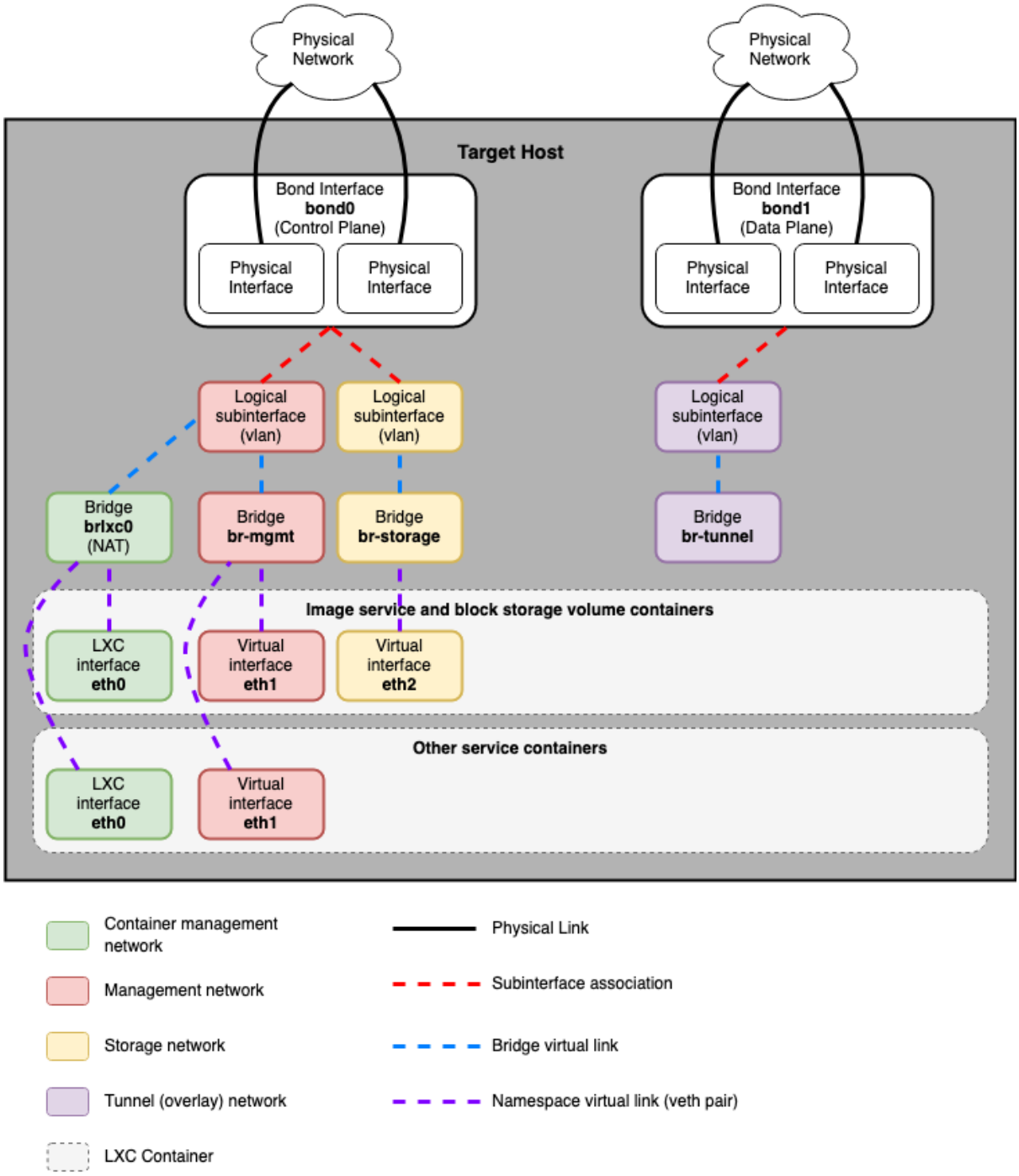
The following image demonstrates how the container network interfaces are connected to the hosts bridges and physical network interfaces:



Network diagrams

Hosts with services running in containers

The following diagram shows how all of the interfaces and bridges interconnect to provide network connectivity to the OpenStack deployment:



The bridge `lxcbr0` is configured automatically and provides connectivity for the containers (via eth0) to the outside world, thanks to dnsmasq (dhcp/dns) + NAT.

Note

If you require additional network configuration for your container interfaces (like changing the routes on eth1 for routes on the management network), please adapt your `openstack_user_config.yml` file. See *openstack_user_config settings reference* for more details.

Neutron traffic

Common reference drivers, including ML2/OVS, and ML2/OVN, and their respective agents, are responsible for managing the virtual networking infrastructure on each node. OpenStack-Ansible refers to Neutron traffic as data plane traffic, and can consist of flat, VLAN, or overlay technologies such as VXLAN and Geneve.

Neutron agents can be deployed across a variety of hosts, but are typically limited to dedicated network hosts or infrastructure hosts (controller nodes). Neutron agents are deployed on metal and not within an LXC container. Neutron typically requires the operator to define provider bridge mappings, which map a provider network name to a physical interface. These provider bridge mappings provide flexibility and abstract physical interface names when creating provider networks.

Open vSwitch/OVN Example:

```
bridge_mappings = physnet1:br-ex
```

OpenStack-Ansible provides two overrides when defining provider networks that can be used for creating the mappings and in some cases, connecting the physical interfaces to provider bridges:

- `host_bind_override`
- `network_interface`

The `host_bind_override` key is used to replace an LXC-related interface name with a physical interface name when a component is deployed on bare metal hosts. It will be used to populate `network_mappings` for Neutron.

The `network_interface` override is used for Open vSwitch and OVN-based deployments, and requires a physical interface name which will be connected to the provider bridge (ie. br-ex) for flat and vlan-based provider and project network traffic.

Note

Previous versions of OpenStack-Ansible utilized a bridge named `br-vlan` for flat and vlan-based provider and project network traffic. The `br-vlan` bridge is a leftover of containerized Neutron agents and is no longer useful or recommended.

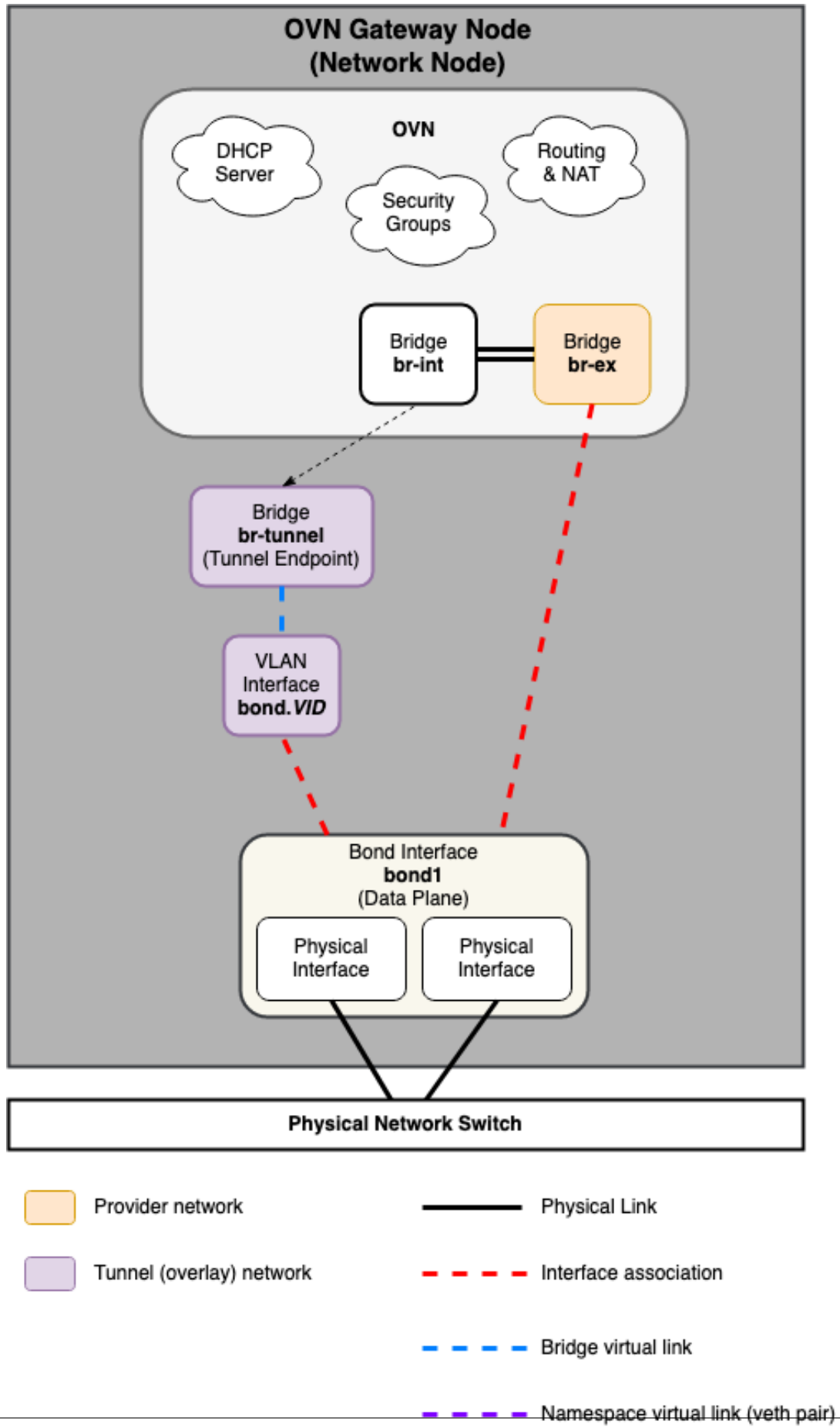
The following diagrams reflect the differences in the virtual network layout for supported network architectures.

Open Virtual Network (OVN)

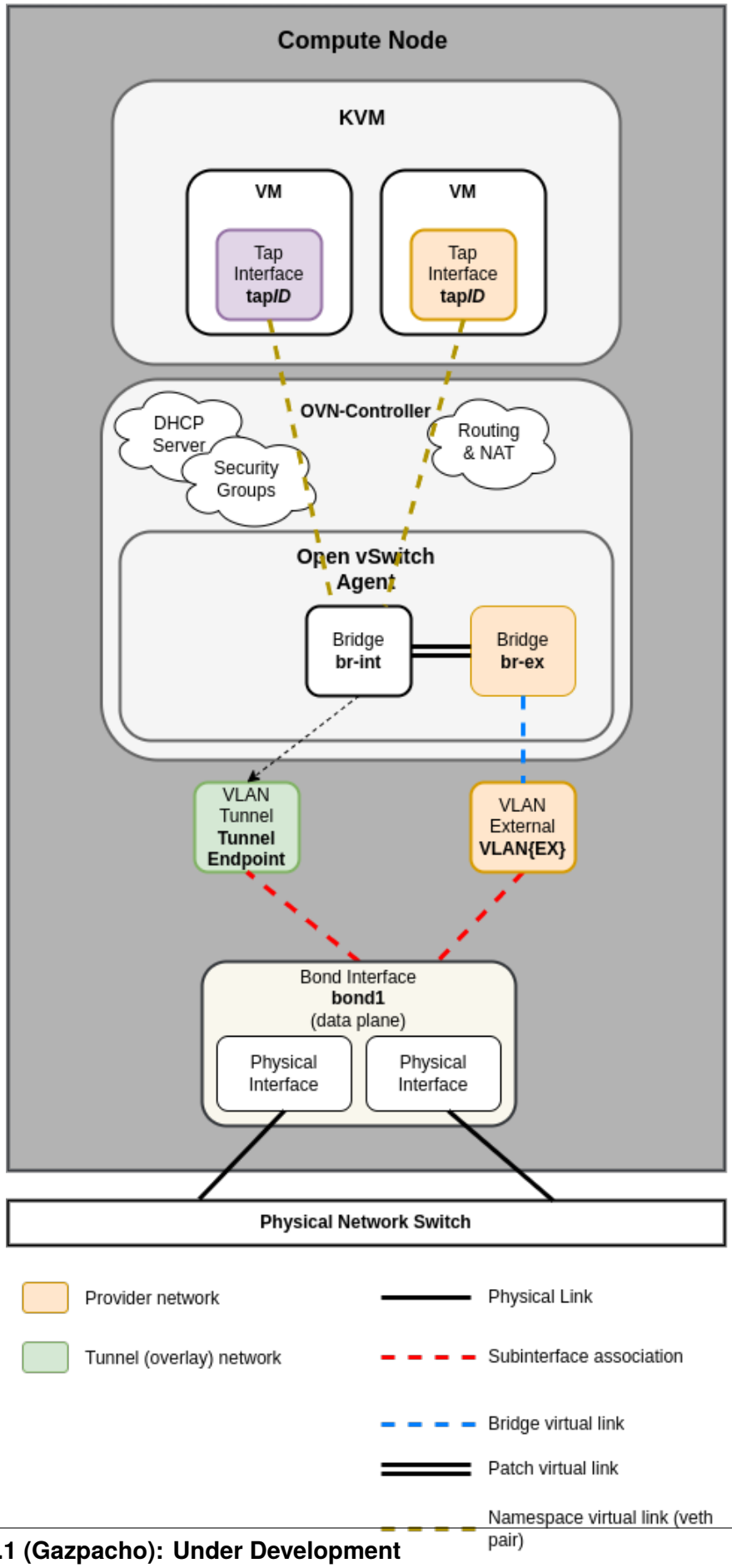
Note

The ML2/OVN mechanism driver is deployed by default as of the Zed release of OpenStack-Ansible.

Networking Node

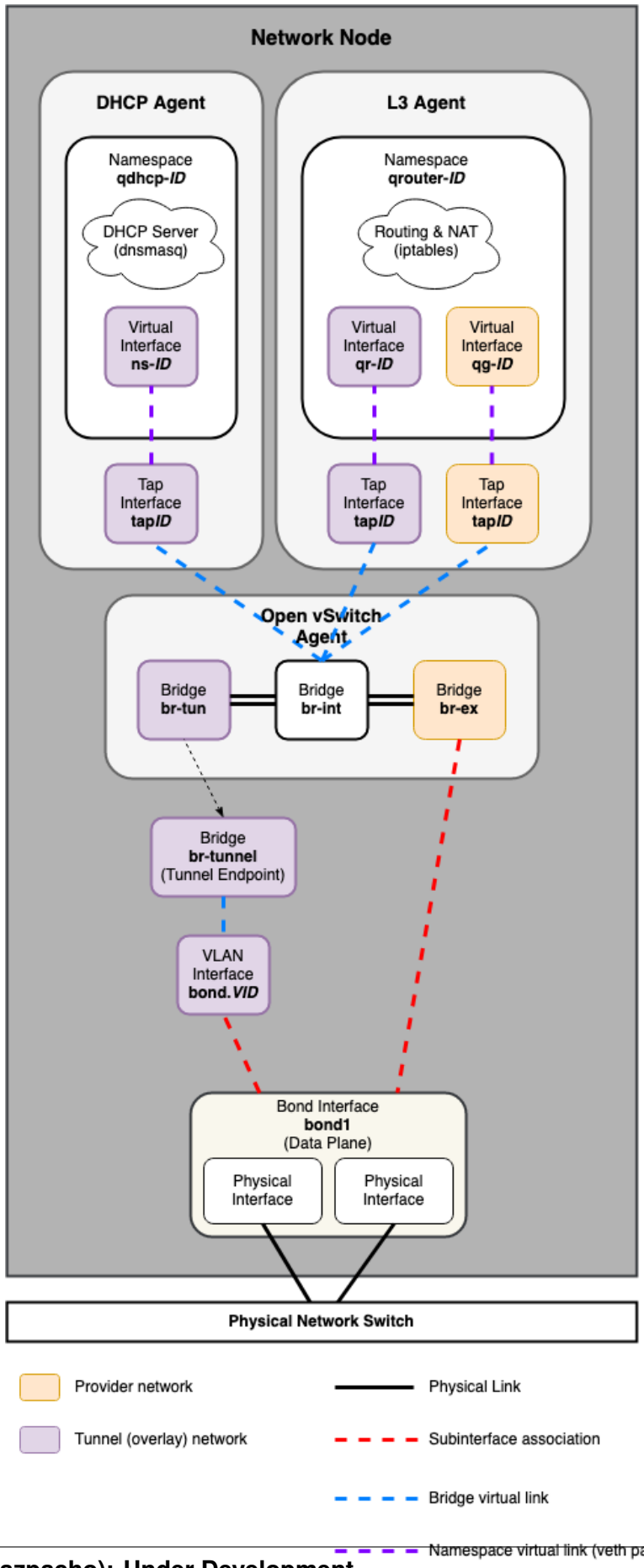


Compute Node

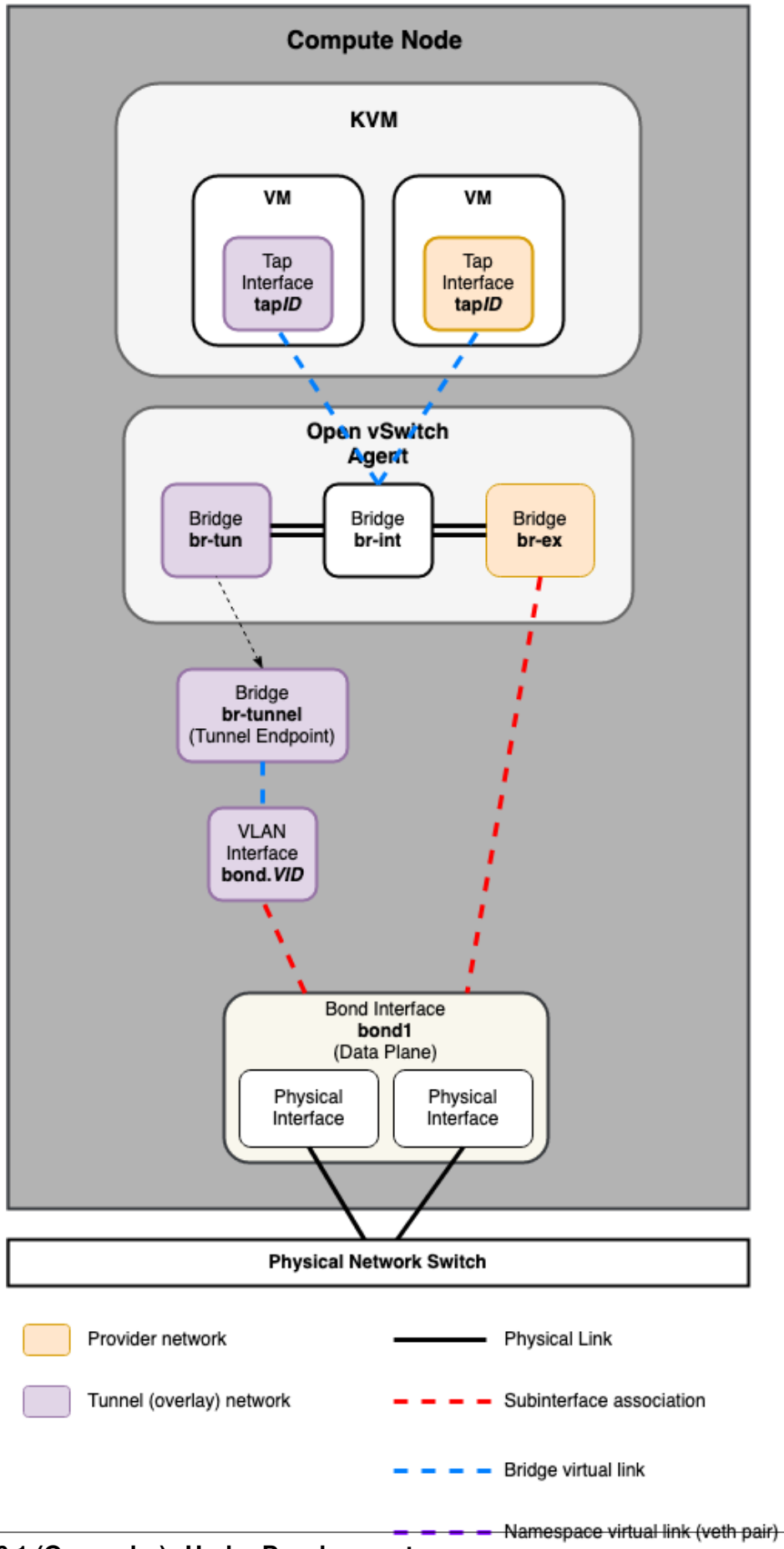


Open vSwitch (OVS)

Networking Node



Compute Node



Metal networking

OpenStack-Ansible supports deploying OpenStack and related services on metal as well as inside LXC containers. Python virtual environments (venvs) provide OpenStack service and Python library segregation, while other services such as Galera and RabbitMQ are co-located on the host. All services in this deployment model share the *same* IP address.

This appendix describes how the interfaces are connected and how traffic flows.

For more information about how the OpenStack Networking service (Neutron) uses the interfaces for instance traffic, please see the OpenStack Networking Guide.

For details on the configuration of networking for your environment, please have a look at *openstack_user_config settings reference*.

Physical host interfaces

In a typical production environment, physical network interfaces are combined in bonded pairs for better redundancy and throughput. Avoid using two ports on the same multiport network card for the same bonded interface, because a network card failure affects both of the physical network interfaces used by the bond. Multiple bonded interfaces (ie. bond0, bond1) can be used to segregate traffic, if desired. Single (bonded) interfaces are also a supported configuration, but will require the use of VLAN subinterfaces.

Linux Bridges/Switches

The combination of containers and flexible deployment options requires implementation of advanced Linux networking features, such as bridges, switches, and namespaces.

- Bridges provide layer 2 connectivity (similar to switches) among physical, logical, and virtual network interfaces within a host. After a bridge/switch is created, the network interfaces are virtually plugged in to it.

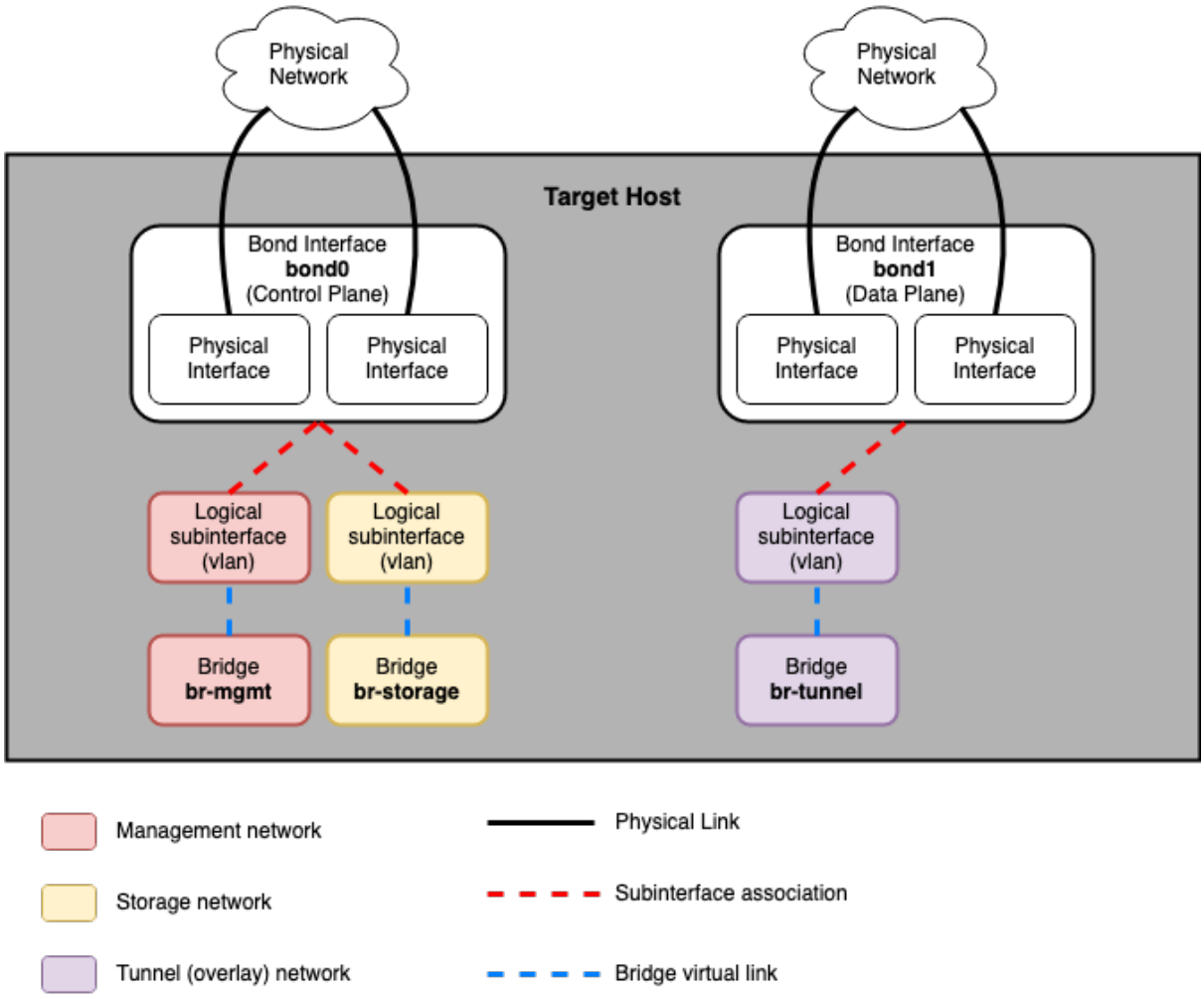
  OpenStack-Ansible uses Linux bridges for control plane connections to LXC containers, and can use Linux bridges or Open vSwitch-based bridges for data plane connections that connect virtual machine instances to the physical network infrastructure.

- Network namespaces provide logically separate layer 3 environments (similar to VRFs) within a host. Namespaces use virtual interfaces to connect with other namespaces, including the host namespace. These interfaces, often called `veth` pairs, are virtually plugged in between namespaces similar to patch cables connecting physical devices such as switches and routers.

Network diagrams

Hosts with services running on metal

The following diagram shows how all of the interfaces and bridges interconnect to provide network connectivity to the OpenStack deployment:



Neutron traffic

Common reference drivers, including ML2/OVS, and ML2/OVN, and their respective agents, are responsible for managing the virtual networking infrastructure on each node. OpenStack-Ansible refers to Neutron traffic as data plane traffic, and can consist of flat, VLAN, or overlay technologies such as VXLAN and Geneve.

Neutron agents can be deployed across a variety of hosts, but are typically limited to dedicated network hosts or infrastructure hosts (controller nodes). Neutron agents are deployed on metal and not within an LXC container. Neutron typically requires the operator to define provider bridge mappings, which map a provider network name to a physical interface. These provider bridge mappings provide flexibility and abstract physical interface names when creating provider networks.

Open vSwitch/OVN Example:

```
bridge_mappings = physnet1:br-ex
```

OpenStack-Ansible provides two overrides when defining provider networks that can be used for creating the mappings and in some cases, connecting the physical interfaces to provider bridges:

- `host_bind_override`
- `network_interface`

The `host_bind_override` key is used to replace an LXC-related interface name with a physical

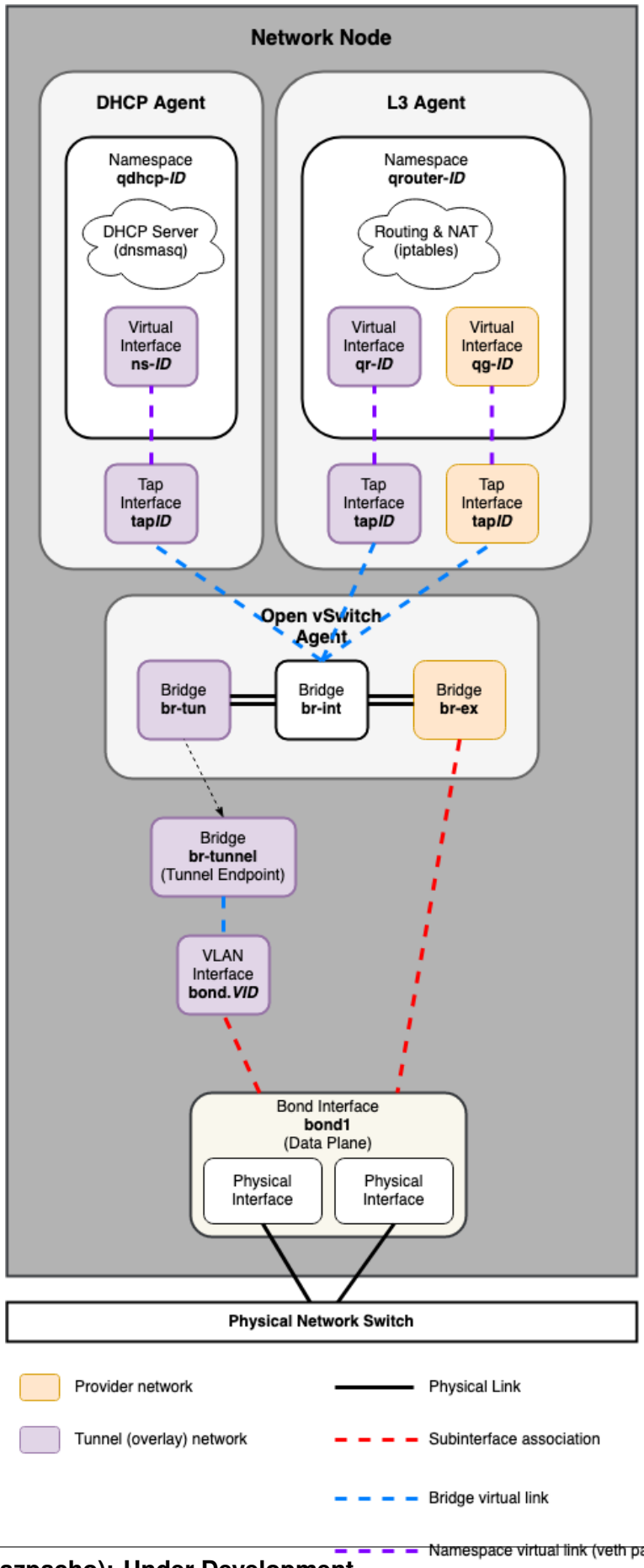
interface name when a component is deployed on bare metal hosts. It will be used to populate `network_mappings` for Neutron.

The `network_interface` override is used for Open vSwitch and OVN-based deployments, and requires a physical interface name which will be connected to the provider bridge (ie. br-ex) for flat and vlan-based provider and project network traffic.

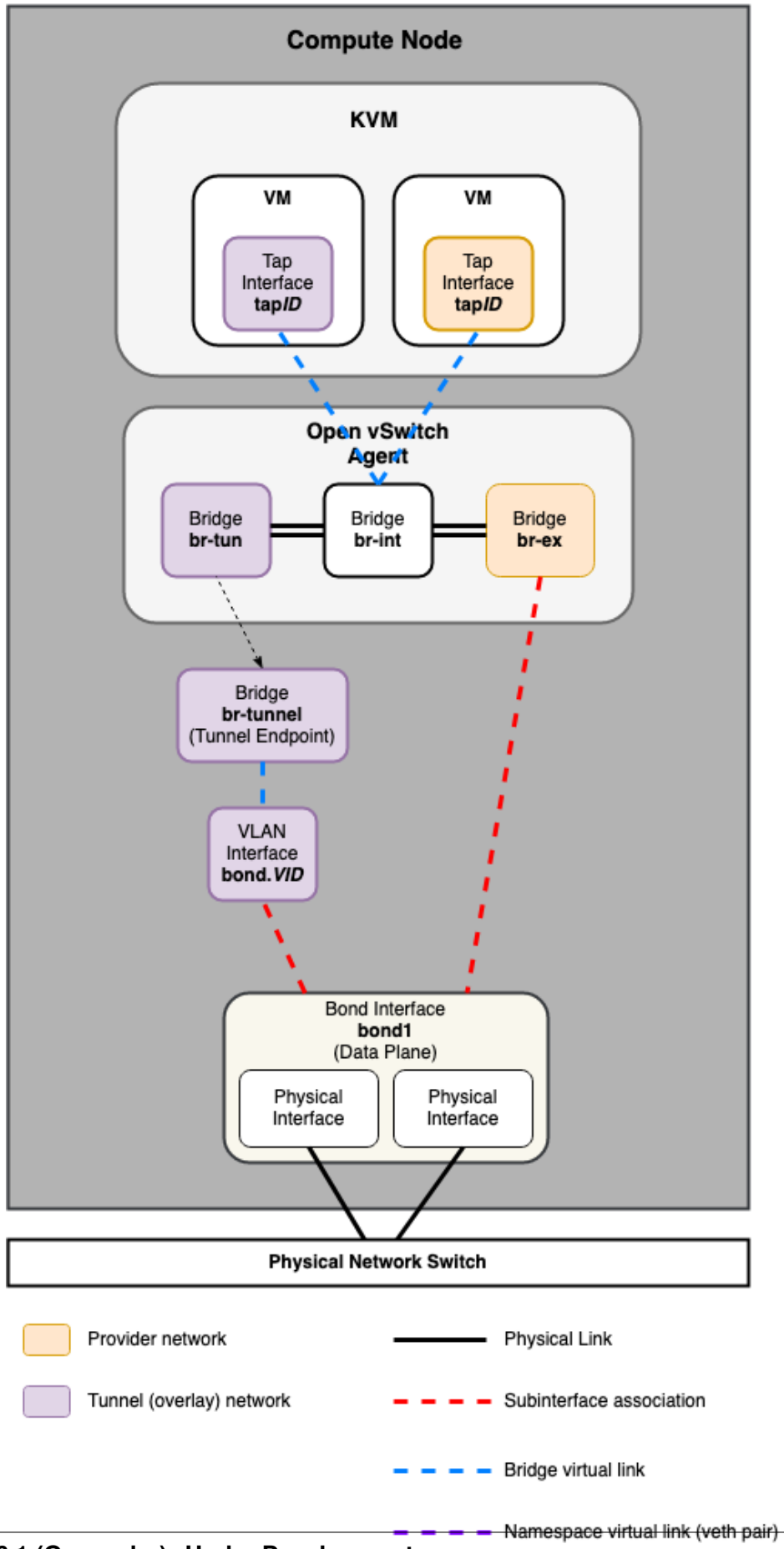
The following diagrams reflect the differences in the virtual network layout for supported network architectures.

Open vSwitch (OVS)

Networking Node



Compute Node

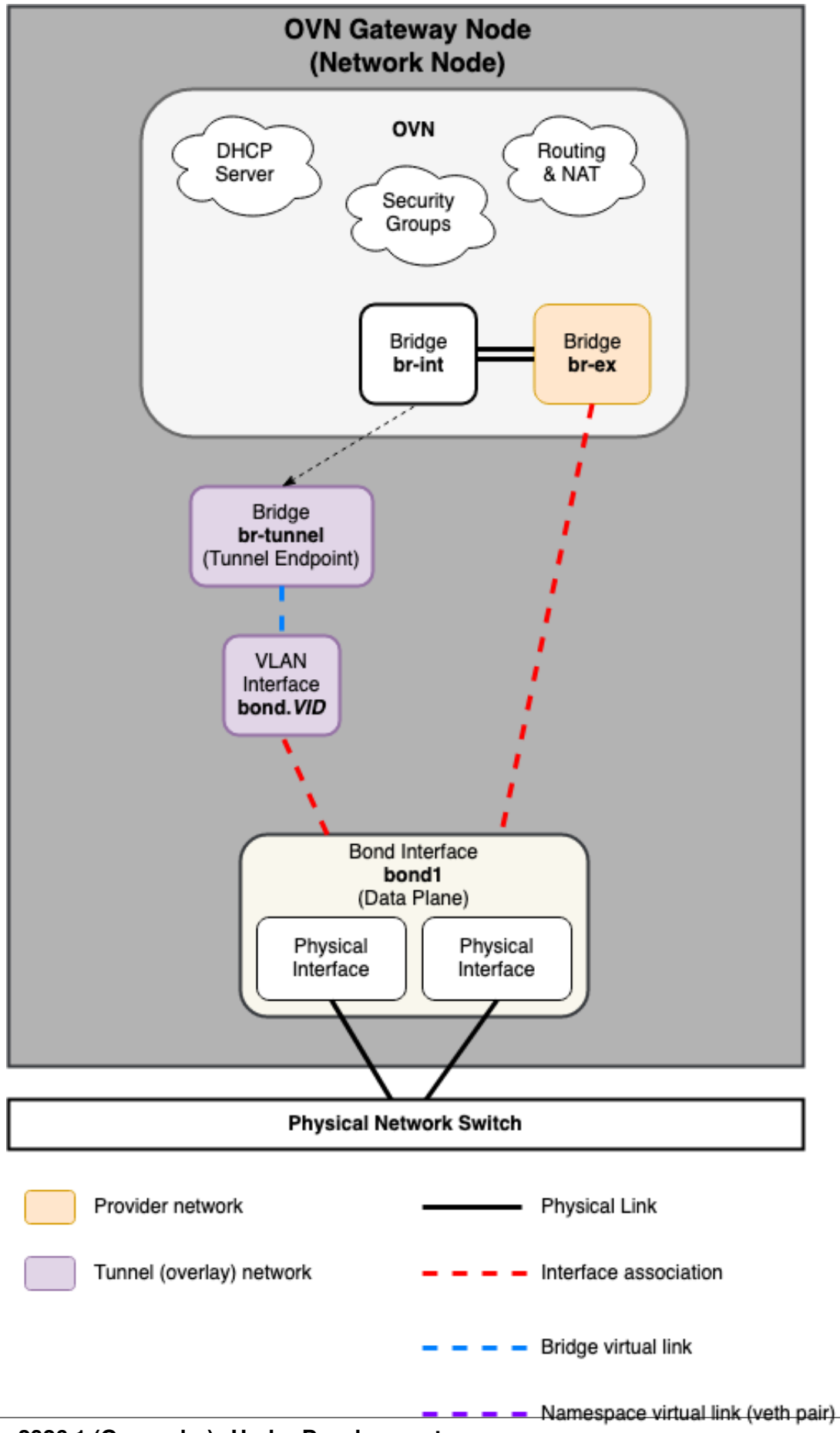


Open Virtual Network (OVN)

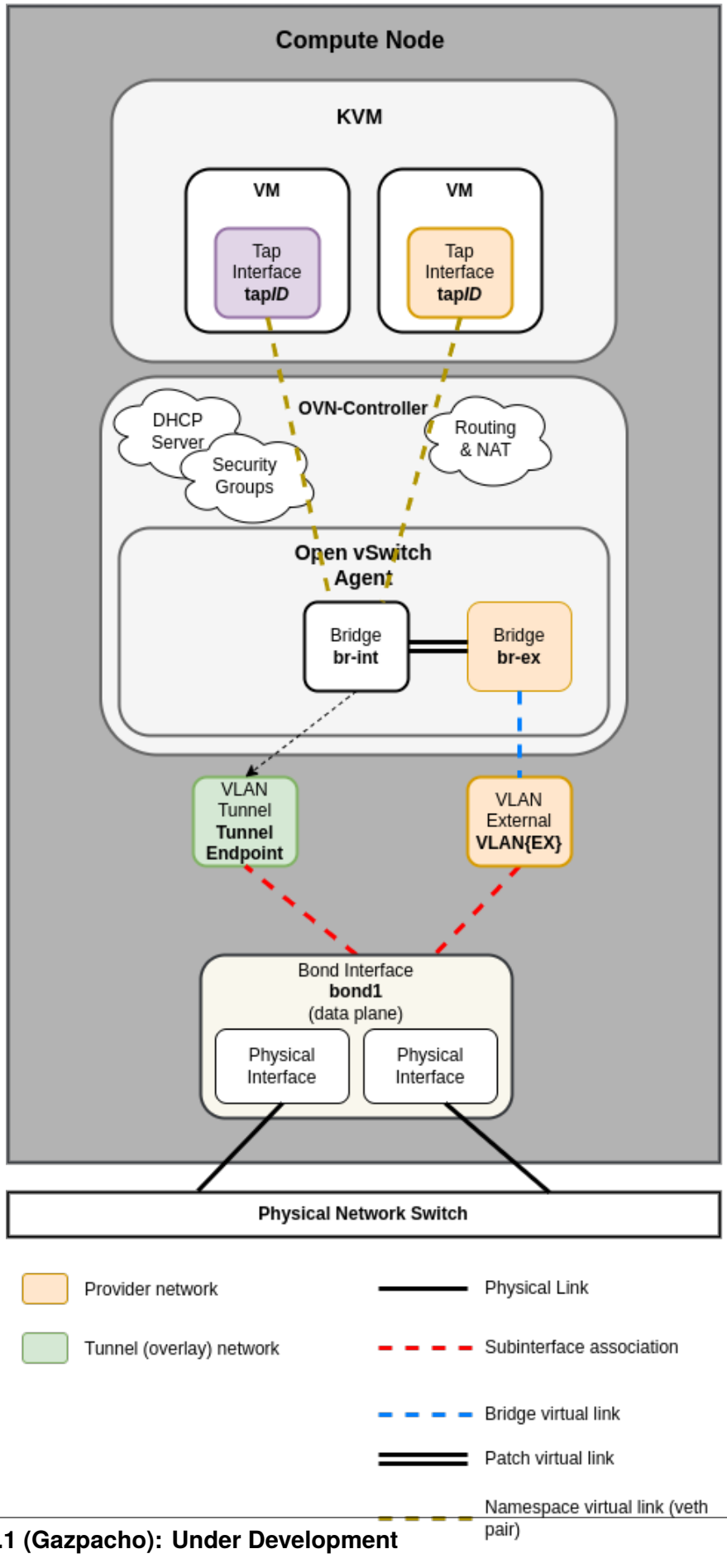
Note

The ML2/OVN mechanism driver is deployed by default as of the Zed release of OpenStack-Ansible.

Networking Node



Compute Node



Command Line Reference

Linux Container commands

The following are some useful commands to manage LXC:

- List containers and summary information such as operational state and network configuration:

```
# lxc-ls --fancy
```

- Show container details including operational state, resource utilization, and `veth` pairs:

```
# lxc-info --name container_name
```

- Start a container:

```
# lxc-start --name container_name
```

- Attach to a container:

```
# lxc-attach --name container_name
```

- Stop a container:

```
# lxc-stop --name container_name
```

Additional Resources

Ansible Resources

- Ansible Documentation
- Configuring Ansible
- Ansible tips and tricks

OpenStack Resources

- OpenStack Documentation
- OpenStack API Documentation

Ceph Resources

- Ceph Documentation

About OpenStack-Ansible

OpenStack-Ansible (OSA) uses the Ansible IT automation engine to deploy an OpenStack environment on

For isolation and ease of maintenance, all OpenStack services are installed by default from source code into python virtual environments.

The services are further isolated via the use of LXC containers, but these are optional and a bare-metal-based installation is also possible.

OpenStack-Ansible Manifesto

All the design considerations (the container architecture, the ability to override any code, the network considerations, etc.) of this project are listed in our *Architecture* reference.

Why choose OpenStack-Ansible?

- Supports the major Linux distributions
- Offers automation for upgrades between major OpenStack releases.
- Uses OpenStack defaults for each of the project roles, and provides extra wiring and optimized configuration when combining projects together.
- Does not implement its own DSL, and uses wherever possible Ansible directly. All the experience acquired using Ansible can be used in OpenStack-Ansible, and the other way around.
- You like to use reliable, proven technology. We try to run OpenStack with a minimum amount of packages that are not provided by distributions or the OpenStack community. Less dependencies and distribution tested software make the project more reliable.
- You want to be able to select how to deploy on your hardware: deploy partially on metal, fully on metal, or fully in machine containers.

When not to choose OpenStack-Ansible?

- If your company is already invested in other configuration management system (Puppet) and does not want to use Ansible we recommend building on your existing knowledge and experimenting with a different OpenStack deployment project.
- You want to deploy OpenStack with 100% application containers. We currently support LXC containers, if you want to go 100% Docker, there are other projects in the OpenStack community that can help you.
- You want to deploy OpenStack services from distribution packages (deb or rpm). Whilst there is some support for this, coverage of the services is incomplete and a lot of operator flexibility is lost when using this approach.

1.1.4 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 OpenStack-Ansible.

Communication

IRC channel

Warning

The OpenStack Community moved the IRC network from Freenode to OFTC on May 31, 2021. All the current IRC channels used in The OpenStack community are registered in OFTC network too.

The OpenStack-Ansible community communicates in the #openstack-ansible IRC channel hosted on OFTC. This channel is logged, and its logs are published on https://meetings.opendev.org/irclogs/%23openstack-ansible/

Weekly meetings are held in our IRC channel. The schedule and logs can be found on https://meetings.opendev.org/%23OpenStack_Ansible_Deployment_Meeting The agenda for the next meeting can be found on our Meetings wiki page.

Mailing lists

Members of the OpenStack-Ansible community should monitor the **OpenStack-discuss** mailing lists.

All our communications should be prefixed with **[openstack-ansible]**.

Contacting the Core Team

All of our core team is available through IRC and present in #openstack-ansible channel on OFTC. The list of the current members of the OpenStack-Ansible Team might be found on gerrit.

New Feature Planning

If you would like to contribute towards a role to introduce an OpenStack or infrastructure service, or to improve an existing role, the OpenStack-Ansible project would welcome that contribution and your assistance in maintaining it.

Please look through Contributor Guidelines page for more information about the process.

Task Tracking

We track our tasks in Launchpad

> https://bugs.launchpad.net/openstack-ansible

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.

Also you may find more detailed information about how to work with bugs on the page Bug Handling.

Getting Your Patch Merged

Any new code will be reviewed before merging into our repositories and requires at least 2 approvals from our Core team.

We follow openstack guidelines for the code reviewing process.

Please be aware that any patch can be refused by the community if they dont match the *General Guidelines for Submitting Code*.

Project Team Lead Duties

All common PTL duties are enumerated in the PTL guide.

All Core reviewer duties are described on the page Core Reviewers.

1.1.5 Developer Documentation

In this section, you will find documentation relevant to developing OpenStack-Ansible.

For information on how to deploy your OpenStack-Ansible cloud, refer to the Deployment Guide for step-by-step instructions on how to deploy the OpenStack packages and dependencies on your cloud using OpenStack-Ansible.

For user guides, see the *User Guide*.

For information on how to manage and operate OpenStack-Ansible, see the *Operations Guide*.

For in-depth technical information, see the *OpenStack-Ansible Reference*.

Contents:

Project Onboarding

This document should help you understand how to contribute to OpenStack-Ansible.

Project repositories

The OpenStack-Ansible project has different kinds of git repositories, each of them with specific use cases, and different sets of practices.

Repository type or name	Code location	Repository purpose
OpenStack-Ansible Also called *integrated repository*	• https://opendev.org/openstack/openstack-ansible	Our main repository, used by deployers. Uses the other repositories.
The **OpenStack-Ansible roles** repositories	• https://opendev.org/openstack/openstack-ansible-os_nova • https://opendev.org/openstack/openstack-ansible-os_glance • https://opendev.org/openstack/ansible-role-systemd_mount • https://opendev.org/openstack/ansible-config_template • https://opendev.org/openstack/ansible-hardening •	Each role is in charge of deploying **exactly one** component of an OpenStack-Ansible deployment.
The **specs** repository	• https://opendev.org/openstack/openstack-ansible-specs	This repository contains all the information concerning large bodies of work done in OpenStack-Ansible, split by cycle.
The **ops** repository	• https://opendev.org/openstack/openstack-ansible-ops	This repository is an incubator for new projects, each project solving a particular operational problem. Each project has its own folder in this repository.
External repositories	• https://github.com/ceph/ceph-ansible • https://github.com/logan2211/ansible-resolvconf • https://github.com/willshersystems/ansible-sshd • https://github.com/evrardjp/ansible-keepalived •	OpenStack-Ansible is not reinventing the wheel, and tries to reuse as much as possible existing roles. A bugfix for one of those repositories must be handled to these repositories maintainers.

How to contribute on code or issues

- For contributing code and documentation, you must follow the OpenStack practices. Nothing special is required for OpenStack-Ansible.

  See also the OpenStack developers getting started page. and our *contributor guidelines* before hacking.

- For helping on or submitting bugs, you must have an account on ubuntu Launchpad. All our repositories share the same Launchpad project.

  Please check our *bug report* and *bug triage* processes.

  Easy to fix bugs are marked with the tag *low hanging fruit*, and should be the target of first time contributors.

- For sharing your user experience, stories, and helping other users, please join us in our *IRC channel*.
- The OpenStack-Ansible project has recurring tasks that need attention, like releasing, or other code duties. See our page *Periodic work*.

Community communication channels

IRC channel

> **Warning**
>
> The OpenStack Community moved the IRC network from Freenode to OFTC on May 31, 2021. All the current IRC channels used in The OpenStack community are registered in OFTC network too.

The OpenStack-Ansible community communicates a lot through IRC, in the #openstack-ansible channel, on OFTC. This channel is logged, and its logs are published on https://meetings.opendev.org/irclogs/%23openstack-ansible/

Weekly meetings are held in our IRC channel. The schedule and logs can be found on https://meetings.opendev.org/%23OpenStack_Ansible_Deployment_Meeting

Next meeting agenda can be found on our Meetings wiki page.

Mailing lists

A member of the OpenStack-Ansible community should monitor the **OpenStack-discuss** mailing lists.

All our communications should be prefixed with **[openstack-ansible]**.

OpenStack-Ansible Bug Handling

Bug Reporting

Bugs should be filed on the OpenStack-Ansible Launchpad project.

When submitting a bug, or working on a bug, please ensure the following criteria are met:

- The description clearly states or describes the original problem or root cause of the problem.
- The description clearly states the expected outcome of the user action.
- Include historical information on how the problem was identified.

- Any relevant logs or user configuration are included, either directly or through a pastebin.
- If the issue is a bug that needs fixing in a branch other than master, please note the associated branch within the launchpad issue.
- The provided information should be totally self-contained. External access to web services/sites should not be needed.
- Steps to reproduce the problem if possible.

Bug Tags

If the reported needs fixing in a branch in addition to master, add a <release>-backport-potential tag (e.g. `liberty-backport-potential`). There are predefined tags that will auto-complete.

Status

Please leave the **status** of an issue alone until someone confirms it or a member of the bugs team triages it. While waiting for the issue to be confirmed or triaged the status should remain as **New**.

Importance

Should only be touched if it is a Blocker/Gating issue. If it is, please set to **High**, and only use **Critical** if you have found a bug that can take down whole infrastructures. Once the importance has been changed the status should be changed to **Triaged** by someone other than the bug creator.

The triaging process is explained here below.

Bug triage

What is a bug triage

Bug triage is a process where tracker issues are screened and prioritised. Triage should help ensure we appropriately manage all reported issues - bugs as well as improvements and feature requests. (Source: Moodle bug triage)

Reported bugs need confirmation, prioritization, and ensure they do not go stale. If you care about OpenStack stability but are not wanting to actively develop the roles and playbooks used within the OpenStack-Ansible project, consider contributing in the area of bug triage.

Please reference the *Project Team Guide bugs reference_* for information about bug status/importance and the life cycle of a bug.

Bug triage meeting duties

If the bug description is incomplete, or the report is lacking the information necessary to reproduce the issue, ask the reporter to provide missing information, and set the bug status to *Incomplete*

If the bug report contains enough information and you can reproduce it (or it looks valid), then you should set its status to *Confirmed*.

If the bug has security implications, set the security flag (under This report is public on the top right)

If the bug affects a specific area covered by an official tag, you should set the tag. For example, if the bug is likely to be quite easy to solve, add the *low-hanging-fruit* tag.

The bug triage meeting is probably a good time for people with bug supervisors rights to also prioritize bugs per importance (on top of classifying them on status).

Bug skimming duty

To help triaging bugs, one person of the bug team can be on bug skimming duty.

Q

What is the goal of the bug skimming duty?

A

Bug skimming duty reduces the amount of work other developers have to spend to do a proper root cause analysis (and later fix) of bug reports. For this, close the obviously invalid bug reports, confirm the obviously valid bug reports, ask questions if things are unclear.

Q

Do I need to prove that a bug report is valid/invalid before I can set it to *Confirmed*/*Invalid* ?

A

No. Sometimes it is not even possible because you do not have the resources. Looking at the code and tests often enables you to make an educated guess. Citing your sources in a comment helps the discussion.

Q

What is the best status to close a bug report if its issue cannot be reproduced?

A

Definitively *Invalid*. The status *Incomplete* is an open state and means that more information is necessary.

Q

How do I handle open bug reports which are Incomplete for too long?

A

If it is in this state for more than 30 days and no answers to the open questions are given, close it with Wont Fix.

Q

How do I handle dependencies to other bugs or TBD features in other projects? For example, I can fix a bug in OpenStack-Ansible but I need that a feature in Compute (nova) gets implemented before.

A

Leave a comment in the OpenStack-Ansible bug report which explains this dependency and leave a link to the blueprint or bug report of the other project you depend on.

Q

Do I have to double-check bug reports which are New and have an assignee?

A

Usually not. This bug report has an inconsistent state though. If a bug report has an assignee, it should be In Progress and have an importance set.

Bug skimming duty weekly checklist

- Prioritize or reprioritize OpenStack-Ansible confirmed bugs.
- Move year old wishlist bugs to Opinion/Wishlist to remove clutter. You can use the following message:

  > This wishlist bug has been open a year without any activity. I am moving this to Opinion / Wishlist. This is an easily-obtainable queue of older requests. This bug can be reopened (set back to New) if someone decides to work on this.

- Move bugs that can not be reproduced to an invalid state if they are unmodified for more than a month.
- Send an email to the openstack-discuss list with the list of bugs to triage during the week. A new bug marked as *Critical* or *High* must be treated in priority.

Contributor Guidelines

Before submitting code

Before jumping ahead and working on code, a series of steps should be taken:

- Is there a bug for it? Can your track if someone else has seen the same bug?
- Are you sure nobody is working on this problem at the moment? Could there be a review pending fixing the same issue?
- Have you checked if your issue/feature request hasnt been solved in another branch?

If youre willing to submit code, please remember the following rules:

- All code should match our *General Guidelines for Submitting Code*.
- All code requires to go through our *Review process*.
- Documentation should be provided with the code directly. See also *Documentation and Release Note Guidelines*.
- Fixing bugs and increasing test coverage have priority to new features. See also the section *Working on bug fixes*.
- New features are following a process, explained in the section *Working on new features*. New features are less likely to be *backported* to previous branches.

Review process

Any new code will be reviewed before merging into our repositories.

We follow openstack guidelines for the code reviewing process.

Please be aware that any patch can be refused by the community if they dont match the *General Guidelines for Submitting Code*.

Working on bug fixes

Any bug fix should have, in its commit message:

> Closes-Bug: #bugnumber

or

Related-Bug: #bugnumber

where #bugnumber refers to a Launchpad issue.

See also the working on bugs section of the openstack documentation.

Working on new features

If you would like to contribute towards a role to introduce an OpenStack or infrastructure service, or to improve an existing role, the OpenStack-Ansible project would welcome that contribution and your assistance in maintaining it.

Here are a few rules to get started:

- All large feature additions/deletions should be accompanied by a blueprint/spec. e.g. adding additional active agents to neutron, developing a new service role, etc See also *Submitting a specification*.
- Before creating blueprint/spec an associated Wishlist Bug can be raised on launchpad. This issue will be triaged and a determination will be made on how large the change is and whether or not the change warrants a blueprint/spec. Both features and bug fixes may require the creation of a blueprint/spec. This requirement will be voted on by core reviewers and will be based on the size and impact of the change.
- All blueprints/specs should be voted on and approved by core reviewers before any associated code will be merged. For more information on blueprints/specs please review the OpenStack documentation regarding Working on Specifications and Blueprints and our own *Submitting a specification*.
- Once the blueprint work is completed the author(s) can request a backport of the blueprint work into a stable branch. Each backport will be evaluated on a case by case basis with cautious consideration based on how the backport affects any existing deployments. See the *Backporting* section for more information.
- Any new OpenStack services implemented which have Tempest tests available must be implemented along with suitable functional tests enabled as part of the feature development in order to ensure that any changes to the code base do not break the service functionality.
- Feature additions must include documentation which provides reference to OpenStack documentation about what the feature is and how it works. The documentation should then describe how it is implemented in OpenStack-Ansible and what configuration options there are. See also the *Documentation and Release Note Guidelines* section.

Example process to develop a new role

Here are the steps to write the role:

1. You can review roles which may be currently in development by checking our specs repository and unmerged specs on review.openstack.org. If you do not find a spec for the role, propose a blueprint/spec. See also *Submitting a specification*.
2. Create a source repository (e.g. on Github) to start your work on the Role.
3. Generate the reference directory structure for an Ansible role which is the necessary subset of the documented Best Practice. You might use Ansible Galaxy tools to do this for you (e.g. `ansible-galaxy init`). You may additionally want to include directories such as `docs` and `examples` and `tests` for your role.

4. Generate a meta/main.yml right away. This file is important to Ansible to ensure your dependent roles are installed and available and provides others with the information they will need to understand the purpose of your role.
5. Develop task files for each of the install stages in turn, creating any handlers and templates as needed. Ensure that you notify handlers after any task which impacts the way the service would run (such as configuration file modifications). Also take care that file ownership and permissions are appropriate.

   Hint

   Fill in variable defaults, libraries, and prerequisites as you discover a need for them. You can also develop documentation for your role at the same time.

6. Add tests to the role. See also our *Testing* page.
7. Ensuring the role matches OpenStack-Ansibles latest standards. See also our *Code rules* page.
8. Ensure the role converges:
   - Implement **developer_mode** to build from a git source into a Python virtual environment.
   - Deploy the applicable configuration files in the right places.
   - Ensure that the service starts.

   The convergence may involve consuming other OpenStack-Ansible roles (For example: **galera_server, galera_client, rabbitmq_server**) in order to ensure that the appropriate infrastructure is in place. Re-using existing roles in OpenStack-Ansible or Ansible Galaxy is strongly encouraged.
9. Once the initial convergence is working and the services are running, the role development should focus on implementing some level of functional testing. See also *Improve testing with tempest*.
10. Test the role on a new machine, using our provided scripts.
11. Submit your role for review.
12. If required, ask the OpenStack-Ansible PTL to import the github role into the openstack-ansible namespace (This can only be done early in the development cycle, and may be postponed to next cycle).
13. If necessary, work on the integration within the openstack-ansible integrated repository, and deploy the role on an AIO. See also *Testing a new role with an AIO*.

Example process for adding a feature to an existing role

1. Search for in the OpenStack-Ansible Launchpad project for the feature request.
2. If no Wishlist item exist in Launchpad for your feature, create a bug for it. Dont hesitate to ask if a spec is required in the bug.
3. The *Bug triage* will classify if this new feature requires a spec or not.
4. Work on the role files, following our *Code rules*.
5. Add an extra role test scenario, to ensure your code path is tested and working.
6. Test your new scenario with a new machine. See also the *Running tests locally* page.

7. Submit your code for review, with its necessary documentation and release notes.

Example process to incubate a new ops project

A new project in openstack-ansible-ops can be started at any time, with no constraint like writing a specification, or creating a bug.

Instead, the new code has to be isolated on a separate folder of the openstack-ansible-ops repo.

Backporting

- Backporting is defined as the act of reproducing a change from another branch. Unclean/squashed/modified cherry-picks and complete reimplementations are OK.
- Backporting is often done by using the same code (via cherry picking), but this is not always the case. This method is preferred when the cherry-pick provides a complete solution for the targeted problem.
- When cherry-picking a commit from one branch to another the commit message should be amended with any files that may have been in conflict while performing the cherry-pick operation. Additionally, cherry-pick commit messages should contain the original commit *SHA* near the bottom of the new commit message. This can be done with `cherry-pick -x`. Heres more information on Submitting a change to a branch for review.
- Every backport commit must still only solve one problem, as per the guidelines in *General Guidelines for Submitting Code*.
- If a backport is a squashed set of cherry-picked commits, the original SHAs should be referenced in the commit message and the reason for squashing the commits should be clearly explained.
- When a cherry-pick is modified in any way, the changes made and the reasons for them must be explicitly expressed in the commit message.
- Refactoring work must not be backported to a released branch.
- Backport reviews should be done with due consideration to the effect of the patch on any existing environment deployed by OpenStack-Ansible. The general OpenStack Guidelines for stable branches can be used as a reference.

Code rules

General Guidelines for Submitting Code

- Write good commit messages. We follow the OpenStack Git Commit Good Practice guide. if you have any questions regarding how to write good commit messages please review the upstream OpenStack documentation.
- Changes to the project should be submitted for review via the Gerrit tool, following the workflow documented here.
- Pull requests submitted through GitHub will be ignored and closed without regard.
- Patches should be focused on solving one problem at a time. If the review is overly complex or generally large the initial commit will receive a **-2** and the contributor will be asked to split the patch up across multiple reviews. In the case of complex feature additions the design and implementation of the feature should be done in such a way that it can be submitted in multiple patches using dependencies. Using dependent changes should always aim to result in a working build throughout the dependency chain. Documentation is available for advanced gerrit usage too.

- All patch sets should adhere to the *Ansible Style Guide* listed here as well as adhere to the Ansible playbooks when possible.
- All changes should be clearly listed in the commit message, with an associated bug id/blueprint along with any extra information where applicable.
- Refactoring work should never include additional rider features. Features that may pertain to something that was re-factored should be raised as an issue and submitted in prior or subsequent patches.
- New features, breaking changes and other patches of note must include a release note generated using the reno tool. Please see the *Documentation and Release Note Guidelines* for more information.
- All patches including code, documentation and release notes should be built and tested locally with the appropriate test suite before submitting for review. See *Development and Testing* for more information.

Documentation and Release Note Guidelines

Documentation is a critical part of ensuring that the deployers of OpenStack-Ansible are appropriately informed about:

- How to use the projects tooling effectively to deploy OpenStack.
- How to implement the right configuration to meet the needs of their specific use-case.
- Changes in the project over time which may affect an existing deployment.

To meet these needs developers must submit *code comments*, documentation (see also the *documentation locations section*) and *release notes* with any code submissions.

All forms of documentation should comply with the guidelines provided in the OpenStack Documentation Contributor Guide, with particular reference to the following sections:

- Writing style
- RST formatting conventions

Code Comments

Code comments for variables should be used to explain the purpose of the variable. This is particularly important for the role defaults file as the file is included verbatim in the roles documentation. Where there is an optional variable, the variable and an explanation of what it is used for should be added to the defaults file.

Code comments for bash/python scripts should give guidance to the purpose of the code. This is important to provide context for reviewers before the patch has merged, and for later modifications to remind the contributors what the purpose was and why it was done that way.

Documentation locations

OpenStack-Ansible has multiple forms of documentation with different intent.

The Deployment Guide intends to help deployers deploy OpenStack-Ansible for the first time.

The User Guide intends to provide user stories on how to do specific things with OpenStack-Ansible.

The Operations Guide provide help on how to manage and operate OpenStack-Ansible.

The in-depth technical information is located in the OpenStack-Ansible Reference.

The role documentation (for example, the keystone role documentation) intends to explain all the options available for the role and how to implement more advanced requirements. To reduce duplication, the role documentation directly includes the roles default variables file which includes the comments explaining the purpose of the variables. The long hand documentation for the roles should focus less on explaining variables and more on explaining how to implement advanced use cases.

The role documentation must include a description of the mandatory infrastructure (For example: a database and a message queue are required), variables (For example: the database name and credentials) and group names (For example: The role expects a group named `foo_all` to be present and it expects the host to be a member of it) for the roles execution to succeed.

Where possible the documentation in OpenStack-Ansible should steer clear of trying to explain OpenStack concepts. Those explanations belong in the OpenStack Manuals or service documentation and OpenStack-Ansible documentation should link to those documents when available, rather than duplicate their content.

Release Notes

Release notes are generated using the reno tool. Release notes must be written with the following guidelines in mind:

- Each list item must make sense to read without the context of the patch or the repository the patch is being submitted into. The reason for this is that all release notes are consolidated and presented in a long list without reference to the source patch or the context of the repository.
- Each note should be brief and to the point. Try to avoid multi-paragraph notes. For features the note should typically refer to documentation for more details. For bug fixes the note can refer to a registered bug for more details.

In most cases only the following sections should be used for new release notes submitted with patches:

- `features`: This should inform the deployer briefly about a new feature and should describe how to use it either by referencing the variables to set or by referring to documentation.
- `issues`: This should inform the deployer about known issues. This may be used when fixing an issue and wanting to inform deployers about a workaround that can be used for versions prior to that which contains the patch that fixes the issue. Issue notes should specifically make mention of what versions of OpenStack-Ansible are affected by the issue.
- `upgrade`: This should inform the deployer about changes which may affect them when upgrading from a previous major or minor version. Typically, these notes would describe changes to default variable values or variables that have been removed.
- `deprecations`: If a variable has been deprecated (ideally using the deprecation filter), then it should be communicated through notes in this section. Note that if a variable has been removed entirely then it has not been deprecated and the removal should be noted in the `upgrade` section.

Submitting a specification

By proposing a draft spec you can help the OpenStack-Ansible community keep track of what roles or large changes are being developed, and perhaps connect you with others who may be interested and able to help you in the process.

Our specifications repository follows the usual OpenStack and OpenStack-Ansible guidelines for submitting code.

However, to help you in the writing of the specification, we have a specification template that can be copied into the latest release name folder. Rename and edit it for your needs.

Ansible Style Guide

YAML formatting

When creating tasks and other roles for use in Ansible please create them using the YAML dictionary format.

Example YAML dictionary format:

```
- name: The name of the tasks
  module_name:
    thing1: "some-stuff"
    thing2: "some-other-stuff"
  tags:
    - some-tag
    - some-other-tag
```

Example what **NOT** to do:

```
- name: The name of the tasks
  module_name: thing1="some-stuff" thing2="some-other-stuff"
  tags: some-tag
```

```
- name: The name of the tasks
  module_name: >
    thing1="some-stuff"
    thing2="some-other-stuff"
  tags: some-tag
```

Usage of the > and | operators should be limited to Ansible conditionals and command modules such as the Ansible `shell` or `command`.

Tags and tags conventions

Note

If you want to learn more about how to use Ansible tags effectively, check out the Operations Guide.

Tags are assigned based on the relevance of each individual item. Higher level includes (for example in the `tasks/main.yml`) need high level tags. For example, `*-config` or `*-install`. Included tasks can have more detailed tags.

The following convention is used:

- A tag including the word `install` handles software installation tasks. Running a playbook with `--tags <role>-install` only deploys the necessary software on the target, and will not configure it to your needs or run any service.
- A tag including the word `config` prepares the configuration of the software (adapted to your needs), and all the components necessary to run the service(s) configured in the role. Run-

ning a playbook with `--tags <role>-config` is only possible if the target already ran the tags `<role>-install`.

Variable files conventions

The variables files in a role are split in 3 locations:

1. The *defaults/main.yml* file
2. The *vars/main.yml* file
3. The *vars/<platform specific>.yml* file

The variables with lower priority should be in the *defaults/main.yml*. This allows their overriding with group variables or host variables. A good example for this are default database connection details, default queues connection details, or debug mode.

In other words, *defaults/main.yml* contains variables that are meant to be overridable by a deployer or a continuous integration system. These variables should be limited as much as possible, to avoid increasing the test matrix.

The *vars/main.yml* is always included. It contains generic variables that arent meant to be changed by a deployer. This includes for example static information that arent distribution specific (like aggregation of role internal variables for example).

The *vars/<platform specific>.yml* is the place where distribution specific content will be stored. For example, this file will hold the package names, repositories urls and keys, file paths, service names/init scripts.

Secrets

Any secrets (For example: passwords) should not be provided with default values in the tasks, role vars, or role defaults. The tasks should be implemented in such a way that any secrets required, but not provided, should result in the task execution failure. It is important for a secure-by-default implementation to ensure that an environment is not vulnerable due to the production use of default secrets. Deployers must be forced to properly provide their own secret variable values.

Task files conventions

Most OpenStack services will follow a common series of stages to install, configure, or update a service deployment. This is apparent when you review *tasks/main.yml* for existing roles.

If developing a new role, please follow the conventions set by existing roles.

Tests conventions

The conventions for writing tests are described in the *Testing* page.

Other OpenStack-Ansible conventions

To facilitate the development and tests implemented across all OpenStack-Ansible roles, a base set of folders and files need to be implemented. A base set of configuration and test facilitation scripts must include at least the following:

- `tox.ini`: The lint testing, documentation build, release note build and functional build execution process for the roles gate tests are all defined in this file.

- `test-requirements.txt`: The Python requirements that must be installed when executing the tests.
- `bindep.txt`: The binary requirements that must be installed on the host the tests are executed on for the Python requirements and the tox execution to work. This must be copied from the `openstack-ansible-tests` repository and will be automatically be overridden by our proposal bot should any change happen.
- `setup.cfg` and `setup.py`: Information about the repository used when building artifacts.
- `README.rst`, `LICENSE`, `CONTRIBUTING.rst`: A set of standard files whose content is self-explanatory.
- `.gitignore`: A standard git configuration file for the repository which should be pretty uniform across all the repositories. This must be copied from the `openstack-ansible-tests` repository and will be automatically be overridden by our proposal bot should any change happen.
- `.gitreview`: A standard file configured for the project to inform the `git-review` plugin where to find the upstream gerrit remote for the repository.
- `docs/` and `releasenotes/` folders need to be exist and be properly configured.

Please have a look at a role like os_cinder, os_keystone, or os_neutron for latest files.

Container technology independence

The role implementation should be done in such a way that it is agnostic with regards to whether it is implemented in a container, or on a physical host. The test infrastructure may make use of containers for the separation of services, but if a role is used by a playbook that targets a host, it must work regardless of whether that host is a container, a virtual server, or a physical server. The use of containers for role tests is not required but it may be useful in order to simulate a multi-node build out as part of the testing infrastructure.

Minimum supported distributions

See our *Supported distributions* page.

Testing

Adding tests to a new role

Each of the role tests is in its tests/ folder.

This folder contains at least the following files:

- `test.yml` (super playbook acting as test router to sub-playbooks)
- `<role name>-overrides.yml`. This var file is automatically loaded by our shell script in our tests repository.
- `inventory`. A static inventory for role testing. Its possible some roles have multiple inventories. See for example the neutron role with its `ovs_inventory`.
- `group_vars` and `host_vars`. These folders will hold override the necessary files for testing. For example, this is where you override the IP addresses, IP ranges, and ansible connection details.
- `ansible-role-requirements.yml`. This should be fairly straightforward: this file contains all the roles to clone before running your role. The roles relative playbooks will have to be listed in

the `test.yml` file. However, keep in mind to NOT re-invent the wheel. For example, if your role needs keystone, you dont need to create your own keystone install playbook, because we have a generic keystone install playbook in the *tests repository*.

- Only add a `zuul.d` folder when your role is imported into the openstack-ansible namespace.

Extending tests of an existing role

1. Modify the tox.ini to add your new scenario. If required, you can override the inventory, and/or the variable files.
2. Add a new non-voting job in `zuul.d/jobs.yaml`, and wire it in the project tests file `zuul.d/project.yaml`.

Improve testing with tempest

Once the initial convergence is working and the services are running, the role development should focus on implementing some level of functional testing.

Ideally, the functional tests for an OpenStack role should make use of Tempest to execute the functional tests. The ideal tests to execute are scenario tests as they test the functions that the service is expected to do in a production deployment. In the absence of any scenario tests for the service a fallback option is to implement the smoke tests instead.

If no tempest is provided, some other functional testing should be done. For APIs, you can probably check the HTTP response codes, with specially crafted requests.

Running tests locally

Linting

Python coding conventions are tested using PEP8, with the following convention exceptions:

- F403 - from ansible.module_utils.basic import *

Testing may be done locally by executing:

```
./run_tests.sh pep8
```

Bash coding conventions are tested using Bashate, with the following convention exceptions:

- E003: Indent not multiple of 4. We prefer to use multiples of 2 instead.
- **E006: Line longer than 79 columns. As many scripts are deployed as templates** and use jinja templating, this is very difficult to achieve. It is still considered a preference and should be a goal to improve readability, within reason.
- **E040: Syntax error determined using *bash -n*. As many scripts are deployed** as templates and use jinja templating, this will often fail. This test is reasonably safely ignored as the syntax error will be identified when executing the resulting script.

Testing may be done locally by executing:

```
./run_tests.sh bashate
```

Ansible is lint tested using ansible-lint.

Testing may be done locally by executing:

```
./run_tests.sh ansible-lint
```

Ansible playbook syntax is tested using ansible-playbook.

Testing may be done locally by executing:

```
./run_tests.sh ansible-syntax
```

A consolidated set of all lint tests may be done locally by executing:

```
./run_tests.sh linters
```

Documentation building

Documentation is developed in reStructuredText (RST) and compiled into HTML using Sphinx.

Documentation may be built locally by executing:

```
./run_tests.sh docs
```

The OpenStack-Ansible integrated repo also has an extra documentation building process, to build the deployment guide.

This guide may be built locally by executing:

```
./run_tests.sh deploy-guide
```

Release notes building

Release notes are generated using the the reno tool and compiled into HTML using Sphinx.

Release notes may be built locally by executing:

```
./run_tests.sh releasenotes
```

Note

The `releasenotes` build argument only tests committed changes. Ensure your local changes are committed before running the `releasenotes` build.

Roles functional or scenario testing

To run a functional test of the role, execute:

```
./run_tests.sh functional
```

Testing a new role with an AIO

1. Include your role on the deploy host. See also *Adding new or overriding roles in your OpenStack-Ansible installation*.

2. Perform any other host preparation (such as the tasks performed by the `bootstrap-aio.yml` playbook). This includes any preparation tasks that are particular to your service.

3. Generate files to include your service in the Ansible inventory using `env.d` and `conf.d` files for use on your deploy host.

   Hint

   You can follow examples from other roles, making the appropriate modifications being sure that group labels in `env.d` and `conf.d` files are consistent.

   Hint

   A description of how these work can be found in *Understanding host groups (conf.d structure)* and *Understanding container groups (env.d structure)*.

4. Generate secrets, if any, as described in the Configure Service Credentials. You can append your keys to an existing `user_secrets.yml` file or add a new file to the `openstack_deploy` directory to contain them. Provide overrides for any other variables you will need at this time as well, either in `user_variables.yml` or another file.

   See also our *Overriding default configuration* page.

   Any secrets required for the role to work must be noted in the `etc/openstack_deploy/user_secrets.yml` file for reuse by other users.

5. If your service is installed from source or relies on python packages which need to be installed from source, specify a repository for the source code of each requirement by adding a file to your deploy host under `inventory/group_vars/<service_group>/source_git.yml` in the OpenStack-Ansible source repository and following the pattern of files currently in that directory. You could also simply add an entry to an existing file there.

6. Make any required adjustments to the load balancer configuration (e.g. modify `inventory/group_vars/all/haproxy.yml` in the OpenStack-Ansible source repository on your deploy host) so that your service can be reached through a load balancer, if appropriate, and be sure to run the `haproxy-install.yml` play later so your changes will be applied. Please note, you can also use `haproxy_extra_services` variable if you dont want to provide your service as default for everyone.

7. Put together a service install playbook file for your role. This can also be modeled from any existing service playbook that has similar dependencies to your service (database, messaging, storage drivers, container mount points, etc.). A common place to keep playbook files in a Galaxy role is in an `examples` directory off the root of the role. If the playbook is meant for installing an OpenStack service, name it `os-<service>-install.yml` and target it at the appropriate group defined in the service `env.d` file. It is crucial that the implementation of the service is optional and that the deployer must opt-in to the deployment through the population of a host in the applicable host group. If the host group has no hosts, Ansible skips the playbooks tasks automatically.

8. Any variables needed by other roles to connect to the new role, or by the new role to connect to other roles, should be implemented in `inventory/group_vars`. The group vars are essentially the glue which playbooks use to ensure that all roles are given the appropriate information. When group vars are implemented it should be a minimum set to achieve the goal of integrating the new role into the integrated build.

9. Documentation must be added in the role to describe how to implement the new service in an integrated environment. This content must adhere to the *Documentation and Release Note Guidelines*. Until the role has integrated functional testing implemented (see also the Role development maturity paragraph), the documentation must make it clear that the service inclusion in OpenStack-Ansible is experimental and is not fully tested by OpenStack-Ansible in an integrated build. Alternatively, an user story can be created.
10. A feature release note must be added to announce the new service availability and to refer to the role documentation for further details. This content must adhere to the *Documentation and Release Note Guidelines*.
11. It must be possible to execute a functional, integrated test which executes a deployment in the same way as a production environment. The test must execute a set of functional tests using Tempest. This is the required last step before a service can remove the experimental warning from the documentation.

Hint

If you adhere to the pattern of isolating your roles extra deployment requirements (secrets and var files, HAProxy yml fragments, repo_package files, etc.) in their own files it makes it easy for you to automate these additional steps when testing your role.

Integrated repo functional or scenario testing

To test the integrated repo, follow the Deployment Guide

Alternatively, you can check the *aio guide*, or even run the gate wrapper script, named `scripts/gate-check-commit.sh`, described below.

The OpenStack Infrastructure automated tests

There should be no difference between running tests in the openstack infrastructure, versus running locally.

The tests in the openstack infrastructure are triggered by jobs defined in each repo `zuul.d` folder.

See also the zuul user guide.

However, for reliability purposes, a few variables are defined to point to the OpenStack infra pypi and packages mirrors.

The integrated repo functional test is using the `scripts/gate-check-commit.sh` script, which receives arguments from the zuul run playbook definition.

While this script is primarily developed and maintained for use in OpenStack-CI, it can be used in other environments.

Role development maturity

A role may be fully mature, even if it is not integrated in the `openstack-ansible` repository. The maturity depends on its testing levels.

A role can be in one of the four maturity levels:

- `Complete`

- `Incubated`
- `Unmaintained`
- `Retired`

Here are a series of rules that define maturity levels:

- A role can be retired at any time if it is not relevant anymore.
- A role can be `Incubated` for maximum 2 cycles.
- An `Incubated` role that passes functional testing will be upgraded to the `Complete` status, and cannot return in `Incubated` status.
- An `Incubated` role that didnt implement functional testing in the six month timeframe will become `Unmaintained`.
- A role in `Complete` status can be downgraded to `Unmaintained`. status, according to the maturity downgrade procedure.

Maturity downgrade procedure

If a role has failed periodics or gate test for two weeks, a bug should be filed, and a message to the mailing list will be sent, referencing the bug.

The next community meeting should discuss about role deprecation, and if no contributor comes forward to fix the role, periodic testing will be turned off, and the role will move to an `unmaintained` state.

Maturity Matrix

All of the OpenStack-Ansible roles do not have the same level of maturity and testing.

Here is a dashboard of the current status of the roles:

Periodic Work

Releasing

Our release frequency is discussed in *Releases*.

OSA CLI tooling

OpenStack-Ansible used to bump everything in a single script, which made it hard to maintain, and was very branch specific. It made it hard for users to consume either an update of the upstream shas, or to bump roles with their own pace.

Since then, the OpenStack-Ansible has agreed to provide more metadata necessary for releasing into the openstack-ansible code tree. This allowed the tooling for releasing to be more flexible, and lighter, over time.

All the functions are separated, and included as an extra console script `openstack-ansible-releases`.

By default, dependencies for the script are not installed to minimize amount of them for regular users. To ensure all required dependencies are installed for the script, you can run:

```
pip3 install -e git+https://opendev.org/openstack/openstack-ansible
↪#egg=openstack-ansible[releases]
```

Each subcommand contains help by default.

Updating upstream SHAs

The dependencies for OpenStack-Ansible are updated through the use of `openstack-ansible-releases bump_upstream_shas`. This script updates the projects pinned SHAs, located in the `inventory/group_vars/<service_group>/source_git.yml` file, based on their `_git_track_branch` value.

Updating OpenStack-Ansible roles

Updating the roles to their latest version per branch is done through `openstack-ansible-releases bump_roles`.

This can do multiple things:

- Freeze ansible-role-requirements to their latest SHAs for the branch they are tracking.
- Copy release notes relevant to the freeze.
- Unfreeze of master.

Master doesnt get frozen, unless explicitly asked for it for release milestones, using the command `openstack-ansible-releases freeze_roles_for_milestone`

Updating required collections

Updating collections to their latest version is done through `openstack-ansible-releases bump_collections`.

Please note, that only collections with type *git* are supported. Ones, that are installed from Ansible Galaxy should be bumped manually.

Check current pins status

You can check the current PyPI pins that are used in openstack-ansible repository by running `openstack-ansible-releases check_pins`. This will display a table, showing the current pin in OSA, and what is available upstream on PyPI.

This doesnt patch the `global-requirements-pins`, as this should be a manual operation. See the *Development cycle checklist* to know when to bump the global-requirements-pins.

Adding patch in releases repo

When the patches to update SHAs and roles have landed, you can propose the parent SHA as a release in the releases repo.

This can be done using the `new-release` command, and then editing the SHA used for openstack-ansible. See also new_releases page for an explanation of this command.

Please note that branches before Stein will require cleanup of the YAML file generated by new_releases, as it will contain ALL the roles and openstack-ansible repo SHAs. We have decided to NOT tag the roles anymore, so you will need to remove all the lines which are not relevant to the *openstack-ansible* repository.

Trasition releases to Extended Maintenance

With migrating a release to Extended Maintenance (EM) status no future releases of the release are possible. With that, a new tag <release>-em is created and it must be based on the latest numeric release.

To transition a release to Extended Maintenance we need to:

- Wait until all upstream repositories will transition to EM
- Update OpenStack-Ansible roles normally
- Update `<service>_git_install_branch` to be `<release>-em` instead of the SHA.
- Propose a new numeric release to `openstack/releases` repository
- Propose to create a `<release>-em` tag with the same SHA as previous numeric release has.
- Once EM tag is created, switch `version` field in ansible-role-requirements.yml to track `stable/<release>`.
- Update `index.rst` for master branch to reflect support status of the release

Trasition releases to End Of Life

With releases reaching End Of Life (EOL), a new tag `<release>-eol` is created, after which branch will be deleted. Projects are free to EOL their branches anytime, after coordinated migration to EM is completed. Due to this we dont track `stable/<release>` branch for upstream services for EM, but only for our roles. At the same time there is a coordinated deadline, when all projects must EOL their old branches.

To transition a release to End Of Life we need to:

- Create a <release>-eol tag for OpenStack-Ansible Roles
- Switch ansible-role-requirements.yml `version` field to <release>-eol tag from stable/<release> branch.
- Wait for all projects to EOL
- Update `<service>_git_install_branch` variables to use <release>-eol tag instead of SHAs.
- Create a <release>-eol tag for OpenStack-Ansible repository
- Update release support status in index.rst on the master branch.

Development cycle checklist

On top of the normal cycle goals, a contributor can help the OpenStack-Ansible development team by performing one of the following recurring tasks:

- By milestone 1:
  - Community goal acknowledgement.
  - Define supported platforms release will run on. Remove testing and support for deprecated ones.
  - Update `global-requirements-pins`, upstream SHAs and required collections
- By milestone 2:
  - Handle deprecations from upstream projects previous cycle.

  - Handle OpenStack-Ansible roles deprecations from the previous cycle.
  - Refresh static elements in roles. For example, update a specific version of the software packages.
  - Update release-versioned components such as Octavia test ampohora image and Ironic IPA image/kernel.
  - Bump `ceph_stable_release` to latest Ceph LTS release in the integrated OpenStack-Ansible repo, and inside the `ceph_client` role defaults.
  - Check and bump galera versions if required.
  - Check and bump rabbitmq versions if required.
  - Check outstanding reviews and move to merge or abandon them if no longer valid.
  - Update `global-requirements-pins`, upstream SHAs and required collections
- By milestone 3:
  - Implement features
  - Update `global-requirements-pins`, upstream SHAs and required collections
- After milestone 3:
  - Feature freeze, bug fixes, and testing improvements.
  - Ansible version and collections freeze
- After a new stable branch is created for services:
  - Update `<service>_git_track_branch` variables to match the new branch name.
  - Set all `tempest_plugin_<service>_git_track_branch` to None to prevent SHA update for them.
  - Update upstream SHAs
- After coordinated OpenStack release, before OpenStack-Ansible release:
  - Update release name in `openstack_hosts` repository. This will also bump RDO and Ubuntu Cloud Archive repositories.
  - Branch all the independent repos that arent part of the release in gerrit. See also the `projects.yaml` in the governance repo. Manually branched repos need extra editions, like updating the .gitreview, or the reno index. Please reference previous branch creations by using the appropriate topic in gerrit (e.g.: `create-stein`). The previous new branch creation may be different as the tooling/process may have evolved, so be aware that the changes needed may need to be adapted.
  - Switch `trackbranch` in ansible-role-requirements.yml to the new branch and update OpenStack-Ansible roles after that.
  - Add supported platforms for the release to `os-compatibility-matrix.html`
- Immediately after official OpenStack-Ansible release:
  - Send a thank you note to all the contributors through the mailing lists. They deserve it.
  - Revert changes made to ansible-role-requirements.yml and `*_git_track_branch` variables to track stable branch instead of master. Update upstream SHAs.

- Reflect changes in documentation
  - Create a patch to openstack-manuals and uncomment OpenStack-Ansible in *www/project-data/<release>.yaml*.
  - Update index page on master to mention release date of recently released version as well as set its status to Maintained. With that also add a new release that we are about to start working on.
  - Update index page on stable branch to mention only the release in topic rather all historical releases as well. Historical data should be present only on master branch.
  - Update upgrade scripts and documentation to keep track on supported upgrade paths:
    - For SLURP releases, define `previous_slurp_name` in doc/source/conf.py. For non-SLURP - set it to `False`.
    - Update `previous_series_name` and `current_series_name` in doc/source/conf.py and deploy-guide/source/conf.py
    - Update `UPGRADE_SOURCE_RELEASE` in scripts/gate-check-commit.sh
    - Update `SUPPORTED_SOURCE_SERIES` and `TARGET_SERIES` in scripts/run-upgrade.sh. Also dont forget to cleanup irrelevant upgrade scripts.
    - Add/remove required for SLURP upgrade jobs. ACTION for such jobs should be defined as `upgrade_<release>`.

Core Reviewers

General Responsibilities

The OpenStack-Ansible Core Reviewer Team is responsible for many aspects of the OpenStack-Ansible project. These include, but are not limited to:

- Mentor community contributors in solution design, testing, and the review process
- Actively reviewing patch submissions, considering whether the patch: - is functional - fits the use-cases and vision of the project - is complete in terms of testing, documentation, and release notes - takes into consideration upgrade concerns from previous versions
- Assist in bug triage and delivery of bug fixes
- Curating the gate and triaging failures
- Maintaining accurate, complete, and relevant documentation
- Ensuring the level of testing is adequate and remains relevant as features are added
- Answering questions and participating in mailing list discussions
- Interfacing with other OpenStack teams

In essence, core reviewers share the following common ideals:

- They share responsibility in the projects success in its mission.
- They value a healthy, vibrant, and active developer and user community.
- They have made a long-term, recurring time investment to improve the project.
- They spend their time doing what needs to be done to ensure the projects success, not necessarily what is the most interesting or fun.

- A core reviewers responsibility doesnt end with merging code.

Core Reviewer Expectations

Members of the core reviewer team are expected to:

- Attend and participate in the weekly IRC meetings
- Monitor and participate in-channel at #openstack-ansible
- Monitor and participate in OpenStack-Ansible discussions on the mailing list
- Participate in team sessions at the OpenStack Projects Team Gatherings (PTG)
- Participate in Forum sessions at the OpenStack Summits
- Review patch submissions actively and consistently

Please note in-person attendance at PTG, Summit, mid-cycles, and other code sprints is not a requirement to be a core reviewer. The team will do its best to facilitate virtual attendance at all events. Travel is not to be taken lightly, and we realize the costs involved for those who attend these events.

Code Merge Responsibilities

While everyone is encouraged to review changes, members of the core reviewer team have the ability to set +2/-2 on the Code-Review (CR) label as well as +1 on Workflow (+W) changes to these repositories. This is an extra level of responsibility not to be taken lightly. Correctly merging code requires not only understanding the code itself, but also how the code affects things like documentation, testing, upgrade impacts and interactions with other projects. It also means you pay attention to release milestones and understand if a patch you are merging is marked for the release, especially critical during the feature freeze.

Code Merge Policies

Below you will find general policies on the Code-Review process and when a patch may be considered as ready for merge and when to +W.

It is the responsibility of the Core Reviewer, who reviews the change last, to set the +W label once a change passes the policy. Also, before setting +W please make sure that all dependant patches (marked with `Depends-On` in a commit message) are already merged to avoid unnecessary rechecks or case dependant patch(s) will fail in the gates.

All changes can be split into multiple categories and a slightly different policy may apply for each category.

New features, blueprints, design changes

- Minimum 2 Core Reviewers, excluding the patch owner, voted +2 on Code-Review label
- Voted Code-Reviewers should be representing minimum 2 different organizations or be unaffiliated for diversity reasons

Bug fixes, version bumps

- Minimum 2 Core Reviewers, excluding the patch owner, voted +2 on Code-Review label
- It is allowed for all voted Core Reviewers to be affilated with the same organization

Automated (bot) changes

- Minimum 1 Core Reviewer, excluding the patch owner, voted +2 on Code-Review label

Backports to stable branches

- Minimum 2 Core Reviewers, *including* the patch owner, voted +2 on Code-Review label.
- It is allowed for all voted Core Reviewers to be affilated with the same organization

Backports to unmaintained branches

- Minimum 1 Core Reviewer, excluding the patch owner, voted +2 on Code-Review label

Distribution support

Supported distributions

The list of supported distributions can be found in the *Compatibility Matrix*

Minimum requirements for OpenStack-Ansible roles

Existing and new distributions are expected to meet the following requirements in order for them to be accepted in the individual OpenStack-Ansible roles:

- Pass functional tests

Graduation

For a distribution to be considered supported by the OpenStack-Ansible project, it has to meet the following minimum requirements:

- The necessary steps for bootstrapping the operating system have to be documented in the Deployment Guide.
- The integration repository contains at least one job which passes the Temptest testing framework.

Voting

Distributions can be promoted to voting jobs on individual roles once they move to the *Graduation* phase and their stability has been confirmed by the core OpenStack-Ansible developers. Similar to this, voting can also be enabled in the integration repository for all the scenarios or a specific one.

PYTHON MODULE INDEX

d

INDEX

D

M