OpenStack-Ansible Documentation

OpenStack-Ansible Contributors

Jan 05, 2024

CONTENTS

CHAPTER

ONE

ABSTRACT

OpenStack-Ansible provides Ansible playbooks and roles for the deployment and configuration of an OpenStack environment.

OpenStack-Ansible Train series was first released with the 20.0.0 tag on 16 December 2019.

Warning: This release has reached its End Of Life.

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 Contributors Guide.

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.

1.1.1 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` tenant 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 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 | cinder       | volume         | True    | public  ␣
↪ | https://10.23.100.127:8776/v1/%(project_id)s      |
| [ID truncated] | RegionOne | aodh         | alarming       | True    | internal␣
↪ | http://172.29.236.100:8042                        |
| [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    |
```

(continues on next page)

(continued from previous page)

```
| [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 | ceilometer   | metering       | True    | admin  ␣
↪ | http://172.29.236.100:8777/                       |
| [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 | cinder       | volume         | True    | internal␣
↪ | http://172.29.236.100:8776/v1/%(project_id)s      |
| [ID truncated] | RegionOne | heat-cfn     | cloudformation | True    | public ␣
↪ | https://10.23.100.127:8000/v1                     |
| [ID truncated] | RegionOne | ceilometer   | metering       | True    | public ␣
↪ | https://10.23.100.127:8777                        |
| [ID truncated] | RegionOne | designate    | dns            | True    | public ␣
↪ | http://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    | internal␣
↪ | http://172.29.236.100:8004/v1/%(project_id)s      |
| [ID truncated] | RegionOne | ceilometer   | metering       | True    | internal␣
↪ | http://172.29.236.100:8777                        |
| [ID truncated] | RegionOne | cinder       | volume         | True    | admin  ␣
↪ | http://172.29.236.100:8776/v1/%(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                     |
```

(continues on next page)

(continued from previous page)

```
| [ID truncated] | RegionOne | glance       | image          | True    | admin    ␣
↪ | http://172.29.236.100:9292                          |
+----------------+-----------+--------------+----------------+---------+----------
↪-+-----------------------------------------------------+
```

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-bionic                          | 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 | Linux bridge agent  | ubuntu-bionic      ␣
↪                     | None              | :-)   | UP    | neutron-linuxbridge-
↪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 | Linux bridge agent  | aio1-neutron-
↪agents-container-2b0569d5 | None              | :-)   | UP    | neutron-
↪linuxbridge-agent |
| feb20ed4-4346-4ad9-b50c-41efd784f2e9 | L3 agent            | aio1-neutron-
↪agents-container-2b0569d5 | nova              | :-)   | UP    | neutron-l3-
↪agent          |
```

(continues on next page)

(continued from previous page)

```
+--------------------------------------+--------------------+------------------
↪--------------------+------------------+-------+-------+---------------------
↪-----+
```

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 for the latter is in the `openstack-ansible-ops` repository, under the `openstack-service-setup.yml` playbook.

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
- Image Service (glance) CLI commands
- Image Service (glance) CLI command cheat sheet
- Compute (nova) CLI commands
- Compute (nova) CLI command cheat sheet
- Networking (neutron) CLI commands
- Networking (neutron) CLI command cheat sheet
- Block Storage (cinder) CLI commands
- Block Storage (cinder) CLI command cheat sheet
- python-keystoneclient
- python-glanceclient
- python-novaclient
- python-neutronclient

1.1.2 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 **Compute** project from the drop down list.
2. Click the **Images** option.
3. Locate the image that will act as the instance base from the **Images** table.
4. Click **Launch** from the **Actions** column.

5. Check the **Launch Instances** dialog, and find the **details** tab. Enter the appropriate values for the instance.
    1. In the Launch Instance dialog, click the **Access & Security** tab. Select the keypair. Set the security group as default.
    2. Click the **Networking tab**. This tab will be unavailable if OpenStack networking (neutron) has not been enabled. If networking is enabled, select the networks on which the instance will reside.
    3. Click the **Volume Options tab**. This tab will only be available if a Block Storage volume exists for the instance. Select **Dont boot from a volume** for now.

       For more information on attaching Block Storage volumes to instances for persistent storage, see the *Managing volumes for persistent storage* section below.
    4. Add customisation scripts, if needed, by clicking the **Post-Creation** tab. 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.
    5. Click **Advanced Options**. Specify whether the instance uses a configuration drive to store metadata by selecting a disk partition type.
6. Click **Launch** to create the instance. The instance will start on a compute node. The **Instance** page will open and start creating a new instance. The **Instance** 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.

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 `ami`, `ari`, `aki`, `bare`, or `ovf`
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 Name	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 forwarded to instances.
Keypair	Optional	Specify a key pair with this option. If the image uses a static key set (not recommended), a key pair is not needed.
Selected Networks	Optional	To add a network to an instance, click the **+** in the **Networks field**.
Customisation Script	Optional	Specify a customisation script. This script runs after the instance launches and becomes active.

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             |
|               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 **More** option. Select the instance state.

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

- Resize or rebuild any instance
- View the instance console log
- Edit the instance
- Modify security groups
- Pause, resume, or suspend the instance
- Soft or hard reset the instance

Note: Terminate 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.

Nova 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 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 the config drive, add the following override to the `/etc/openstack_deploy/user_variables.yml` file:

```
nova_force_config_drive: False
```

To change the format of the configuration drive, to a hard disk style format, use the following configuration inside the same `/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`
- xen defaults to `xenmigr://%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 Nova instances are stored on shared storage and KVM/Libvirt only need to synchronize the memory and base image of the Nova 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 Nova 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        ␣
↪   |
+----+-------------+-----------+----------------+--------------+-----------+----------
↪-+---------------+------------+------------+------------+------------+--------------
↪---+
| 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 Nova 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
```

1.1.3 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.

MySQL 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 "mysql -h 127.0.0.1 \
-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
   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/mysql start --wsrep-new-cluster
## for systemd
# systemctl set-environment _WSREP_NEW_CLUSTER='--wsrep-new-cluster'
# systemctl start mysql
# systemctl set-environment _WSREP_NEW_CLUSTER=''
```

Please also have a look at upstream starting a cluster page

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

```
# ansible galera_container -m shell -a "/etc/init.d/mysql 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
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 `galera-install` playbook using the `galera-bootstrap` tag to automatically recover a node or an entire environment.

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

```
# openstack-ansible galera-install.yml --tags galera-bootstrap
```

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 "mysql -h 127.0.0.1 \
-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
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 `mysqld` 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 "mysql \
-h 127.0.0.1 -e 'show status like \"%wsrep_cluster_%\";'"
node2_galera_container-49a47d25 | FAILED | rc=1 >>
ERROR 2002 (HY000): Can't connect to local MySQL server
```

(continues on next page)

(continued from previous page)

```
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:

```
# mysql -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

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 "mysql \
-h 127.0.0.1 -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
```

(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
```

4. If MariaDB fails to start on any of the failed nodes, run the `mysqld` 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
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 `mysqld` 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:

```
# lxc-stop -n node3_galera_container-3ea2cbd3
# lxc-destroy -n node3_galera_container-3ea2cbd3
# rm -rf /openstack/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 setup-hosts.yml -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 setup-infrastructure.yml \
--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 "mysql \
-h 127.0.0.1 -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 `/var/log/mysql_logs/galera_server_error.log` 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 "mysql \
-h 127.0.0.1 -e 'show status like \"%wsrep_cluster_%\";'"
node2_galera_container-49a47d25 | 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

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.

Note: There is currently an Ansible bug in regards to `HOSTNAME`. If the host `.bashrc` holds a var named `HOSTNAME`, the container where the `lxc_container` module attaches will inherit this var and potentially set the wrong `$HOSTNAME`. See the Ansible fix which will be released in Ansible version 2.3.

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 ...
```

(continues on next page)

(continued from previous page)

```
[{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 Rabbit 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 Rabbit and are in a single Rabbit 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 unresponsive 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 Rabbit2 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
```

(continues on next page)

(continued from previous page)

```
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.

Running ad-hoc Ansible plays

Being familiar with running ad-hoc Ansible commands is helpful when operating your OpenStack-Ansible deployment. For example, if we 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 being the hostname command. You can substitute the group for any other 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 commands. You can also use the `*` wild card if you only know the first part of the group name, for example, `compute_h*`. The `-m` argument is for module.

Modules can be used to control system resources, or handle the execution of system commands. For a 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`:

```
$ 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 "mysql -h 127.0.0.1\
-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. Use the fetch module to copy files from remote locations to the local machine. 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'
```

If you want to gather files from remote machines, use the fetch module. The fetch module stores files locally in a file tree, organized by the hostname from remote machines and stores them locally in a file tree, organized by 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 `yes`.

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
```

(continues on next page)

(continued from previous page)

```
}

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 and Understanding ansible 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.

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:

- OpenStack-Ansible Bug 1479812
- Ansible forks entry for ansible.cfg
- Ansible Performance Tuning

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 is 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.

Firewalls

OpenStack-Ansible does not configure firewalling for its infrastructure. It is up to the deployer to define the perimeter and its firewalling 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 firewalling, see the Firewalls and default ports

You can find in each of the roles respective documentatione, 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-Ansibles 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 role 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 whole list of 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

Its 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
```

(continues on next page)

(continued from previous page)

```
-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
```

1.1.4 Scaling your environment

This is a draft environment scaling page for the proposed OpenStack-Ansible operations guide.

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 *Back up 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:
   [...]
   NEW_infra<node-ID>:
     ip: 10.17.136.32
   NEW_infra<node-ID2>:
     ip: 10.17.136.33
   ```

2. Change to playbook folder on the deployment host.

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

3. Update the inventory to add new hosts. Make sure new rsyslog container names are updated. Send the updated results to `dev/null`.

   ```
   # /opt/openstack-ansible/inventory/dynamic_inventory.py > /dev/null
   ```

4. Create the `/root/add_host.limit` file, which contains all new node host names and their containers. Add **localhost** to the list of hosts to be able to access deployment host facts.

   ```
   localhost
   NEW_infra<node-ID>
   NEW_infra<node-ID2>
   NEW_infra<node-ID>_containers
   NEW_infra<node-ID2>_containers
   ```

5. Run the `setup-everything.yml` playbook with the `limit` argument.

```
# openstack-ansible setup-everything.yml --limit @/root/add_host.limit
```

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 setup-hosts.yml --limit localhost,NEW_HOST_NAME
# ansible nova_all -m setup -a 'filter=ansible_local gather_subset="!all"'
# openstack-ansible setup-openstack.yml --limit localhost,NEW_HOST_NAME
```

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.

Remove a compute host

The openstack-ansible-ops 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
# stop nova-compute
# stop neutron-linuxbridge-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 MySQL 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-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 the instance 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:

```
# mysql 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 | fgrep 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 | fgrep vm_state
```

Destroying Containers

1. To destroy a container, execute the following:

```
# cd /opt/openstack-ansible/playbooks
# openstack-ansible lxc-containers-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.

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

```
# cd /opt/openstack-ansible/playbooks
# openstack-ansible lxc-containers-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.

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

+----------------------------------+
| default_project_id               |
```

(continues on next page)

(continued from previous page)

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

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

```
# mysql -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"

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

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

# mysql -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.

1.1.5 Monitoring your environment

This is a draft monitoring system page for the proposed OpenStack-Ansible operations guide.

1.1.6 Back up 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.

Back up 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. Back up 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

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

1.1.7 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, and the Linuxbridge ml2 driver.

Network List

1. `HOST_NET` (Physical Host Management and Access to Internet)
2. `CONTAINER_NET` (LXC container network used Openstack Services)
3. `OVERLAY_NET` (VXLAN overlay network)

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 iptables 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 CONTAINER_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 iptables 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
                                            99999999_eth1
                                            ...
```

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 iptables 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
   ...
```

IP address should be applied to eth10 inside the required LXC containers:

```
# ip address show dev eth10
67: eth10: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 150...UP...
link/ether b1:b1:b1:b1:b1:02 brd ff:ff:ff:ff:ff:ff
inet 172.29.240.55/22 brd 172.29.243.255 scope global eth10
   valid_lft forever preferred_lft forever
   ...
```

`br-vxlan` should contain veth-pair ends from required LXC containers and a physical interface or tagged-subinterface:

```
# brctl show br-vxlan
bridge name     bridge id          STP enabled  interfaces
br-vxlan        8000.ghijkl123456  no           bond1.100
                                                3333333_eth10
```

Checking services

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

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

- Object Storage service (swift)

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	`# service glance-registry restart` `# service glance-api restart`

Compute service (controller node)	`# service nova-api-os-compute restart`
	`# service nova-consoleauth restart`
	`# service nova-scheduler restart`
	`# service nova-conductor restart`
	`# service nova-api-metadata restart`
	`# service nova-novncproxy restart (if using novnc)`
	`# service nova-spicehtml5proxy restart (if using spice)`

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

Networking service	`# service neutron-server restart`
	`# service neutron-dhcp-agent restart`
	`# service neutron-l3-agent restart`
	`# service neutron-metadata-agent restart`
	`# service neutron-linuxbridge-agent restart`

Networking service (compute node)	`# service neutron-linuxbridge-agent restart`

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

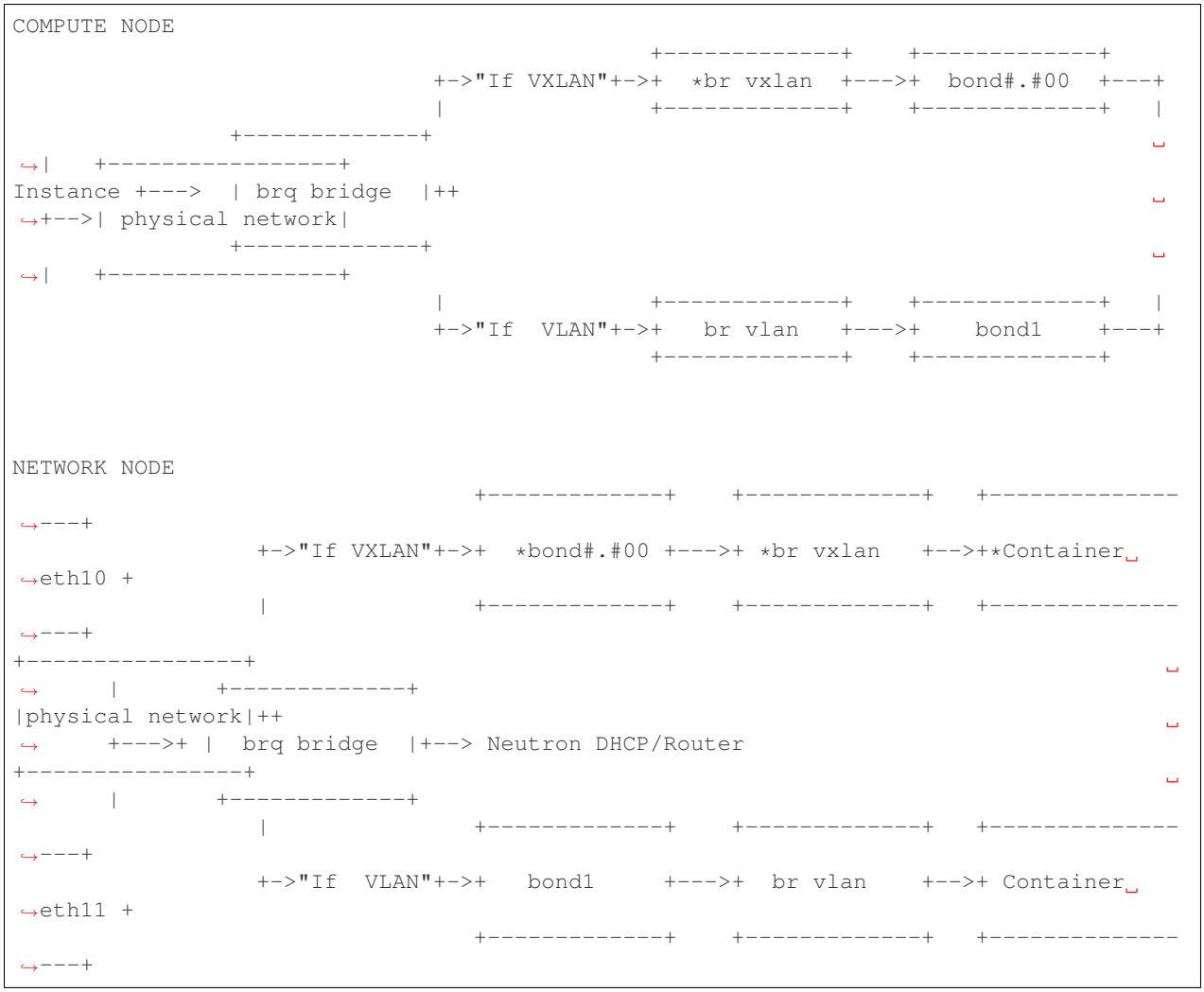
Block Storage service	`# service manila-api restart`
	`# service manila-data restart`
	`# service manila-share restart`
	`# service manila-scheduler restart`

Object Storage service	`# service swift-account-auditor restart`
	`# service swift-account-server restart`
	`# service swift-account-reaper restart`
	`# service swift-account-replicator restart`
	`# service swift-container-auditor restart`
	`# service swift-container-server restart`
	`# service swift-container-reconciler restart`
	`# service swift-container-replicator restart`
	`# service swift-container-sync restart`
	`# service swift-container-updater restart`
	`# service swift-object-auditor restart`
	`# service swift-object-expirer restart`
	`# service swift-object-server restart`
	`# service swift-object-reconstructor restart`
	`# service swift-object-replicator restart`
	`# service swift-object-updater restart`
	`# service swift-proxy-server restart`

Troubleshooting Instance connectivity issues

This section will focus on troubleshooting general instance (VM) connectivity communication. This does not cover any networking related to instance connectivity. This is assuming a OpenStack-Ansible install using LXC containers, VXLAN overlay and the Linuxbridge ml2 driver.

Data flow example



Preliminary troubleshooting questions to answer:

- Which compute node is hosting the VM in question?
- Which interface is used for provider network traffic?
- Which interface is used for VXLAN overlay?
- Is the connectivity issue ingress to the instance?
- Is the 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 tenant 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 Neutron `security-group-rules`, consider adding allow ICMP rule for testing.
- Check that linux bridges contain the proper interfaces. on compute and network nodes.
- Check Neutron DHCP agent logs.
- Check syslogs.
- Check Neutron linux bridge 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 linuxbridge logs.
- Check physical interface mappings.
- Check Neutron Router ports (if applicable).
- Check that linux bridges contain the proper interfaces on compute and network nodes.
- Check Neutron `security-group-rules`, consider adding allow ICMP rule for testing.

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 Neutron `security-group-rules`, consider adding ICMP rule for testing.
- Check Neutron FloatingIP 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:

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 VTEP addresses able to ping each other?

No:

- Check `br-vxlan` interface on Compute and `eth10` inside the Neutron network agent container.
- Check veth pairs between containers and linux bridges on the host.
- Check that linux bridges contain the proper interfaces on compute and network nodes.

Yes:

- Check ml2 config file for local VXLAN IP and other VXLAN 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 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 Neutron `security-group-rules`, consider adding allow ICMP rule for testing.
- Check that linux bridges contain the proper interfaces on compute and network nodes.

- Check Neutron DHCP agent logs.
- Check syslogs.
- Check Neutron linux bridge logs.
- Check that Bridge Forwarding Database (fdb) contains the proper entries on both the compute and Neutron agent container.

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 linux bridge logs.
- Check physical interface mappings.
- Check Neutron router ports (if applicable).
- Check that linux bridges contain the proper interfaces on compute and network nodes.
- Check Neutron `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.

Yes:

- Good! The instance can ping its intended gateway.
- Check gateway or host routes on the Neutron subnet.
- Check Neutron `security-group-rules`, consider adding ICMP rule for testing.
- Check Neutron FloatingIP 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-registry` handles the database operations for managing the storage of the image index and properties. 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` and `/var/log/glance-registry.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 <https://docs.openstack.org/glance/latest/install/verify.html>_.* and *Manage Images <https://docs.openstack.org/glance/latest/admin/manage-images.html>_*

RabbitMQ issues

Analyze RabbitMQ queues

Analyze OpenStack service logs and RabbitMQ logs

Failed security hardening after host kernel upgrade from version 3.13

Ubuntu kernel packages newer than version 3.13 contain a change in module naming from `nf_conntrack` to `br_netfilter`. After upgrading the kernel, run the `openstack-hosts-setup.yml` playbook against those hosts. For more information, see OSA bug 157996.

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.

Failed ansible playbooks during an upgrade

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
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.

1.1.8 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. Ensure that your OpenStack-Ansible code is on the latest Train tagged release:

   ```
   # git checkout 20.2.7
   ```

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

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

4. Change to the playbooks directory:

   ```
   # cd playbooks
   ```

5. Update the hosts:

```
# openstack-ansible setup-hosts.yml
```

6. Update the infrastructure:

```
# openstack-ansible -e rabbitmq_upgrade=true \
setup-infrastructure.yml
```

7. Update all OpenStack services:

```
# openstack-ansible setup-openstack.yml
```

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 os-nova-install.yml --limit nova_compute
```

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

```
# openstack-ansible os-nova-install.yml --limit <node-name> \
    --skip-tags 'nova-key'
```

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 `inventory-manage.py` 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:

```
# ./scripts/inventory-manage.py -G
```

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

```
# ./scripts/inventory-manage.py -g
```

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

1. Change directory to the repository clone playbooks directory:

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

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

```
# openstack-ansible os-nova-install.yml --limit nova_compute \
                                        --list-hosts
```

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

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

1.1.9 Major upgrades

This guide provides information about the upgrade process from Stein to Train for OpenStack-Ansible.

Note: You can only upgrade between sequential releases.

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: The upgrade is always under active development. Test this on a development environment first.

Upgrading by using a script

The Train release series of OpenStack-Ansible contains the code for migrating from Stein to Train.

Running the upgrade script

To upgrade from Stein to Train 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 20.2.7
# ./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 Train release

Ensure that your OpenStack-Ansible code is on the latest Train tagged release.

```
# git checkout 20.2.7
```

Prepare the shell variables

Define these variables to reduce typing when running the remaining upgrade tasks. Because these environments variables are shortcuts, this step is optional. If you prefer, you can reference the files directly during the upgrade.

```
# cd /opt/openstack-ansible
# export MAIN_PATH="$(pwd)"
# export SCRIPTS_PATH="${MAIN_PATH}/scripts"
```

Backup the existing OpenStack-Ansible configuration

Make a backup of the configuration of the environment:

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

Bootstrap the new Ansible and OSA 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 Train release.

```
# ${SCRIPTS_PATH}/bootstrap-ansible.sh
```

Change to the playbooks directory

Change to the playbooks directory to simplify the CLI commands from here on in the procedure, given that most playbooks executed are in this directory.

```
# cd playbooks
```

Implement changes to OSA configuration

If there have been any OSA 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 "${SCRIPTS_PATH}/upgrade-utilities/deploy-config-changes.yml" -e
↪'placement_migrate_flag=true'
```

Upgrade hosts

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

```
# openstack-ansible setup-hosts.yml --limit '!galera_all:!rabbitmq_all'
```

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 setup-hosts.yml -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, and to handle the transition from the nova placement service to the extracted placement service, we pass the appropriate flags.

```
# openstack-ansible setup-infrastructure.yml -e 'galera_upgrade=true' -e 'rabbitmq_
↪upgrade=true' -e 'placement_migrate_flag=true'
```

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 "${SCRIPTS_PATH}/upgrade-utilities/galera-cluster-rolling-restart.
↪yml"
```

During upgrade from S->T nova may fail with DB migrations because of the insufficient size of the table type.

```
# openstack-ansible "${SCRIPTS_PATH}/upgrade-utilities/galera-row-format-switch.yml"
```

Upgrade OpenStack

We can now go ahead with the upgrade of all the OpenStack components, passing the flag that enabled the transition from the nova placement service to the extracted placement service.

```
# openstack-ansible setup-openstack.yml -e 'placement_migrate_flag=true'
```

Remove legacy nova placement service backends

Now that the new extracted placement service is operational, we can remove the legacy implementation from the load balancer.

```
# openstack-ansible haproxy-install.yml
```

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 see the Operations Guide.

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

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

1.2.1 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
# apt-get update
# apt-get dist-upgrade
# reboot
```

```
## CentOS
# yum upgrade
# yum install git
# 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/RHEL due to a lack of maintainers for the feature.

```
## openSUSE
# zypper up
# zypper in git-core
# reboot
```

Note: If you are installing with limited connectivity, please review the *Installing with limited connectivity* appendix in the Deployment Guide 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 stable/train
# git describe --abbrev=0 --tags

# # Checkout the latest tag from either method of retrieving the tag.
# git checkout 20.2.7
```

Note: The Train release is only compatible with Debian 9 (stretch), Debian 10 (buster), Ubuntu 18.04 (Bionic Beaver), CentOS 7 and openSUSE Leap 15.X.

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"
```

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 and nspawn 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'
# scripts/bootstrap-aio.sh
```

Note: If the metal and aio key words are used together, horizon will not be deployed because haproxy and horizon will conflict on the same listening ports.

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:

```
# cd /opt/openstack-ansible/playbooks
# openstack-ansible setup-hosts.yml
# openstack-ansible setup-infrastructure.yml
# openstack-ansible setup-openstack.yml
```

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
```

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 </admin/maintenance-tasks.html#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 \
```

(continues on next page)

(continued from previous page)

```
        | 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}

# # 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

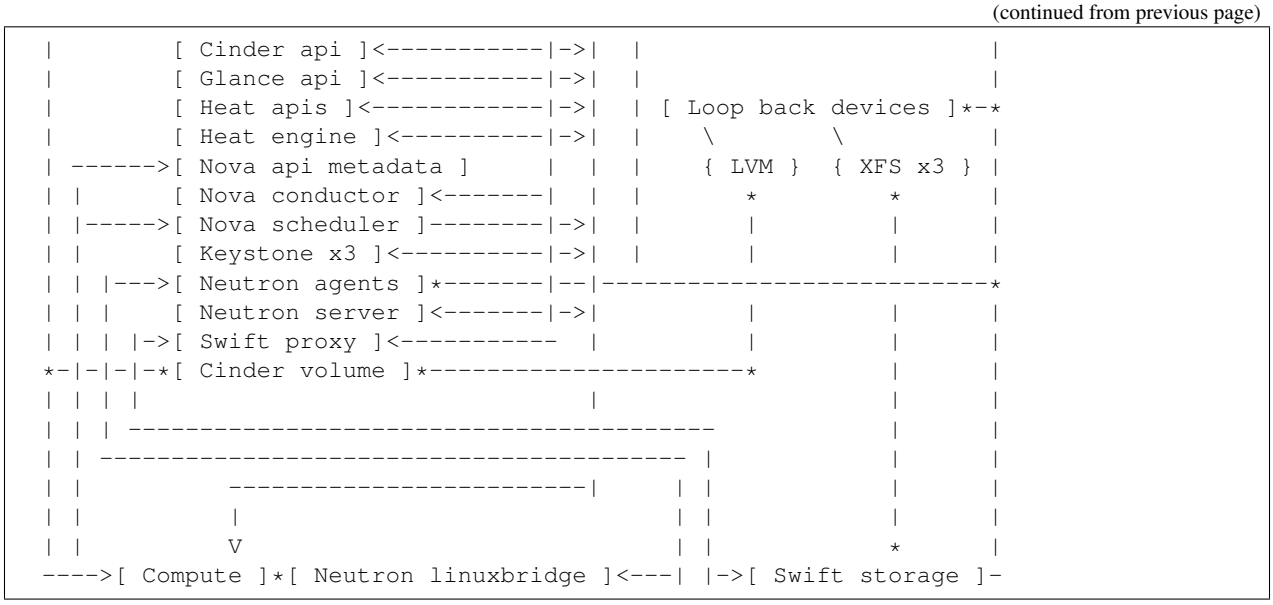
Here is a basic diagram that attempts to illustrate what the resulting AIO deployment looks like.

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.

```
          ------->[ ETH0 == Public Network ]
          |
          V                          [  *   ] Socket Connections
[ HOST MACHINE ]                     [ <>v^ ] Network Connections
  *       ^  *
  |       |  |-------------------------------------------------------
  |       |                                                          |
  |       |---------------->[ HAProxy ]                              |
  |                            ^                                     |
  |                            |                                     |
  |                            V                                     |
  |                        (BR-Interfaces)<------                    |
  |                                ^     *      |                   |
  *-[ LXC ]*--*-----------------------|-----|------|----|            |
  |           |                       |     |      |  | |            |
  |           |                       |     |      |  | |            |
  |           |                       |     |      |  | |            |
  |           |                       |     |      V  * |            |
  |           *                       |     |   [ Galera x3 ]        |
  |        [ Memcached ]<-------------|     |           |            |
  *-------*[ Rsyslog ]<--------------|--|  |           *            |
  |        [ Repos Server x3 ]<------|  ---|-->[ RabbitMQ x3 ]      |
  |        [ Horizon x2 ]<-----------|  |  |                        |
  |        [ Nova api ec2 ]<---------|--|  |                        |
  |        [ Nova api os ]<----------|->|  |                        |
  |        [ Nova console ]<---------|  |  |                        |
  |        [ Nova Cert ]<------------|->|  |                        |
```

(continues on next page)

(continued from previous page)



1.2.2 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/tenant networks for instances

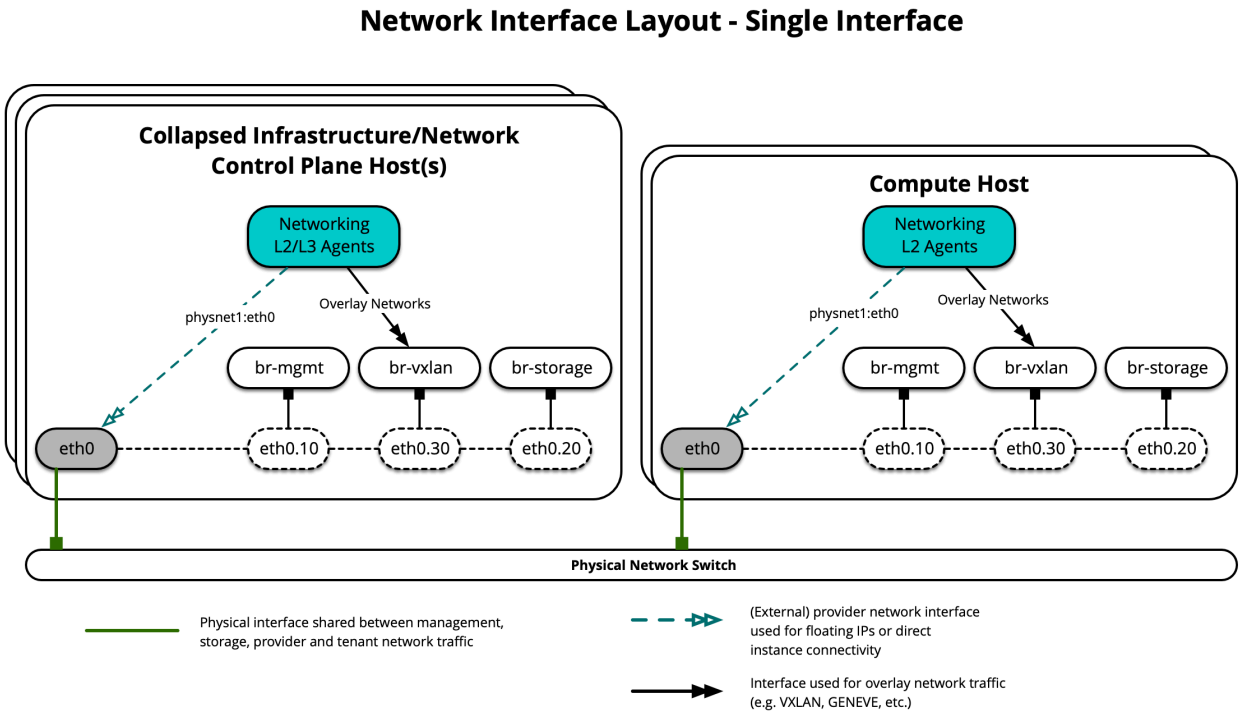
- Other OpenStack services

Network interfaces

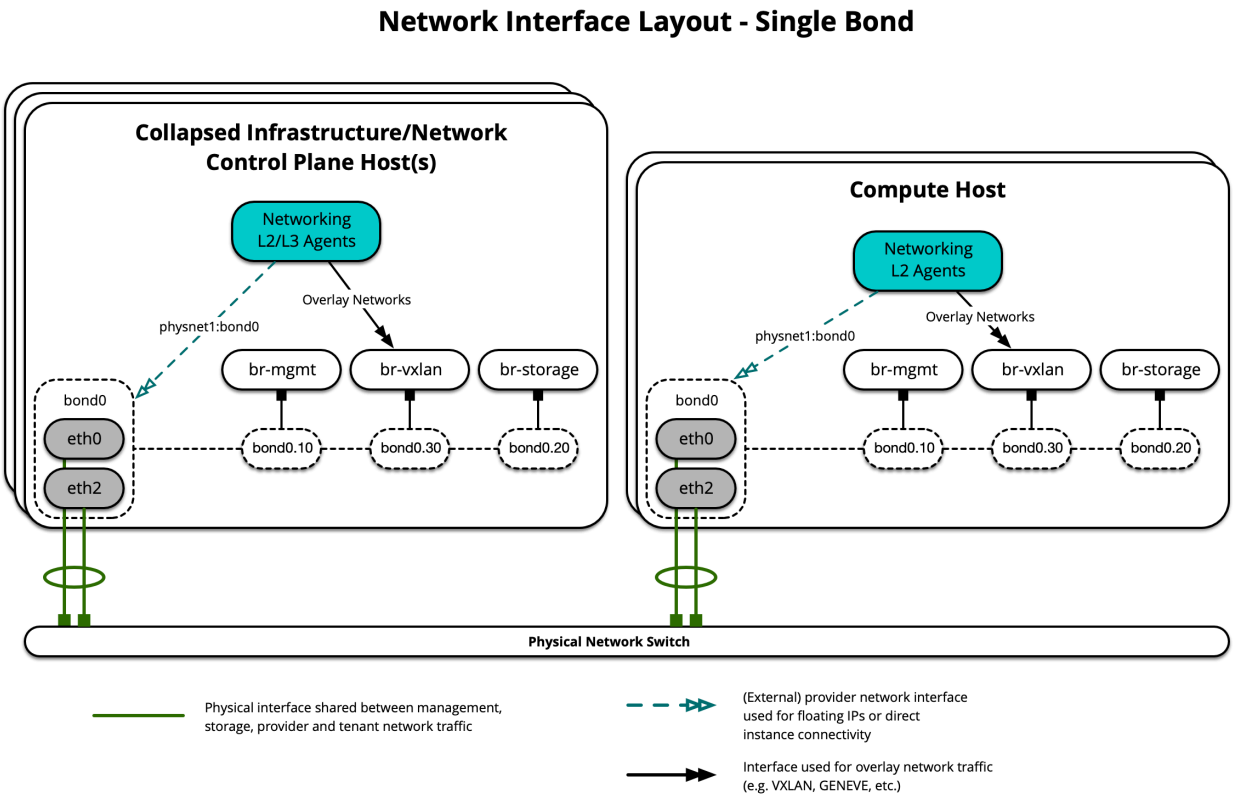
Single interface or bond

OpenStack-Ansible supports the use of a single interface or set of bonded interfaces that carry traffic for OpenStack services as well as instances.

The following diagram demonstrates hosts using a single interface:



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
```

(continues on next page)

(continued from previous page)

```
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
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
```

(continues on next page)

(continued from previous page)

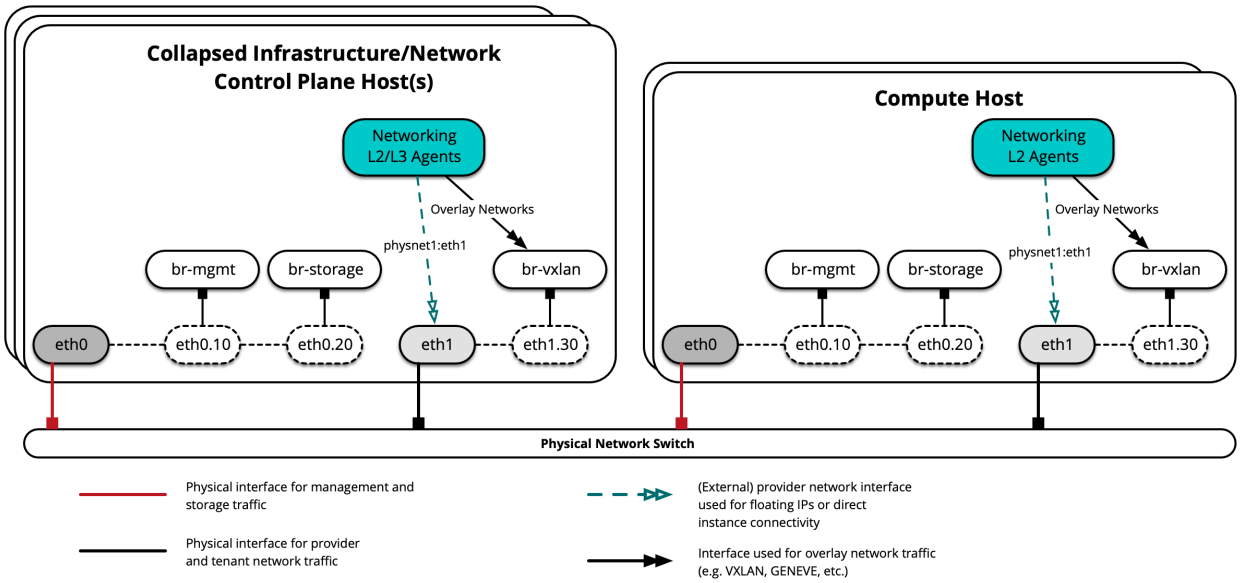
```
    netmask 255.255.252.0

# 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

# 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.

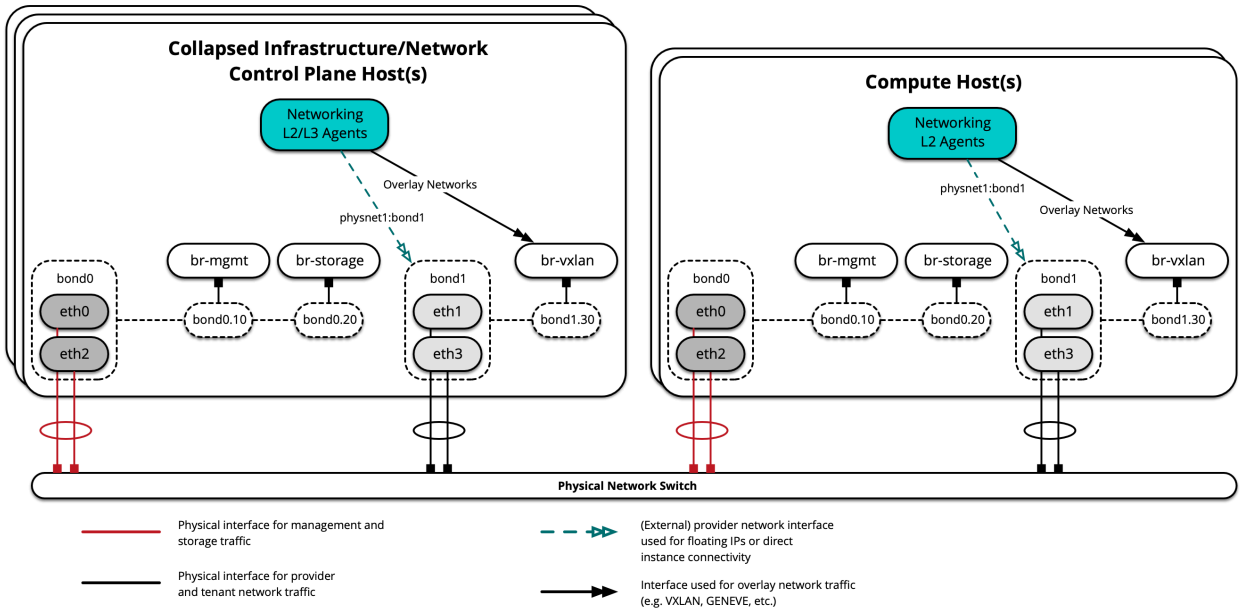
The following diagram demonstrates hosts using multiple interfaces:

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
# 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

# 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
```

(continues on next page)

(continued from previous page)

```
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.
#
#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
#
```

(continues on next page)

(continued from previous page)

```
# 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:

- Configuring a test environment
- Configuring a homogeneous production environment
- Using provider network groups for a heterogeneous environment

For network agent and container networking toplogies, please refer to the following:

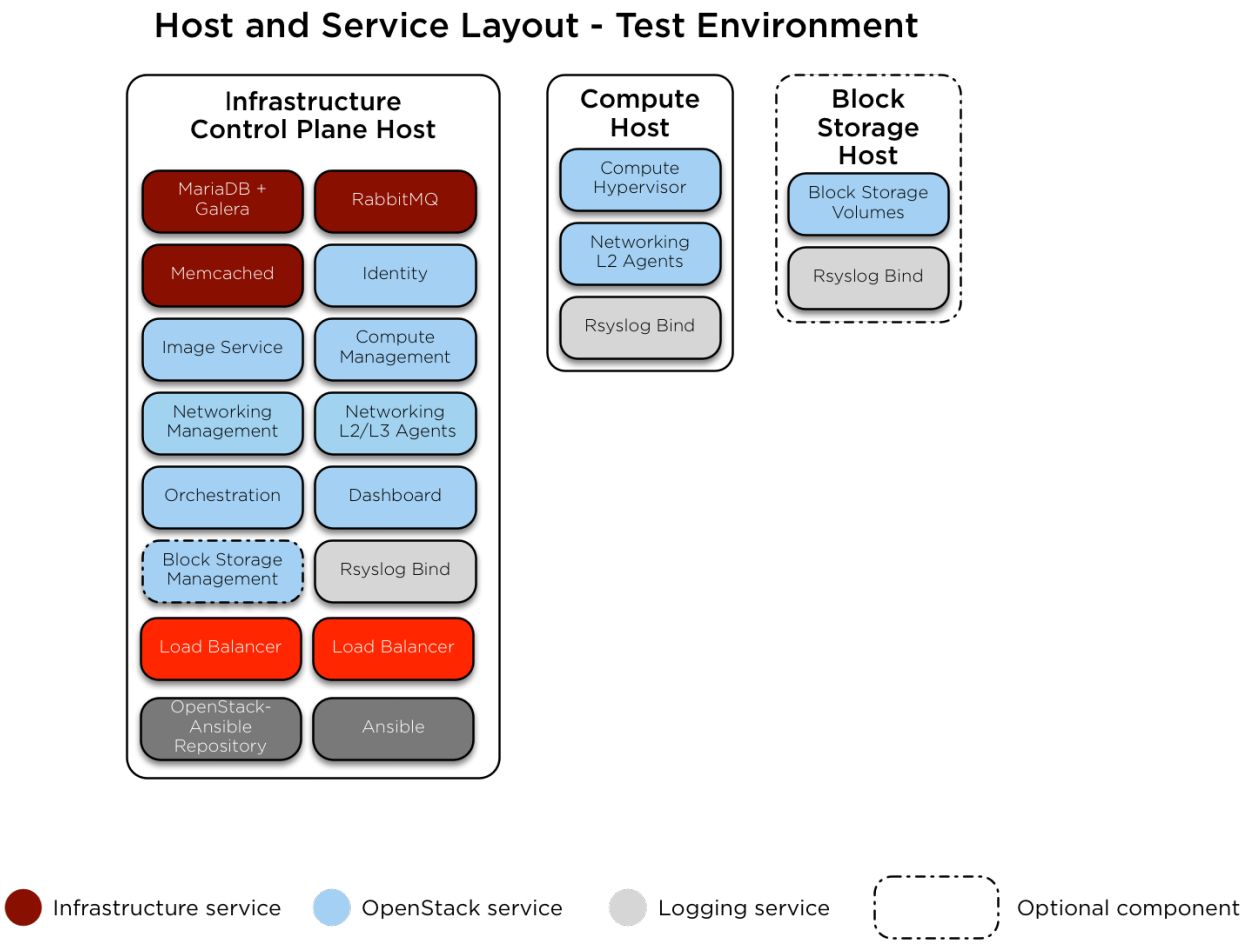
- Container networking architecture

1.2.3 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:

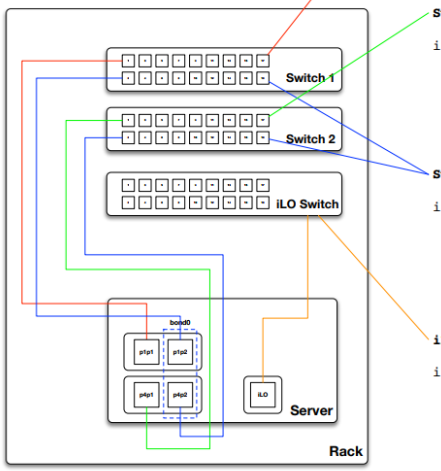
- One infrastructure (control plane) host (8 vCPU, 8 GB RAM, 60 GB HDD)
- One compute host (8 vCPU, 8 GB RAM, 60 GB HDD)
- One Network Interface Card (NIC) for each host
- A basic compute kit environment, with the Image (glance) and Compute (nova) services set to use file-backed storage.
- Internet access 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
# 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
```

(continues on next page)

(continued from previous page)

```
# 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:
  container: 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
  # 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: "container"
        type: "raw"
        group_binds:
          - all_containers
```

(continues on next page)

(continued from previous page)

```
        - hosts
      is_container_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_linuxbridge_agent
  - network:
      container_bridge: "br-vlan"
      container_type: "veth"
      container_interface: "eth12"
      host_bind_override: "eth12"
      type: "flat"
      net_name: "flat"
      group_binds:
        - neutron_linuxbridge_agent
  - network:
      container_bridge: "br-vlan"
      container_type: "veth"
      container_interface: "eth11"
      type: "vlan"
      range: "101:200,301:400"
      net_name: "vlan"
      group_binds:
        - neutron_linuxbridge_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
        - 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
```

(continues on next page)

(continued from previous page)

```
haproxy_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

# 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
```

(continues on next page)

(continued from previous page)

```
    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:

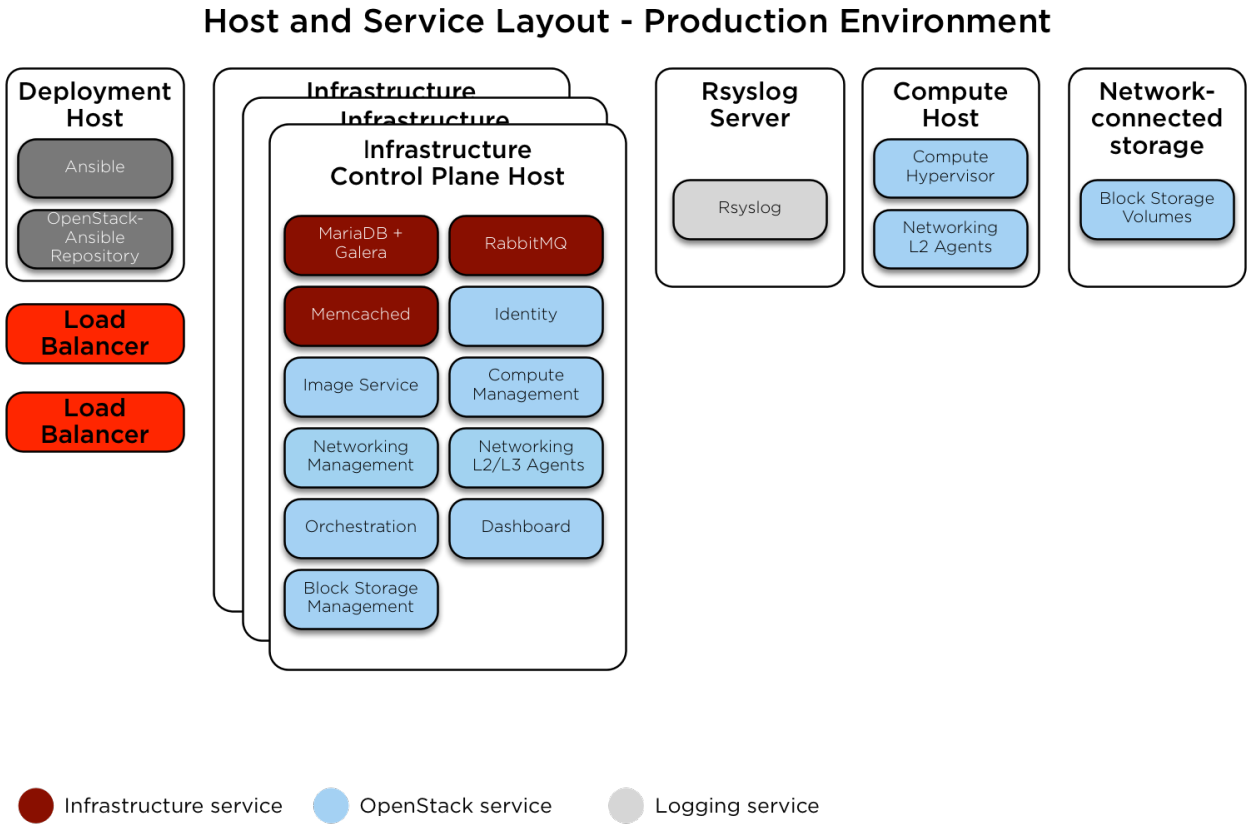
```
---
# This file contains an example of the global variable overrides
# which may need to be set for a production environment.

## OpenStack public endpoint protocol
openstack_service_publicuri_proto: http
```

1.2.4 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

# 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
```

(continues on next page)

(continued from previous page)

```
    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
    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
```

(continues on next page)

(continued from previous page)

```
#    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
    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:
  container: 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: "container"
        type: "raw"
        group_binds:
          - all_containers
          - hosts
        is_container_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_linuxbridge_agent
    - network:
        container_bridge: "br-vlan"
        container_type: "veth"
        container_interface: "eth12"
        host_bind_override: "eth12"
        type: "flat"
        net_name: "flat"
        group_binds:
          - neutron_linuxbridge_agent
    - network:
        container_bridge: "br-vlan"
        container_type: "veth"
        container_interface: "eth11"
        type: "vlan"
```

(continues on next page)

(continued from previous page)

```
        range: "101:200,301:400"
        net_name: "vlan"
        group_binds:
          - neutron_linuxbridge_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
          - nova_compute

###
### Infrastructure
###

# galera, memcache, rabbitmq, utility
shared-infra_hosts:
  infra1:
    ip: 172.29.236.11
    container_vars:
      # Optional | Example setting the container_tech for a target host.
      container_tech: lxc
  infra2:
    ip: 172.29.236.12
    container_vars:
      # Optional | Example setting the container_tech for a target host.
      container_tech: nspawn
  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.
haproxy_hosts:
  infra1:
    ip: 172.29.236.11
  infra2:
    ip: 172.29.236.12
  infra3:
    ip: 172.29.236.13

# rsyslog server
log_hosts:
```

(continues on next page)

(continued from previous page)

```
  log1:
    ip: 172.29.236.14

###
### 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
# each container could have different storage targets.
image_hosts:
  infra1:
    ip: 172.29.236.11
    container_vars:
      limit_container_types: glance
      glance_nfs_client:
        - server: "172.29.244.15"
          remote_path: "/images"
          local_path: "/var/lib/glance/images"
          type: "nfs"
          options: "_netdev,auto"
  infra2:
    ip: 172.29.236.12
    container_vars:
      limit_container_types: glance
      glance_nfs_client:
        - server: "172.29.244.15"
          remote_path: "/images"
          local_path: "/var/lib/glance/images"
          type: "nfs"
          options: "_netdev,auto"
  infra3:
    ip: 172.29.236.13
    container_vars:
      limit_container_types: glance
      glance_nfs_client:
        - server: "172.29.244.15"
```

(continues on next page)

(continued from previous page)

```
          remote_path: "/images"
          local_path: "/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
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:
```

(continues on next page)

(continued from previous page)

```
    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)
# 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
```

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.

## Load Balancer Configuration (haproxy/keepalived)
haproxy_keepalived_external_vip_cidr: "<external_vip_address>/<netmask>"
haproxy_keepalived_internal_vip_cidr: "172.29.236.0/22"
haproxy_keepalived_external_interface: ens2
haproxy_keepalived_internal_interface: br-mgmt
```

1.2.5 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 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

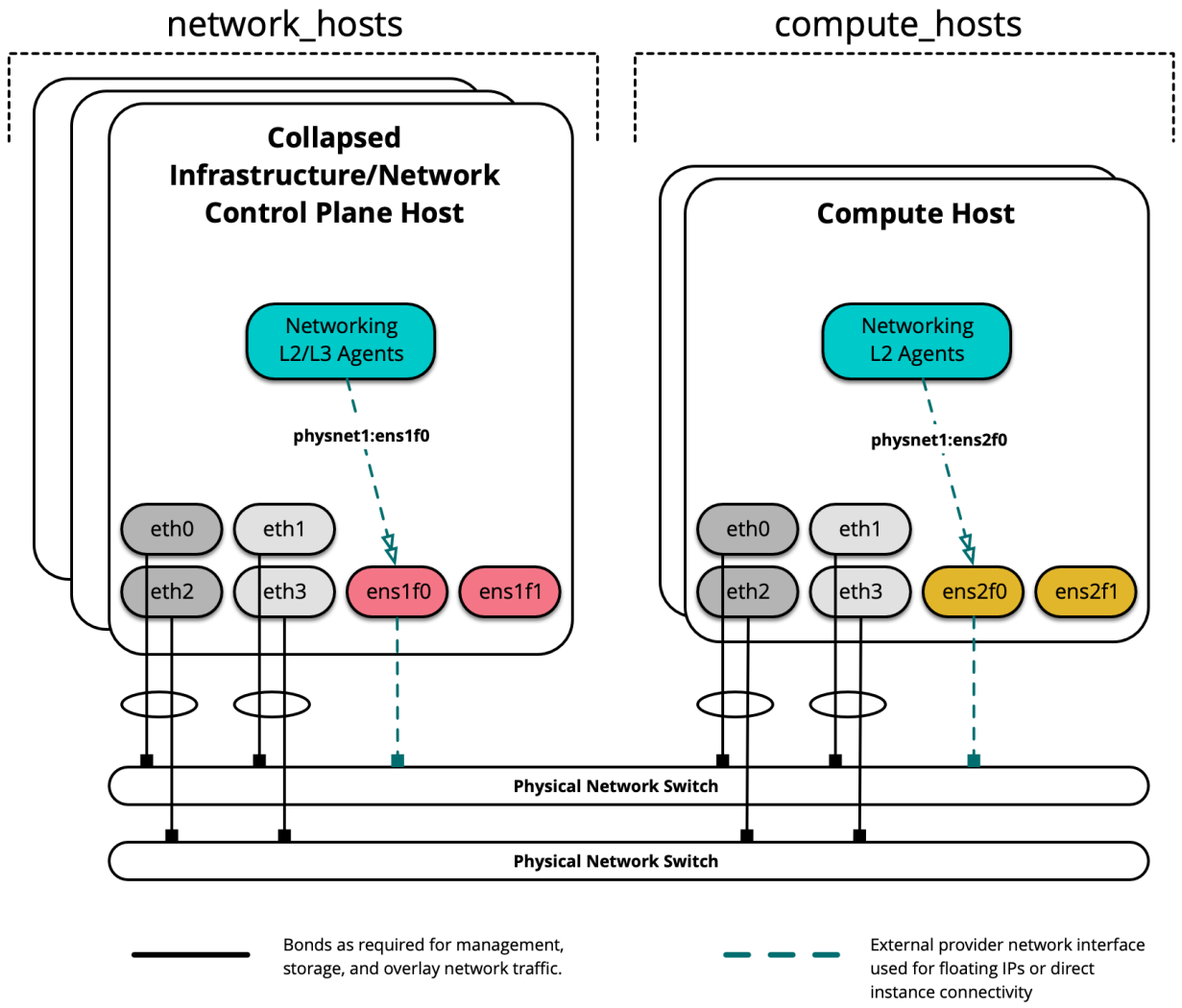
This example environment has the following characteristics:

- 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:
```

(continues on next page)

(continued from previous page)

```
  container: 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: "container"
        type: "raw"
        group_binds:
          - all_containers
          - hosts
        is_container_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.
    #
    # 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
```

(continues on next page)

(continued from previous page)

```
    # 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: "physnet1"
        group_binds:
          - network_hosts
    #
    # 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"
```

(continues on next page)

(continued from previous page)

```
        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
    container_vars:
      # Optional | Example setting the container_tech for a target host.
      container_tech: lxc
  infra2:
    ip: 172.29.236.12
    container_vars:
      # Optional | Example setting the container_tech for a target host.
      container_tech: nspawn
  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.
haproxy_hosts:
  infra1:
    ip: 172.29.236.11
  infra2:
    ip: 172.29.236.12
  infra3:
    ip: 172.29.236.13

# rsyslog server
log_hosts:
  log1:
    ip: 172.29.236.14

###
### OpenStack
###
```

(continues on next page)

(continued from previous page)

```
# 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
# each container could have different storage targets.
image_hosts:
  infra1:
    ip: 172.29.236.11
    container_vars:
      limit_container_types: glance
      glance_nfs_client:
        - server: "172.29.244.15"
          remote_path: "/images"
          local_path: "/var/lib/glance/images"
          type: "nfs"
          options: "_netdev,auto"
  infra2:
    ip: 172.29.236.12
    container_vars:
      limit_container_types: glance
      glance_nfs_client:
        - server: "172.29.244.15"
          remote_path: "/images"
          local_path: "/var/lib/glance/images"
          type: "nfs"
          options: "_netdev,auto"
  infra3:
    ip: 172.29.236.13
    container_vars:
      limit_container_types: glance
      glance_nfs_client:
        - server: "172.29.244.15"
          remote_path: "/images"
          local_path: "/var/lib/glance/images"
          type: "nfs"
          options: "_netdev,auto"

# placement
```

(continues on next page)

(continued from previous page)

```
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
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
```

(continues on next page)

(continued from previous page)

```
  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)
# 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
```

(continues on next page)

(continued from previous page)

```
          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"
```

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.

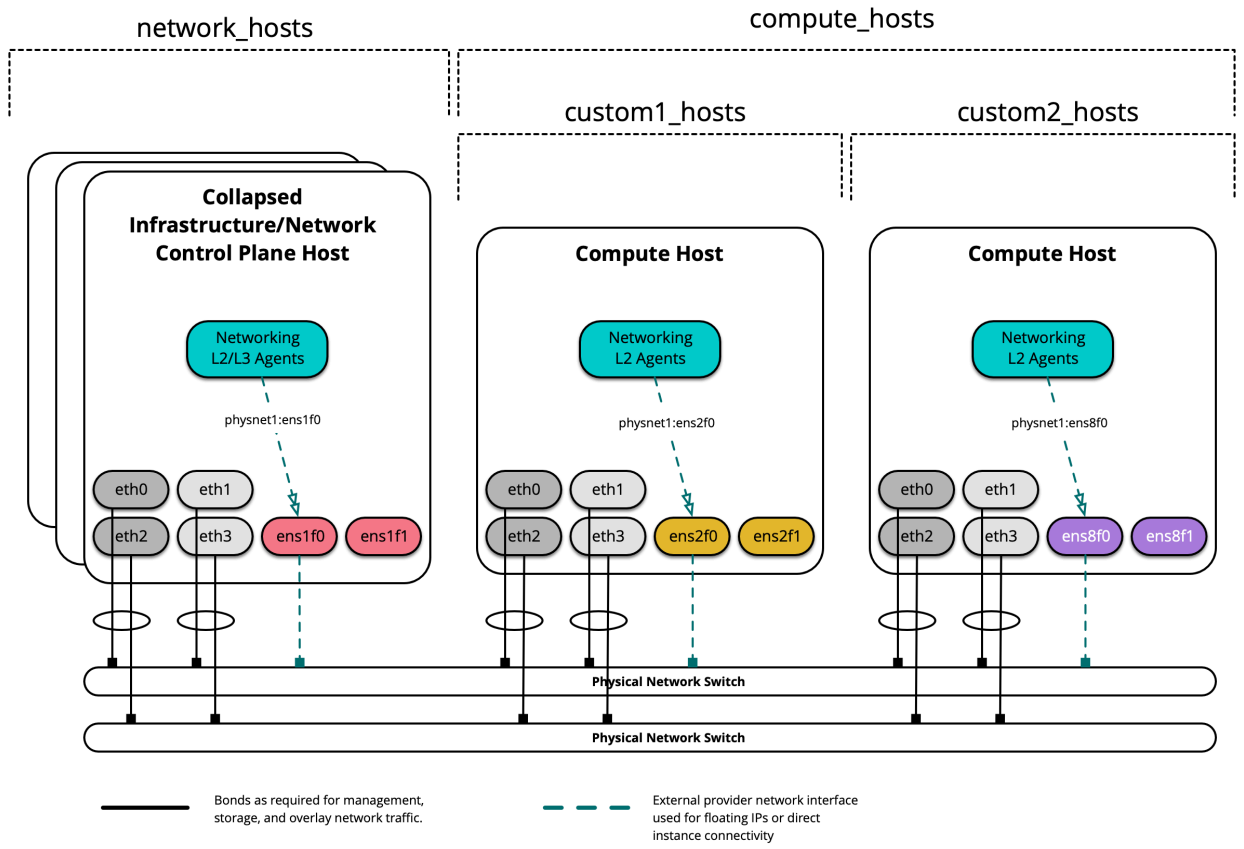
Custom Groups

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`:

```
---
# 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
```

1.2.6 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 good quality Internet connectivity 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

We recommend a set of practices and configuration overrides deployers can use when running OpenStack-Ansible in network environments that block Internet connectivity.

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 build to use an alternative index, create the file */etc/pip.conf* with the following content and ensure that it is placed on all hosts in the environment.

```
[global]
index-url = http://pip.example.org/simple
```

Then, in */etc/openstack_deploy/user_variables.yml*, inform the deployment that it needs to copy that file 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
```

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 18.04 LTS:

- bionic
- bionic-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/debian-luminous/
- https://www.rabbitmq.com/debian
- http://ubuntu-cloud.archive.canonical.com/ubuntu
- https://packages.erlang-solutions.com/ubuntu
- https://mirror.rackspace.com/mariadb/repo/10.1/ubuntu
- https://repo.percona.com/apt

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 relies upon community built LXC images when building containers for OpenStack services. Deployers may choose to create, maintain, and host their own container images. Consult the `openstack-ansible-lxc_container_create` 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`.

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.

Configuration can be applied to target and deployment hosts 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

Further 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, without going 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:

#. 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.

#. 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:

#. Persistent proxy configuration is a standard practice and network clients on the target hosts will be able to access external resources after deployment.

#. 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`.

#. 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.

#. 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. It is then 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. Find an example `/etc/openstack_deploy/user_variables.yml` configuration below:

```
pip_validate_certs: false
galera_package_download_validate_certs: false
```

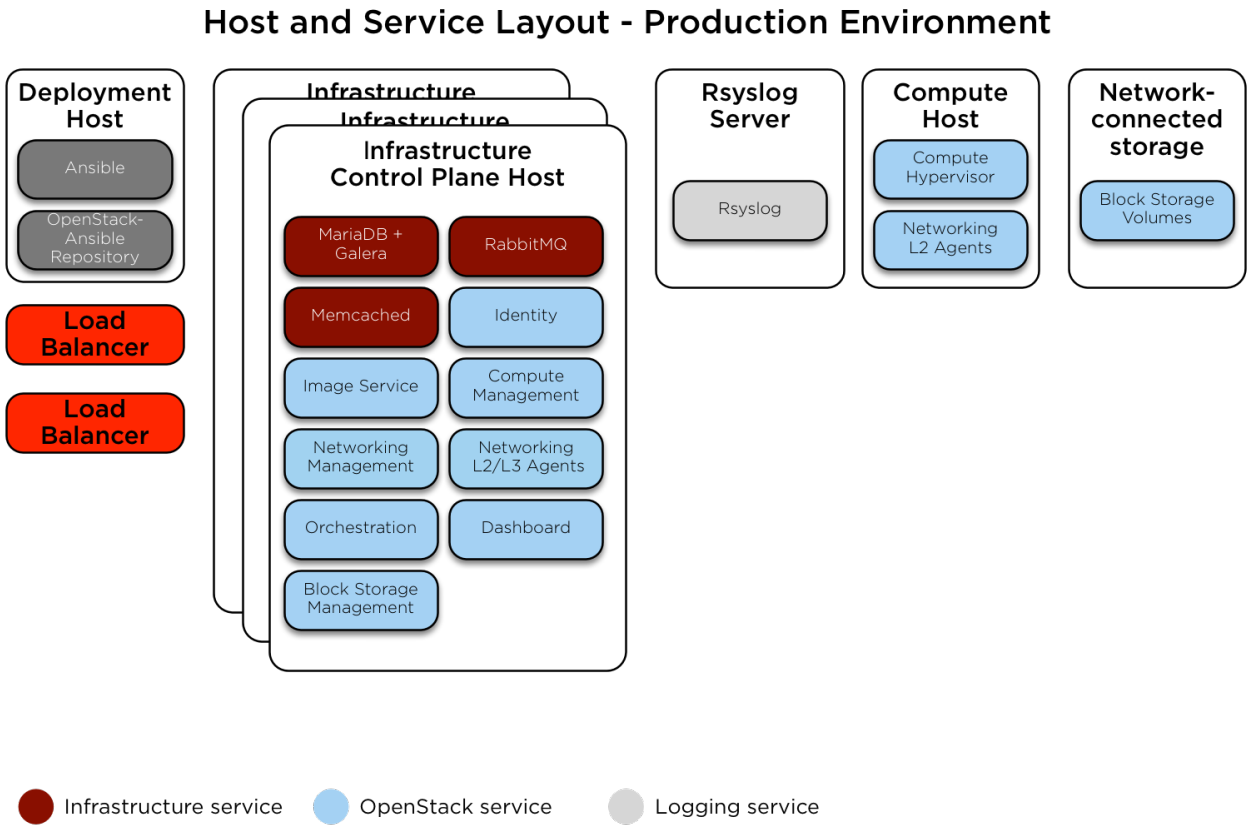
The list above is intentionally not exhaustive. Additional variables may exist within the project and will be named using the **_validate_certs* pattern. Disable certificate chain validation on a case by case basis and only after encountering failures that are known to only be caused by the proxy server(s).

1.2.7 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
```

(continues on next page)

(continued from previous page)

```
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

# 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
```

(continues on next page)

(continued from previous page)

```
    bridge_waitport 0
    bridge_fd 0
    bridge_ports bond0.10
    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
#    post-down ip link del br-vlan-veth || true
#    bridge_ports bond1 br-vlan-veth

# Storage bridge (optional)
#
```

(continues on next page)

(continued from previous page)

```
# 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
  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"
```

(continues on next page)

(continued from previous page)

```
  - "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:
  internal_lb_vip_address: internal-openstack.example.com
  #
  # 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: "pod1_container"
        address_prefix: "container"
        type: "raw"
        group_binds:
          - all_containers
          - hosts
        reference_group: "pod1_hosts"
        is_container_address: true
        # Containers in pod1 need routes to the container networks of other pods
        static_routes:
          # 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: "container"
        type: "raw"
        group_binds:
          - all_containers
          - hosts
        reference_group: "pod2_hosts"
        is_container_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
```

(continues on next page)

(continued from previous page)

```
    - network:
        container_bridge: "br-mgmt"
        container_type: "veth"
        container_interface: "eth1"
        ip_from_q: "pod3_container"
        address_prefix: "container"
        type: "raw"
        group_binds:
          - all_containers
          - hosts
        reference_group: "pod3_hosts"
        is_container_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: "container"
        type: "raw"
        group_binds:
          - all_containers
          - hosts
        reference_group: "pod4_hosts"
        is_container_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_linuxbridge_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"
```

(continues on next page)

(continued from previous page)

```
      type: "vxlan"
      range: "1:1000"
      net_name: "vxlan"
      group_binds:
        - neutron_linuxbridge_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"
      range: "1:1000"
      net_name: "vxlan"
      group_binds:
        - neutron_linuxbridge_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_linuxbridge_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: "flat"
      group_binds:
        - neutron_linuxbridge_agent
  - network:
      container_bridge: "br-vlan"
      container_type: "veth"
```

(continues on next page)

(continued from previous page)

```
      container_interface: "eth11"
      type: "vlan"
      range: "101:200,301:400"
      net_name: "vlan"
      group_binds:
        - neutron_linuxbridge_agent
  - network:
      container_bridge: "br-storage"
      container_type: "veth"
      container_interface: "eth2"
      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:
```

(continues on next page)

(continued from previous page)

```
          # Route to storage networks
          - cidr: 172.29.244.0/22
            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:
  infra1:
    ip: 172.29.236.10
  log1:
    ip: 172.29.236.11

pod2_hosts:
  infra2:
    ip: 172.29.239.10

pod3_hosts:
  infra3:
    ip: 172.29.242.10

pod4_hosts:
  compute1:
    ip: 172.29.245.10
  compute2:
    ip: 172.29.245.11

# galera, memcache, rabbitmq, utility
shared-infra_hosts:
  infra1:
    ip: 172.29.236.10
  infra2:
    ip: 172.29.239.10
  infra3:
    ip: 172.29.242.10

# repository (apt cache, python packages, etc)
repo-infra_hosts:
```

(continues on next page)

(continued from previous page)

```
  infra1:
    ip: 172.29.236.10
  infra2:
    ip: 172.29.239.10
  infra3:
    ip: 172.29.242.10

# load balancer
# Ideally the load balancer should not use the Infrastructure hosts.
# Dedicated hardware is best for improved performance and security.
haproxy_hosts:
  infra1:
    ip: 172.29.236.10
  infra2:
    ip: 172.29.239.10
  infra3:
    ip: 172.29.242.10

# rsyslog server
log_hosts:
  log1:
    ip: 172.29.236.11

###
### OpenStack
###

# keystone
identity_hosts:
  infra1:
    ip: 172.29.236.10
  infra2:
    ip: 172.29.239.10
  infra3:
    ip: 172.29.242.10

# cinder api services
storage-infra_hosts:
  infra1:
    ip: 172.29.236.10
  infra2:
    ip: 172.29.239.10
  infra3:
    ip: 172.29.242.10

# 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_nfs_client:
        - server: "172.29.244.15"
```

(continues on next page)

(continued from previous page)

```
          remote_path: "/images"
          local_path: "/var/lib/glance/images"
          type: "nfs"
          options: "_netdev,auto"
  infra2:
    ip: 172.29.236.12
    container_vars:
      limit_container_types: glance
      glance_nfs_client:
        - server: "172.29.244.15"
          remote_path: "/images"
          local_path: "/var/lib/glance/images"
          type: "nfs"
          options: "_netdev,auto"
  infra3:
    ip: 172.29.236.13
    container_vars:
      limit_container_types: glance
      glance_nfs_client:
        - server: "172.29.244.15"
          remote_path: "/images"
          local_path: "/var/lib/glance/images"
          type: "nfs"
          options: "_netdev,auto"

# nova api, conductor, etc services
compute-infra_hosts:
  infra1:
    ip: 172.29.236.10
  infra2:
    ip: 172.29.239.10
  infra3:
    ip: 172.29.242.10

# heat
orchestration_hosts:
  infra1:
    ip: 172.29.236.10
  infra2:
    ip: 172.29.239.10
  infra3:
    ip: 172.29.242.10

# horizon
dashboard_hosts:
  infra1:
    ip: 172.29.236.10
  infra2:
    ip: 172.29.239.10
  infra3:
    ip: 172.29.242.10

# neutron server, agents (L3, etc)
network_hosts:
  infra1:
    ip: 172.29.236.10
  infra2:
```

(continues on next page)

(continued from previous page)

```
    ip: 172.29.239.10
  infra3:
    ip: 172.29.242.10

# ceilometer (telemetry data collection)
metering-infra_hosts:
  infra1:
    ip: 172.29.236.10
  infra2:
    ip: 172.29.239.10
  infra3:
    ip: 172.29.242.10

# aodh (telemetry alarm service)
metering-alarm_hosts:
  infra1:
    ip: 172.29.236.10
  infra2:
    ip: 172.29.239.10
  infra3:
    ip: 172.29.242.10

# gnocchi (telemetry metrics storage)
metrics_hosts:
  infra1:
    ip: 172.29.236.10
  infra2:
    ip: 172.29.239.10
  infra3:
    ip: 172.29.242.10

# nova hypervisors
compute_hosts:
  compute1:
    ip: 172.29.245.10
  compute2:
    ip: 172.29.245.11

# ceilometer compute agent (telemetry data collection)
metering-compute_hosts:
  compute1:
    ip: 172.29.245.10
  compute2:
    ip: 172.29.245.11

# 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:
```

(continues on next page)

(continued from previous page)

```
          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.

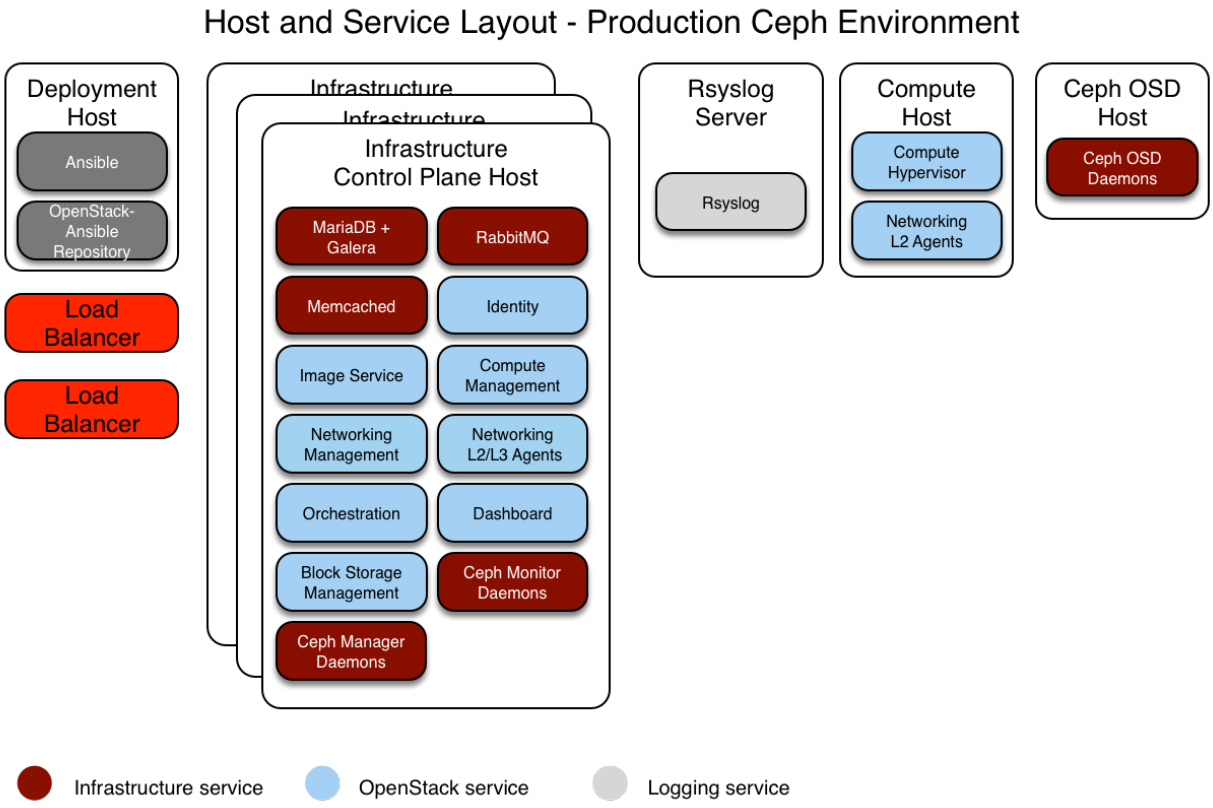
```
---
# This file contains an example of the global variable overrides
# which may need to be set for a production environment.

## Load Balancer Configuration (haproxy/keepalived)
haproxy_keepalived_external_vip_cidr: "<external_vip_address>/<netmask>"
haproxy_keepalived_internal_vip_cidr: "172.29.236.0/22"
haproxy_keepalived_external_interface: ens2
haproxy_keepalived_internal_interface: br-mgmt
```

1.2.8 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 two ways:

- connecting to your own ceph cluster by pointing to its information in `user_variables.yml`
- deploying a ceph cluster by using the roles maintained by the Ceph-Ansible project. Deployers can enable the `ceph-install` playbook by adding hosts to the `ceph-mon_hosts`, `ceph-osd_hosts` and `ceph-rgw_hosts` groups in `openstack_user_config.yml`, and then configuring Ceph-Ansible specific vars in the OpenStack-Ansible `user_variables.yml` file.

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
```

(continues on next page)

(continued from previous page)

```
    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
    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
```

(continues on next page)

(continued from previous page)

```
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
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
    bridge_ports bond0.20

# compute1 Storage bridge
#auto br-storage
```

(continues on next page)

(continued from previous page)

```
#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
  container: 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: "container"
        type: "raw"
        group_binds:
          - all_containers
          - hosts
        is_container_address: true
    - network:
        container_bridge: "br-vxlan"
        container_type: "veth"
        container_interface: "eth10"
        ip_from_q: "tunnel"
```

(continues on next page)

(continued from previous page)

```
        type: "vxlan"
        range: "1:1000"
        net_name: "vxlan"
        group_binds:
          - neutron_linuxbridge_agent
    - network:
        container_bridge: "br-vlan"
        container_type: "veth"
        container_interface: "eth12"
        host_bind_override: "eth12"
        type: "flat"
        net_name: "flat"
        group_binds:
          - neutron_linuxbridge_agent
    - network:
        container_bridge: "br-vlan"
        container_type: "veth"
        container_interface: "eth11"
        type: "vlan"
        range: "101:200,301:400"
        net_name: "vlan"
        group_binds:
          - neutron_linuxbridge_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
          - nova_compute
          - ceph-osd

###
### Infrastructure
###

_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:
```

(continues on next page)

(continued from previous page)

```
    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

# ceph-mon containers
ceph-mon_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.
haproxy_hosts: *infrastructure_hosts

# rsyslog server
log_hosts:
  log1:
    ip: 172.29.236.14

###
### 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)
```

(continues on next page)

(continued from previous page)

```
metering-infra_hosts: *infrastructure_hosts

# aodh (telemetry alarm service)
metering-alarm_hosts: *infrastructure_hosts

# gnocchi (telemetry metrics storage)
metrics_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.
## Load Balancer Configuration (haproxy/keepalived)
haproxy_keepalived_external_vip_cidr: "<external_ip_address>/<netmask>"
haproxy_keepalived_internal_vip_cidr: "172.29.236.0/22"
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()))'
```

(continues on next page)

(continued from previous page)

```
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.container }}"
public_network: "{{ cidr_networks.container }}"
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:
    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
```

1.2.9 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:

```
ceph_conf_overrides_rgw:
  "client.rgw.{{ hostvars[inventory_hostname]['ansible_hostname'] }}":
    # OpenStack integration with Keystone
    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_tenant: "{{ radosgw_admin_tenant }}"
```

(continues on next page)

(continued from previous page)

```
    rgw_keystone_admin_domain: default
    rgw_keystone_accepted_roles: 'member, _member_, admin, swiftoperator'
    rgw_keystone_implicit_tenants: 'true'
    rgw_swift_account_in_url: true
    rgw_swift_versioning_enabled: 'true'
    # Add S3 support, in addition to Swift
    rgw_enable_apis: 'swift, s3'
    rgw_s3_auth_use_keystone: 'true'
```

You may also want to add the `rgw_dns_name` option if you want to enable bucket hostnames with the S3 API.

1.2.10 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
    - gnocchiclient
    - libvirt-python
    - PyMySQL
    - pymongo
    - python-ceilometerclient
    - 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:
```

(continues on next page)

(continued from previous page)

```
    - "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
```

1.2.11 Security settings

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 most environments need are those for haproxy.

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.

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 highly secure environments. By default, self-signed certificates are used in OpenStack-Ansible. When self-signed certificates are used, certificate verification is automatically disabled.

Setting subject data for self-signed certificates

Change the subject data of any self-signed certificate by using configuration variables. The configuration variable for each service is formatted as `<servicename>_ssl_self_signed_subject`. For example, to change the SSL certificate subject data for HAProxy, adjust the `/etc/openstack_deploy/user_variables.yml` file as follows:

```
haproxy_ssl_self_signed_subject: "/C=US/ST=Texas/L=San Antonio/O=IT/CN=haproxy.
↪example.com"
```

For more information about the available fields in the certificate subject, see the OpenSSL documentation for the req subcommand.

Generating and regenerating self-signed certificates

Self-signed certificates are generated for each service during the first run of the playbook.

To generate a new self-signed certificate for a service, you must set the `<servicename>_ssl_self_signed_regen` 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 "horizon_ssl_self_signed_regen=true" os-horizon-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_ssl_self_signed_regen` variable to `true` in the `/etc/openstack_deploy/user_variables.yml` file:

  ```
  haproxy_ssl_self_signed_regen: true
  ```

Note: Regenerating self-signed certificates replaces the existing certificates whether they are self-signed or user-provided.

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.

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 security-hardening.yml

# Perform a quick audit by using Ansible's check mode
  openstack-ansible --check security-hardening.yml
```

For more information about the security configurations, see the security hardening role documentation.

1.2.12 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 `playbooks/defaults/repo_packages` folder.

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 `playbooks/defaults/repo_packages/openstack_services.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.

1.2.13 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]"
# 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]"
- 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 installation uses Ceph as 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
    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 %}
```

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
- python-dev
- libpq-dev
- python-rados
# additional package for python redis client
- python-redis
```

```
gnocchi_pip_packages:
- cryptography
- gnocchiclient
# this is what we want:
#  - "gnocchi[mysql,ceph,ceph_alternative_lib,redis]"
# 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]"
- keystonemiddleware
- python-memcached
- redis
```

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_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]['container_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]"
# 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]"
- keystonemiddleware
- python-memcached
- redis
# 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/test_results/telemetry_gnocchi_with_ceph/index.html

1.2.14 Hybrid messaging example

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 (e.g. rabbitmq and qdrouterd).

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
amqp	AMQP V1.0	qdrouterd	yes		direct
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 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.

```
# RPC
oslomsg_rpc_transport: "{{ (groups[qdrouterd_host_group] | length > 0) | ternary('amqp
↪', 'rabbit') }}"
oslomsg_rpc_port: "{{ (groups[qdrouterd_host_group] | length > 0) | ternary(qdrouterd_
↪port, rabbitmq_port) }}"
oslomsg_rpc_servers: "{{ (groups[qdrouterd_host_group] | length > 0) |␣
↪ternary(qdrouterd_servers, rabbitmq_servers) }}"
oslomsg_rpc_use_ssl: "{{ (groups[qdrouterd_host_group] | length > 0) |␣
↪ternary(qdrouterd_use_ssl, rabbitmq_use_ssl) }}"
oslomsg_rpc_host_group: "{{ (groups[qdrouterd_host_group] | length > 0) |␣
↪ternary(qdrouterd_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 }}"
```

Hybrid messaging deployment with qdrouterd server

In OSA, the deployment of disparate messaging backends is completely transparent to the OpenStack services. By managing the inventories for the messaging servers, a hybrid messaging configuration will be created. The instantiation of the qdrouterd server role in the qdrouterd_host_group will automatically setup the oslomsg_rpc* variables to use this messaging backend. No additional configuration is required. The result is that RPC services will communicate via the amqp (V1.0 protocol) driver and the Notify services will communicate via the rabbit driver. The separation and use of different messaging backends can provide increased scale and resiliency.

1.3 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 see the Operations Guide.

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

Contents:

1.3.1 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://github.com/openstack/openstack-ansible	Our main repository, used by deployers. Uses the other repositories.
The **OpenStack-Ansible roles** repositories	• https://github.com/openstack/openstack-ansible-os_nova • https://github.com/openstack/openstack-ansible-os_glance • https://github.com/openstack/ansible-role-systemd_mount • https://github.com/openstack/ansible-config_template • https://github.com/openstack/ansible-hardening •	Each role is in charge of deploying **exactly one** component of an OpenStack-Ansible deployment.
The **tests repository**	• https://github.com/openstack/openstack-ansible-tests	The tests repository is the location for common code used in the integrated repo and role repos tests. It allows us to not repeat ourselves: it is the location of common playbooks, common tasks and scripts.
The **specs** repository	• https://github.com/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://github.com/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/opendaylight/integration-packaging-ansible-opendaylight • 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

The OpenStack-Ansible community communicates a lot through IRC, in the #openstack-ansible channel, on freenode. This channel is logged, and its logs are published on http://eavesdrop.openstack.org/irclogs/%23openstack-ansible/.

Weekly meetings are held in our IRC channel. The schedule and logs can be found on http://eavesdrop.openstack.org/#OpenStack_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]**.

1.3.2 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.

1.3.3 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.

1.3.4 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 best practices 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

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. Running a playbook with `--tags <role>-config` is only possible if the target already ran the tags `<role>-install`.
- A tag including the word `upgrade` handles all the upgrade tasks.

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.
- `run_tests.sh`: A script for developers to execute all standard tests on a suitable host. This must be copied from the `openstack-ansible-tests` repository and will be automatically be overridden by our proposal bot should any change happen.
- `Vagrantfile`: A configuration file to allow a developer to easily create a test virtual machine using Vagrant. This must automatically execute `run_tests.sh`. This must be copied from the `openstack-ansible-tests` repository and will be automatically be overridden by our proposal bot should any change happen.
- `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.

1.3.5 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 `lxb_inventory`, `calico_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
```

Some roles have extra tests, like neutron, defined in `tox.ini`.

To run a functional test named calico, execute:

```
./run_tests.sh calico
```

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 `playbooks/defaults/repo_packages` 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. Be sure to run the `repo-build.yml` play later so that wheels for your packages will be included in the repository infrastructure.
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 environement. 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:

1.3.6 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.

Now, all the functions are separated, and included into a branch independent tooling, *osa_cli_releases*.

You can install the latest version of this tooling by running:

```
pip3 install -e git+https://github.com/evrardjp/osa-cli.git#egg=openstack_ansible_cli
pip3 install -e git+https://github.com/evrardjp/osa_cli_releases.git#egg=osa_cli_
↪releases
```

This tooling can then be called using `osa releases`. Each subcommand contains help by default.

Updating upstream SHAs

The dependencies for OpenStack-Ansible are updated through the use of `osa releases bump_upstream_shas`. This script updates the projects pinned SHAs, located in the *repo_packages folder*, based on their `_git_track_branch` value.

Updating OpenStack-Ansible roles

Updating the roles to their latest version per branch is done through `osa releases bump_roles $gitbranchname`.

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 `osa releases freeze_roles_for_milestone`

Check current pins status

You can check the current PyPI pins that are used in openstack-ansible repository by running `osa 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.

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.
  - Update `global-requirements-pins`
- 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.
  - 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`
- By milestone 3:
  - Implement features
  - Update `global-requirements-pins`
- After milestone 3:
  - Feature freeze, bug fixes, and testing improvements.
- After creating a new stable branch:
  - For all the repos, update the eventual static files refering to master/previous branch name. The main documentation should be updated to add the new branch. The #openstack-docs team should also be updated, for linking the newly created deployment-guide.

    Use the topic `create-<branchname>` (e.g: `create-stein`) for future reference.
  - 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.
- After official project release, before official OpenStack-Ansible release:
  - Bump RDO, Ubuntu Cloud Archive and openSUSE OBS OpenStack Cloud repositories if they are ready on time.
- Immediately after official OpenStack-Ansible release:
  - Send a thank you note to all the contributors through the mailing lists. They deserve it.

1.3.7 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 +2/-2 and +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.

1.3.8 Distribution support

Supported distributions

The list of supported distributions can be found in the Deployment Guide

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.

1.4 OpenStack-Ansible Reference

This chapter contains all the extra reference information 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 see the Operations Guide.

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

1.4.1 Releases

What is the OSA release model

OpenStack-Ansible (OSA) 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>.0.0.0b<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 (eg: Austin = 1, Bexar = 2, Newton = 14, Pike = 16, etc). OSA adopted the same `<MAJOR>` release numbering as the Nova project to match the overall OpenStack series version numbering.

How frequently does OSA 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 OSA?

For each OSA release, the OpenStack version that is deployed is set to a specific OpenStack *git SHA-1 hash_* (SHA). These are updated after every OSA release. The intent is to ensure that OSA 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 OSA deployment will include a version of a service (eg: nova-17.0.3dev4) which does not match a tag exactly as you may expect (eg: 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 OSA role get into a release?

For each OSA release, the Ansible roles that form that release are set to a specific *git SHA-1 hash_* (SHA). These are updated after every OSA release.

OSA frequently does pro-active bugfix backports. In order to reduce the risk of these backports introducing any destabilization, OSA 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 rule is for the `master` branch, which intentionally consumes the `master` branch from all roles between releases so that any changes are immediately integration tested.

1.4.2 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/bin/openstack-ansible`. It sources an environment variable file located in: `/usr/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 confer to the inventory section of this reference.

1.4.3 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`.

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/
```

This invocation is useful when testing changes to the dynamic inventory script.

Inputs

The `dynamic_inventory.py` 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.

Note: In all versions prior to Stein, this argument was `--file`.

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 `dynamic_inventory.py` 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.

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 files `/etc/openstack_deploy/openstack_inventory.json` or `/etc/openstack_deploy/openstack_hostnames_ips.yml`. This can lead to file corruptions, and problems with the inventory: hosts and container could disappear and new ones would appear, 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:
    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.

Deploying 0 (or more than one) of component type per host

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 *Deploying 0 (or more than one) of component type per host* 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.

Deploying using a different container technology

Note: While nspawn is an available containerization technology it should be considered experemental at this time. Even though this subsystem is not yet recommended for production, it is stable enough to introduce to the community and something wed like feedback on as we improve it over the next cycle.

OpenStack-Ansible presently supports two different container technologies, LXC and nspawn. These two container technologies can be used separately or together within the same cluster but has a limitation of only one setting per host.

Using `shared-infra_hosts` as an example, consider this `openstack_user_config.yml` configuration:

```
shared-infra_hosts:
  infra1:
    ip: 172.29.236.101
    container_vars:
      container_tech: lxc
  infra2:
    ip: 172.29.236.102
    container_vars:
      container_tech: nspawn
  infra3:
    ip: 172.29.236.103
```

In this example the three hosts are assigned to the *shared-infra_hosts* group, and will deploy containerized workloads using `lxc` on **infra1**, `nspawn` on **infra2**, and `lxc` on **infra3**. Notice **infra3** does not define the `container_tech` option because it not required. If this option is undefined the value will automatically be set to `lxc` within the generated inventory. The two supported options for the `container_tech` configuration variable are `lxc` or `nspawn`.

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 (`haproxy_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
# ========
```

(continues on next page)

(continued from previous page)

```
#
# 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.
#
#   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:
#   container: 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
```

(continues on next page)

(continued from previous page)

```
# 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.
#
# 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 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 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)
```

(continues on next page)

(continued from previous page)

```
#       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.
#
#       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_container_address (required, boolean)
#       If true, the load balancer uses this IP address to access services
#       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.
```

(continues on next page)

(continued from previous page)

```
#
#       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, poitining 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. Maps to the
#     'eth10' device in containers. Applies to all neutron agent containers
#     and neutron agents on bare metal hosts.
#   - Network of type 'vlan' that contains neutron VLAN networks 101 to 200
#     and 301 to 400 and uses the 'br-vlan' bridge on target hosts. Maps to
#     the 'eth11' device in containers. 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
```

(continues on next page)

(continued from previous page)

```
#           - hosts
#         type: "raw"
#         container_bridge: "br-mgmt"
#         container_interface: "eth1"
#         container_type: "veth"
#         ip_from_q: "container"
#         is_container_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_linuxbridge_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:
#         group_binds:
#           - neutron_linuxbridge_agent
#         container_bridge: "br-vlan"
#         container_type: "veth"
#         container_interface: "eth11"
#         type: "vlan"
#         range: "101:200,301:400"
#         net_name: "vlan"
#     - network:
#         group_binds:
#           - neutron_linuxbridge_agent
#         container_bridge: "br-vlan"
#         container_type: "veth"
#         container_interface: "eth12"
#         host_bind_override: "eth12"
#         type: "flat"
#         net_name: "flat"
#
# --------
#
# 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.
```

(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 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
#
# List of target hosts on which to deploy shared infrastructure services
# and define the container_tech for a specific infra node. If this setting
# is omitted the inventory generation system will default to "lxc". Accpetable
# options are "lxc" and "nspawn".
#
#   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.
#
# Example:
#
# Define three shared infrastructure hosts with different "container_tech":
#
# shared-infra_hosts:
#   infra1:
#     ip: 172.29.236.101
#     container_vars:
#       container_tech: nspawn
#   infra2:
#     ip: 172.29.236.102
#     container_vars:
#       container_tech: lxc
#   infra3:
#     ip: 172.29.236.103
#
# --------
#
# Level: repo-infra_hosts (required)
# List of target hosts on which to deploy the package repository. Recommend
# minimum three target hosts for this service. Typically contains the same
# target hosts as the 'shared-infra_hosts' level.
```

(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 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: 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)
#   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)
```

(continues on next page)

(continued from previous page)

```
#     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
# 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
#
# --------
#
# 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.
#
#   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:
#
```

(continues on next page)

(continued from previous page)

```
# 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.
#
#     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)
```

(continues on next page)

(continued from previous page)

```
# 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
#         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'.
```

(continues on next page)

(continued from previous page)

```
#
#           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)
#           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:
```

(continues on next page)

(continued from previous page)

```
#       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':
#
#     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:
```

(continues on next page)

(continued from previous page)

```
#       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
#           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:
#
# Use the lvm backend with manila:
#
```

(continues on next page)

(continued from previous page)

```
#     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_ip: "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: log_hosts (optional)
# List of target hosts on which to deploy logging services. Recommend
# one minimum target host for this service.
#
#   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 a logging host:
#
# log_hosts:
#   log1:
#     ip: 172.29.236.171
#
# --------
#
# Level: haproxy_hosts (optional)
```

(continues on next page)

(continued from previous page)

```
# 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
#     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.
#
# haproxy_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 cam ping a public IP address to check its status.  To enable this
# feature, set the ``keepalived_ping_address`` variable in the
# ``user_variables.yml`` file.
```

Inspecting and manipulating the inventory

Warning: Never edit or delete the files `/etc/openstack_deploy/openstack_inventory.json` or `/etc/openstack_deploy/openstack_hostnames_ips.yml`. This can lead to file corruptions, and

problems with the inventory: hosts and container could disappear and new ones would appear, 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.

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

`dynamic_inventory.`**`args`**(*arg_list*)
Setup argument Parsing.

1.4.4 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 openstack-ansible.rc file, 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:
    description:
      - Path of a Jinja2 formatted template on the local server. This can
```

(continues on next page)

(continued from previous page)

```
        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:

```
find . -name "main.yml" -exec grep '_.*_overrides:' {} \; \
    | grep -v "^#" \
    | sort -u
```

The following override variables are currently available:

Galera:

- galera_client_my_cnf_overrides
- galera_my_cnf_overrides
- galera_cluster_cnf_overrides
- galera_debian_cnf_overrides

Telemetry service (ceilometer):

- ceilometer_policy_overrides
- ceilometer_ceilometer_conf_overrides
- ceilometer_event_definitions_yaml_overrides
- ceilometer_event_pipeline_yaml_overrides
- ceilometer_pipeline_yaml_overrides

Block Storage (cinder):

- cinder_policy_overrides
- cinder_rootwrap_conf_overrides
- cinder_api_paste_ini_overrides
- cinder_cinder_conf_overrides

Image service (glance):

- glance_glance_api_paste_ini_overrides
- glance_glance_api_conf_overrides
- glance_glance_cache_conf_overrides
- glance_glance_manage_conf_overrides
- glance_glance_registry_paste_ini_overrides
- glance_glance_registry_conf_overrides
- glance_glance_scrubber_conf_overrides
- glance_glance_scheme_json_overrides
- glance_policy_overrides

Orchestration service (heat):

- heat_heat_conf_overrides
- heat_api_paste_ini_overrides
- heat_default_yaml_overrides
- heat_aws_rds_dbinstance_yaml_overrides
- heat_policy_overrides

Identity service (keystone):

- keystone_keystone_conf_overrides
- keystone_keystone_default_conf_overrides
- keystone_keystone_paste_ini_overrides
- keystone_policy_overrides

Networking service (neutron):

- neutron_neutron_conf_overrides
- neutron_ml2_conf_ini_overrides
- neutron_dhcp_agent_ini_overrides
- neutron_api_paste_ini_overrides
- neutron_rootwrap_conf_overrides
- neutron_policy_overrides
- neutron_dnsmasq_neutron_conf_overrides
- neutron_l3_agent_ini_overrides
- neutron_metadata_agent_ini_overrides
- neutron_metering_agent_ini_overrides

Compute service (nova):

- nova_nova_conf_overrides
- nova_rootwrap_conf_overrides
- nova_api_paste_ini_overrides
- nova_policy_overrides

Object Storage service (swift):

- swift_swift_conf_overrides
- swift_swift_dispersion_conf_overrides
- swift_proxy_server_conf_overrides
- swift_account_server_conf_overrides
- swift_account_server_replicator_conf_overrides
- swift_container_server_conf_overrides
- swift_container_server_replicator_conf_overrides
- swift_object_server_conf_overrides
- swift_object_server_replicator_conf_overrides

Tempest:

- tempest_tempest_conf_overrides

pip:

- pip_global_conf_overrides

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`.

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
    - 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`.

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.

Using OpenStack-Ansible within your project

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`.

You can override the ansible-role-requirement file used by defining the environment variable `ANSIBLE_ROLE_FILE` before running the `bootstrap-ansible.sh` script.

1.4.5 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

This project 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.

Project scope

This project 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 - Intro to 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.

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 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 multimaster 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). OSA 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. OSA 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.

- Log aggregation host

A rsyslog service is optionally set up to receive rsyslog traffic from all hosts and containers. You can replace rsyslog with any alternative log receiver.

- 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, OSA modifies `/etc/hosts` entries for all hosts in the environment.

OpenStack services

OSA 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)
- Filesystem 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.

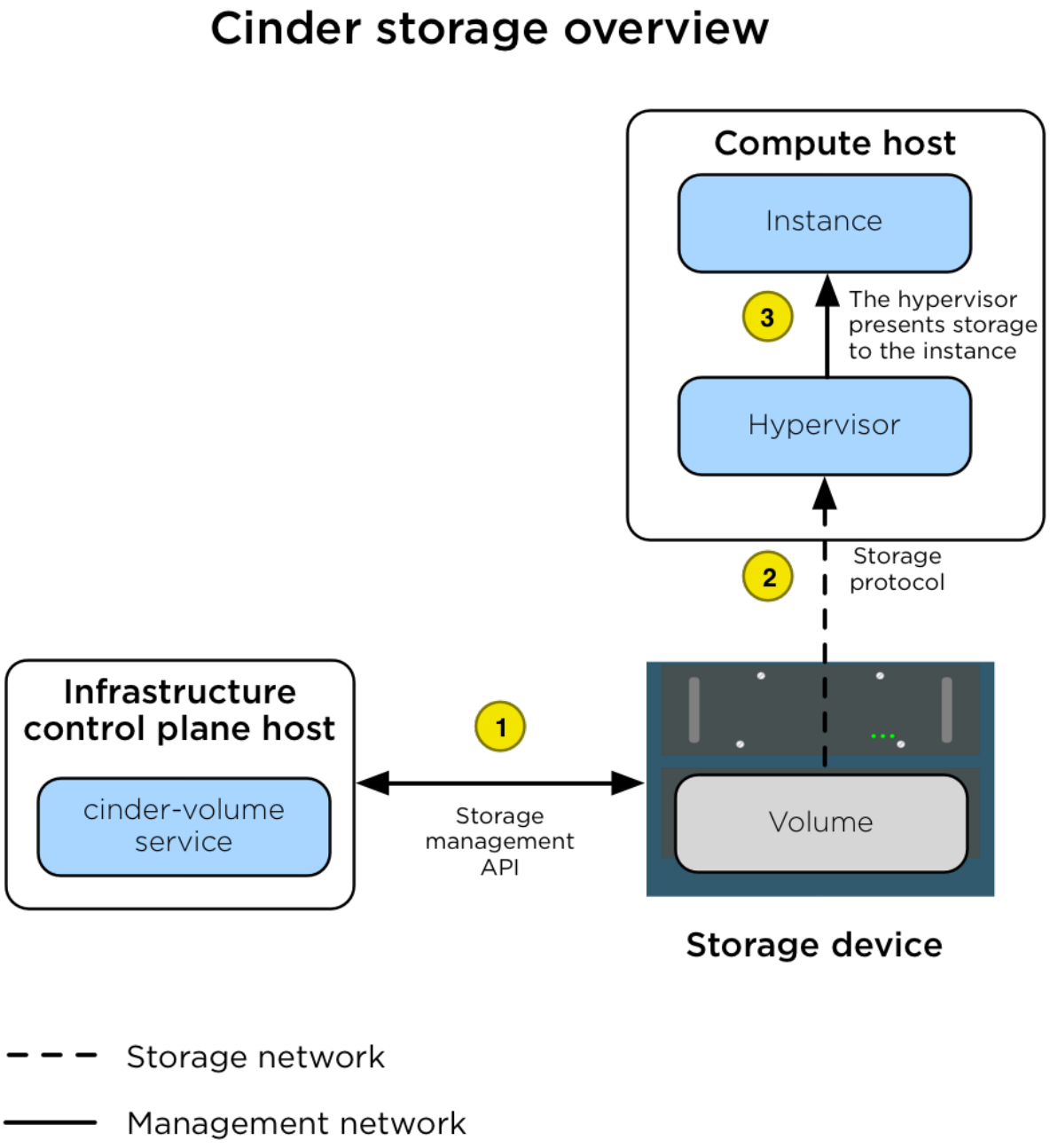


Fig. 1: The diagram shows the following steps.

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.

Filesystem 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 bridging between the container and the host interfaces to ensure that all traffic from containers flows over multiple host interfaces. 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*.

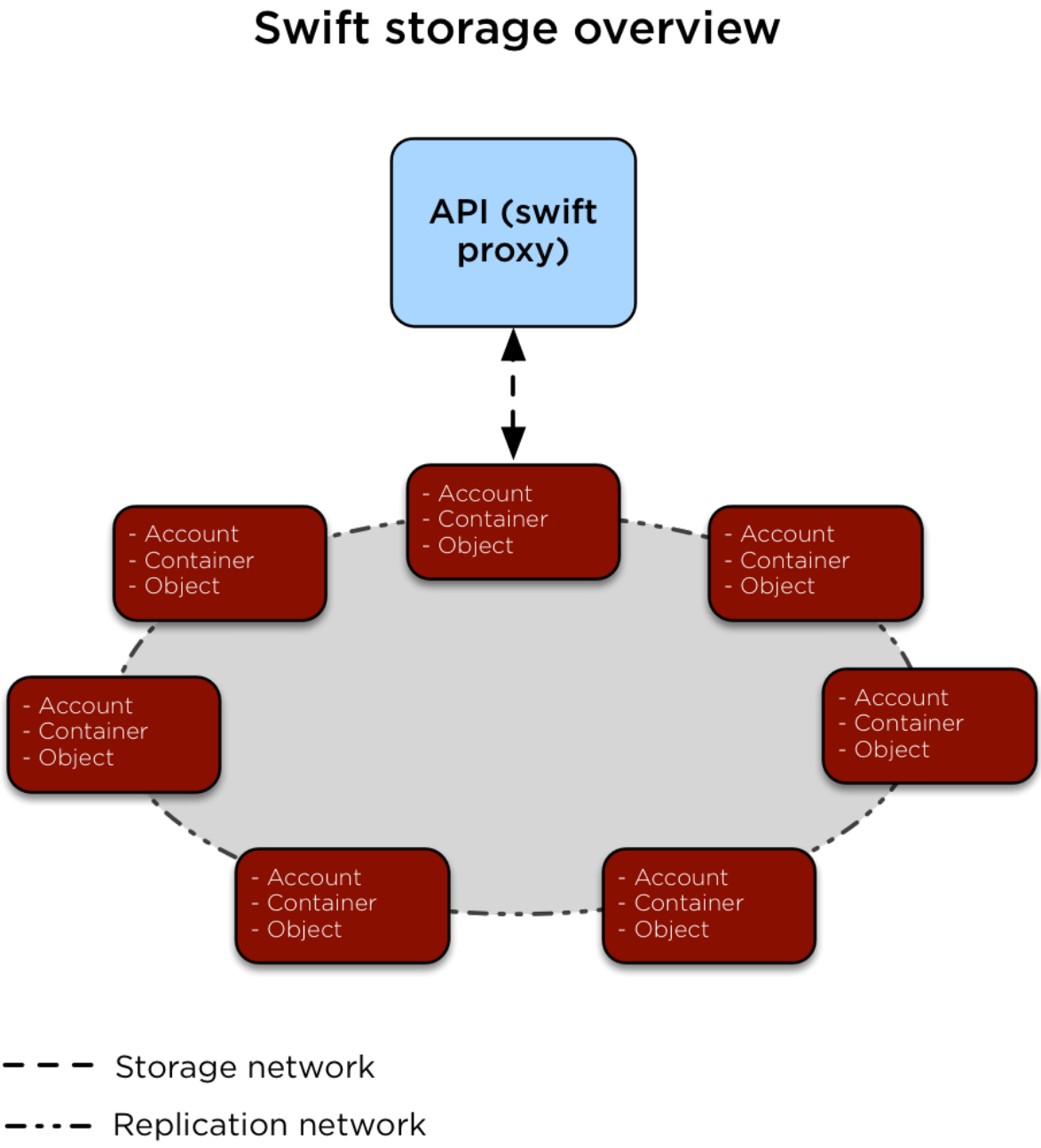


Fig. 2: The `swift-proxy` service is accessed by clients via the load balancer on the management network (`br-mgmt`). The `swift-proxy` service communicates with the Account, Container, and Object services on the Object Storage hosts via the storage network(`br-storage`). Replication between the Object Storage hosts is done via the replication network (`br-repl`).

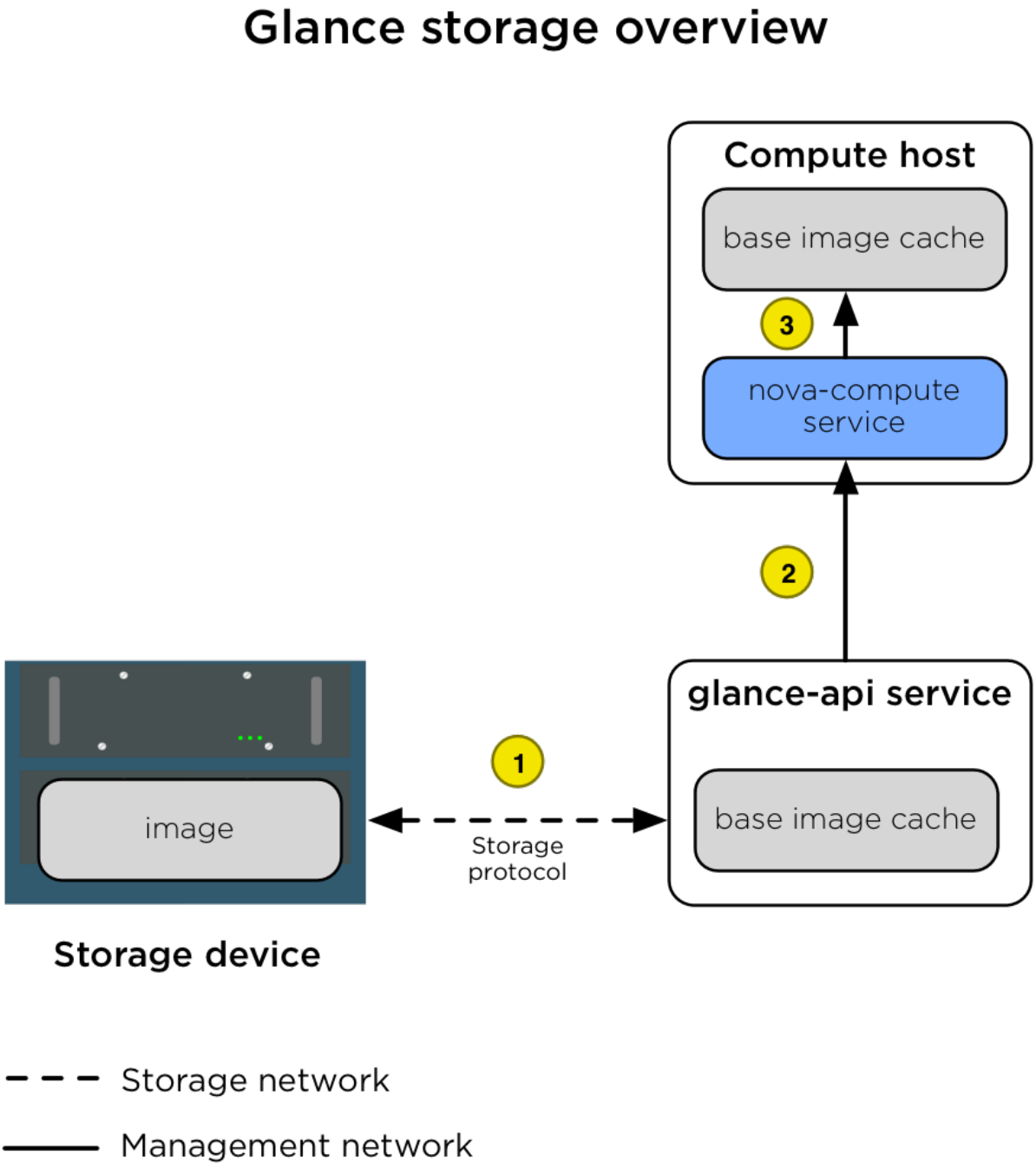


Fig. 3: The diagram shows the following steps.

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.

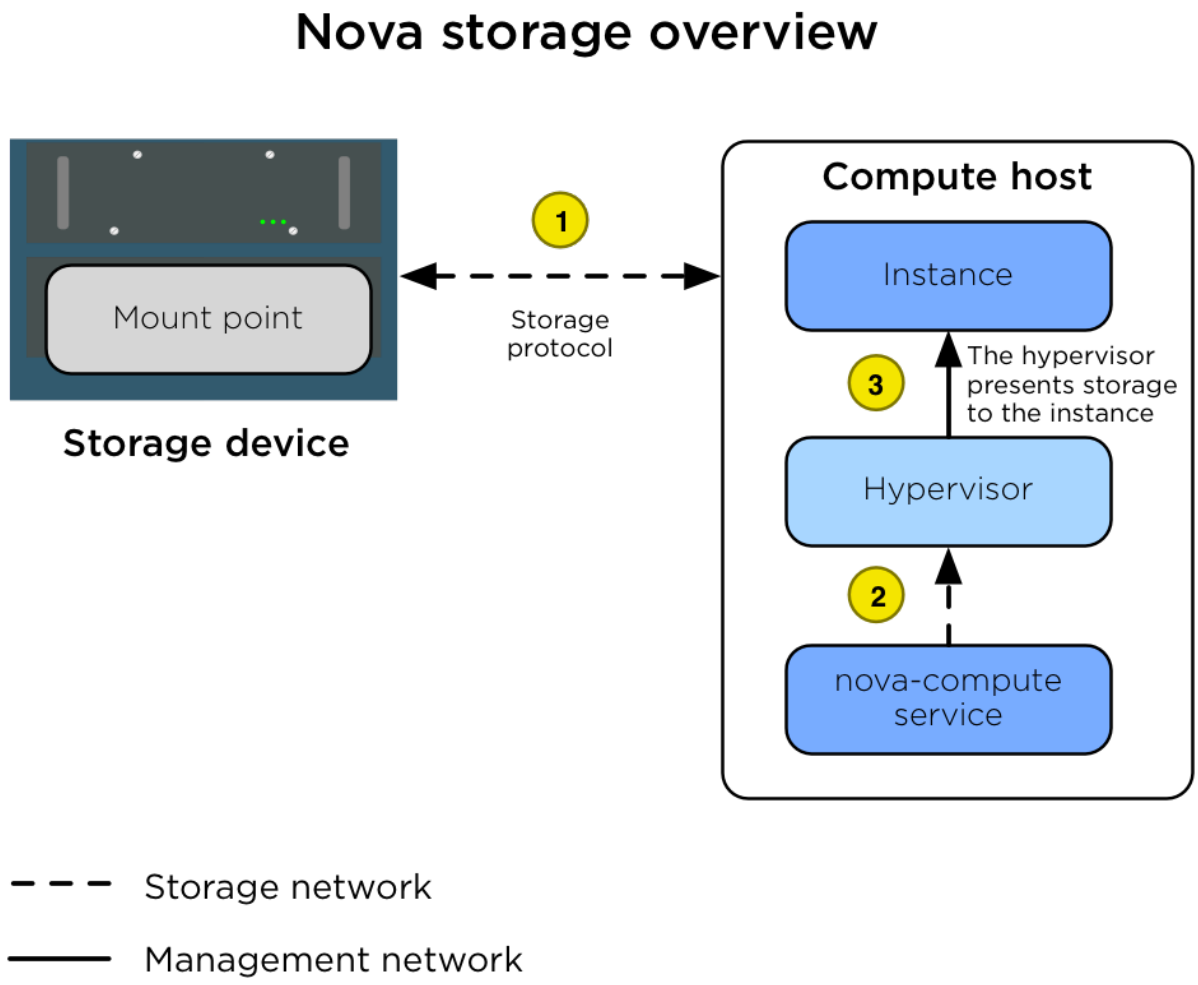


Fig. 4: The diagram shows the following steps.

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.

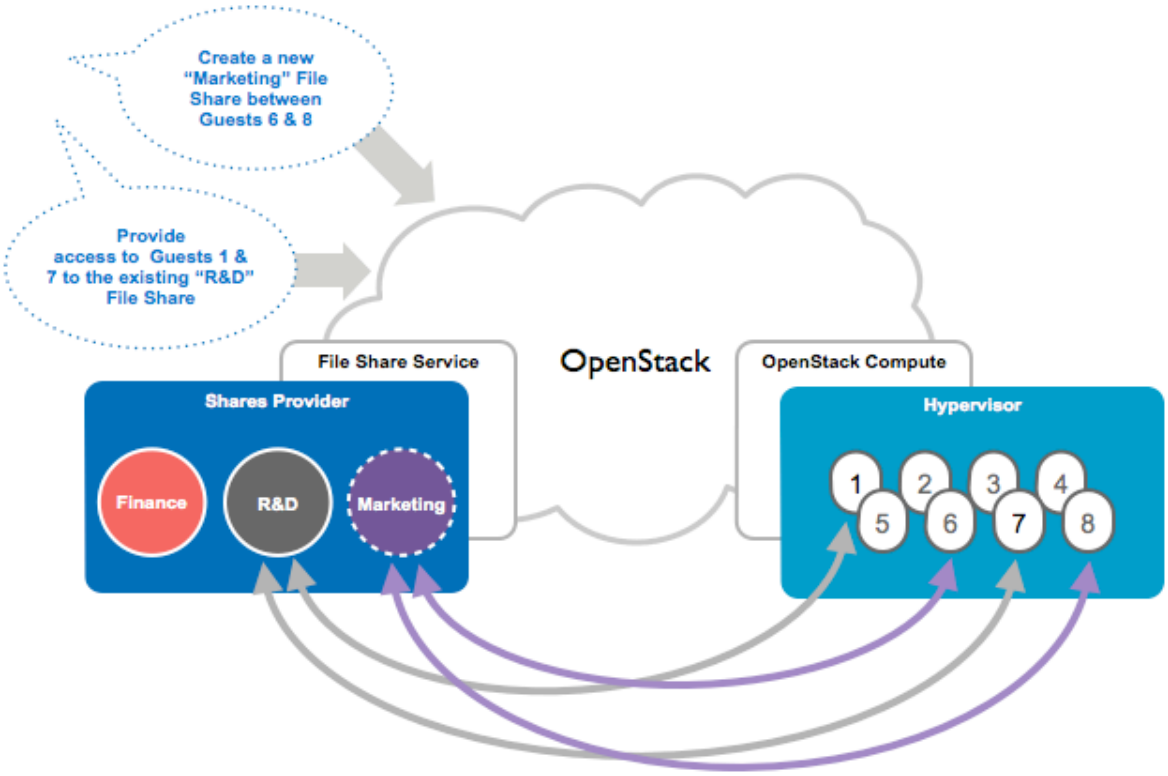


Fig. 5: The diagram shows a basic overview of the manila service.

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.

Linux bridges

The combination of containers and flexible deployment options requires implementation of advanced Linux networking features, such as bridges and namespaces.

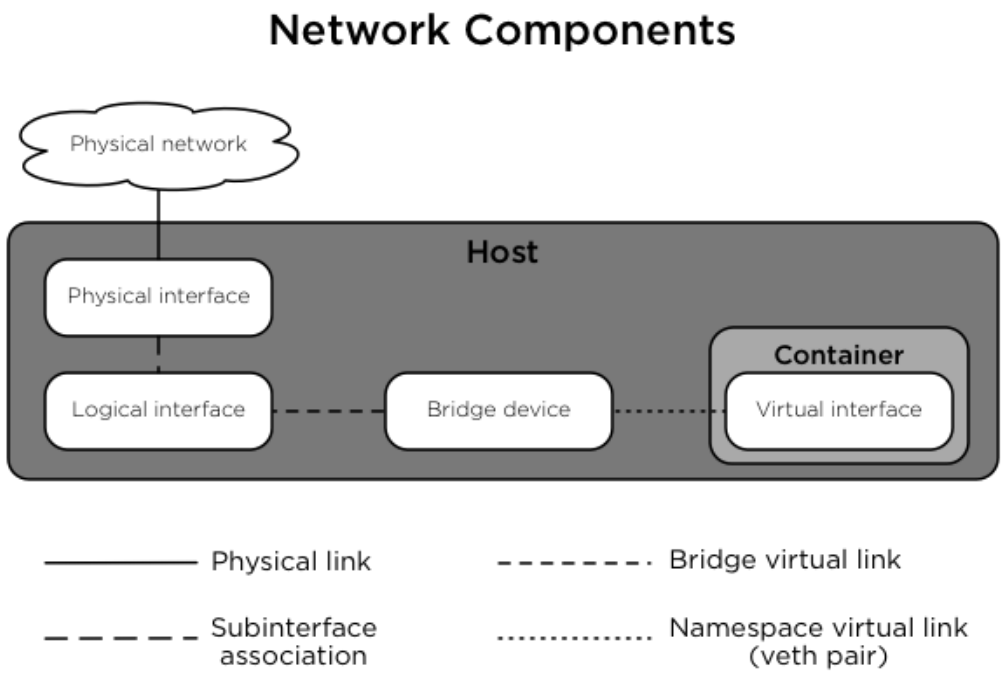
- Bridges provide layer 2 connectivity (similar to switches) among physical, logical, and virtual network interfaces within a host. After a bridge is created, the network interfaces are virtually plugged in to it.

  OpenStack-Ansible uses bridges to connect physical and logical network interfaces on the host to virtual network interfaces within containers.

- Namespaces provide logically separate layer 3 environments (similar to routers) 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.

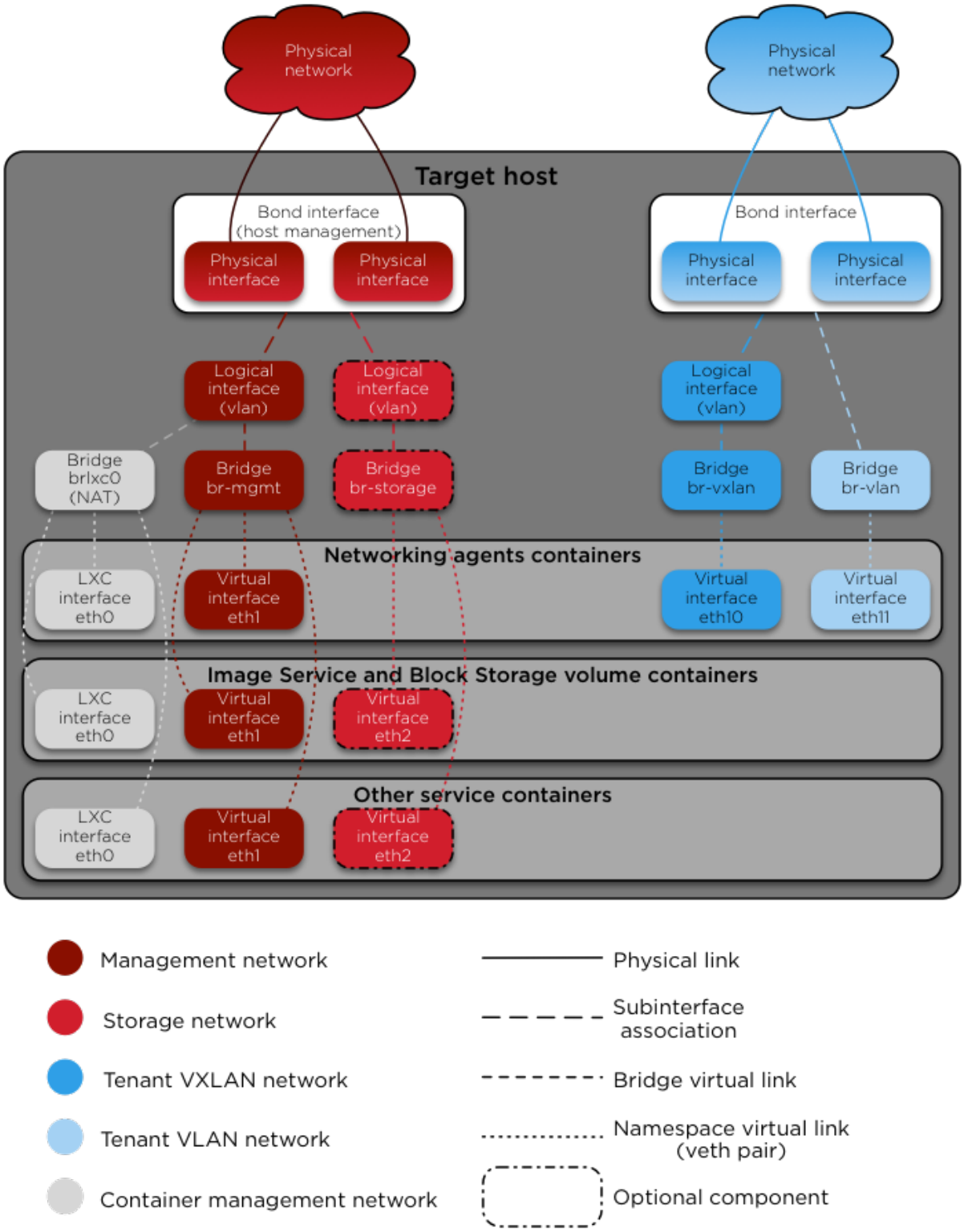
The following image demonstrates how the container network interfaces are connected to the hosts bridges and physical network interfaces:



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:

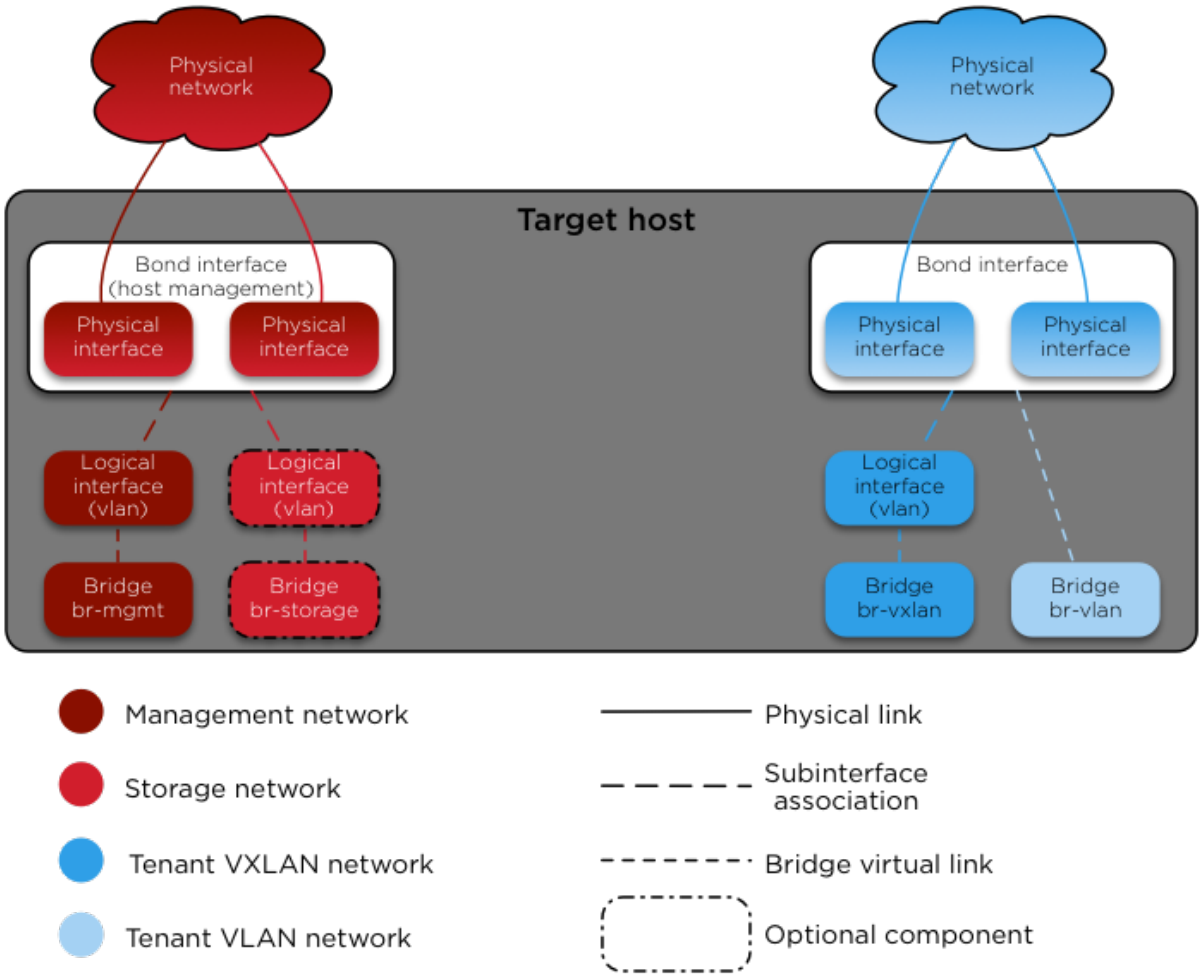


The interface `lxcbr0` provides connectivity for the containers to the outside world, thanks to dnsmasq (dhcp/dns) + NAT.

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.

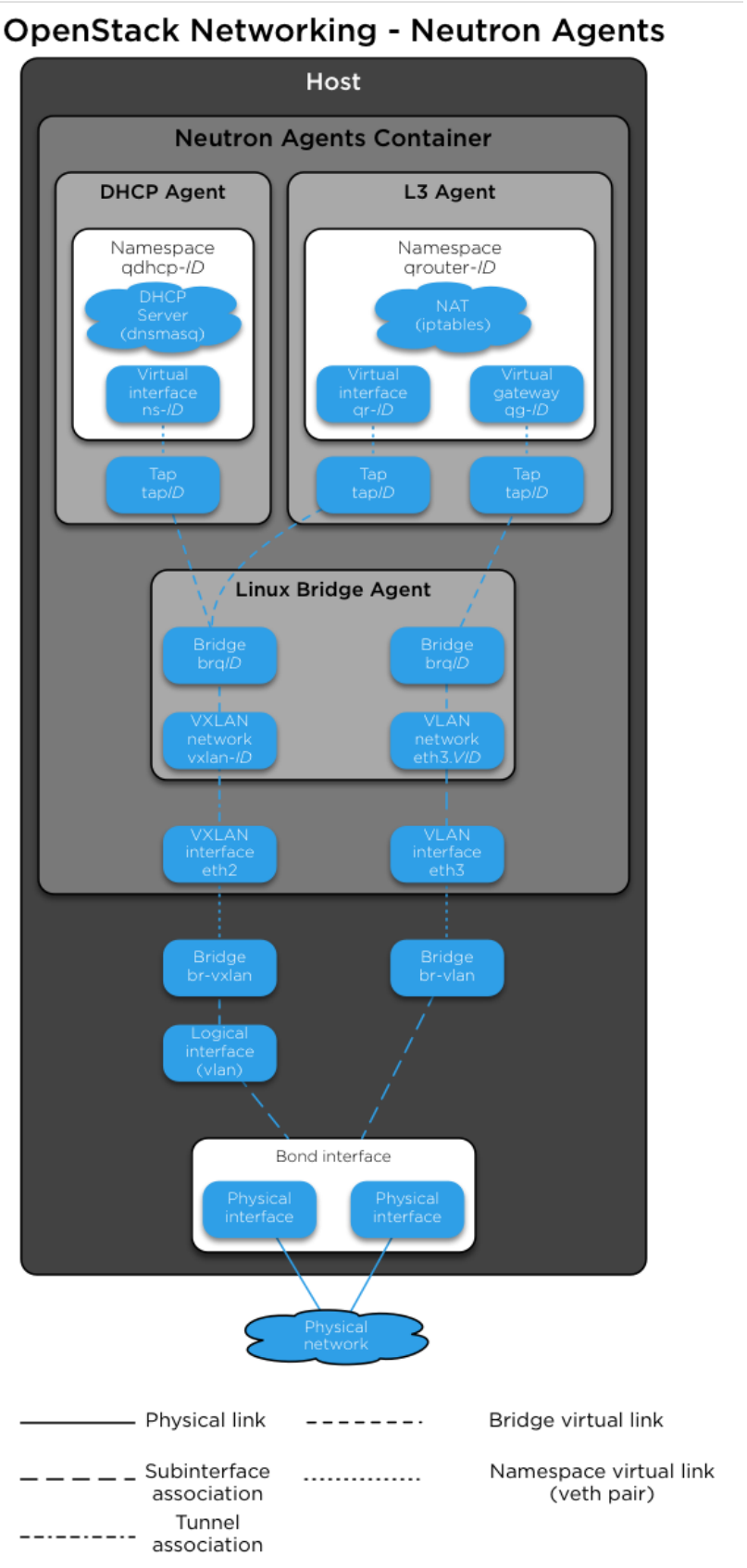
Services running on metal (deploying directly on the physical hosts)

OpenStack-Ansible deploys the Compute service on the physical host rather than in a container. The following diagram shows how to use bridges for network connectivity:

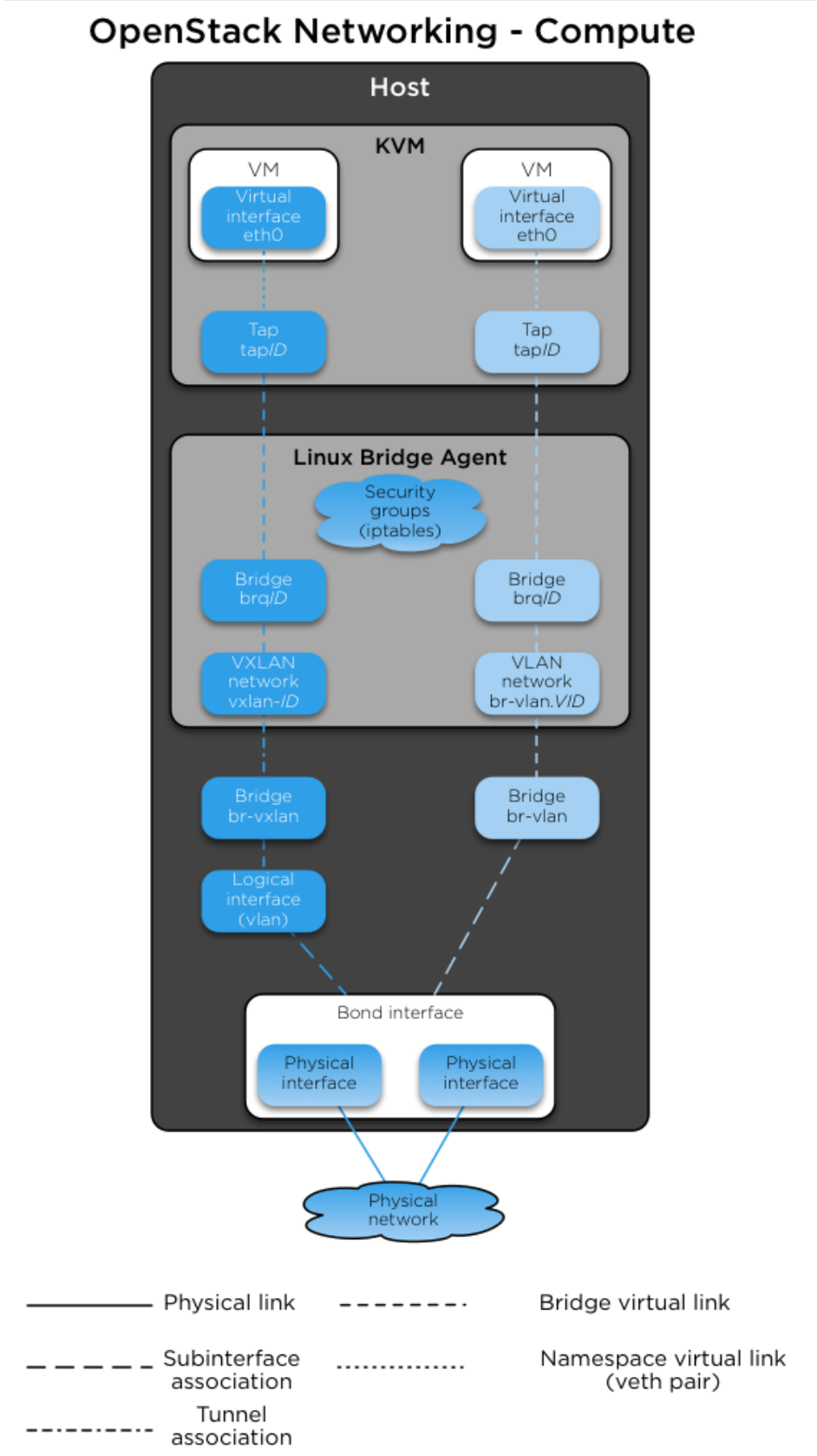


Neutron traffic

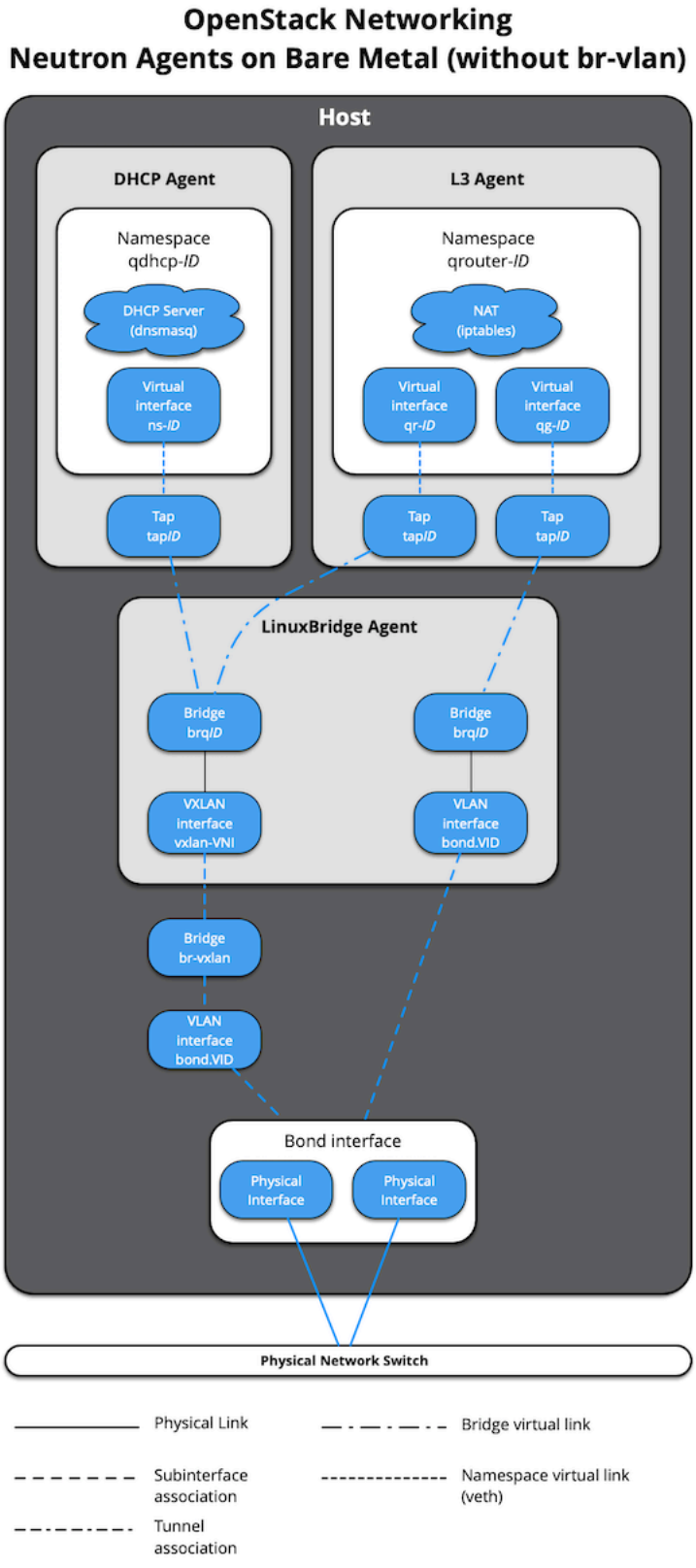
The following diagram shows how the Networking service (neutron) agents work with the `br-vlan` and `br-vxlan` bridges. Neutron is configured to use a DHCP agent, an L3 agent, and a Linux Bridge agent within a networking-agents container. The diagram shows how DHCP agents provide information (IP addresses and DNS servers) to the instances, and how routing works on the image.


OpenStack Networking - Neutron Agents
Host
Neutron Agents Container
DHCP Agent
L3 Agent
Namespace qdhcp-ID
DHCP Server (dnsmasq)
Virtual interface ns-ID
Tap tapID
Namespace qrouter-ID
NAT (iptables)
Virtual interface qr-ID
Virtual gateway qg-ID
Tap tapID
Tap tapID
Linux Bridge Agent
Bridge brqID
VXLAN network vxlan-ID
VXLAN interface eth2
Bridge brqID
VLAN network eth3.VID
VLAN interface eth3
Bridge br-vxlan
Logical interface (vlan)
Bridge br-vlan
Bond interface
Physical interface
Physical interface
Physical network
Physical link
Bridge virtual link
Subinterface association
Namespace virtual link (veth pair)
Tunnel association

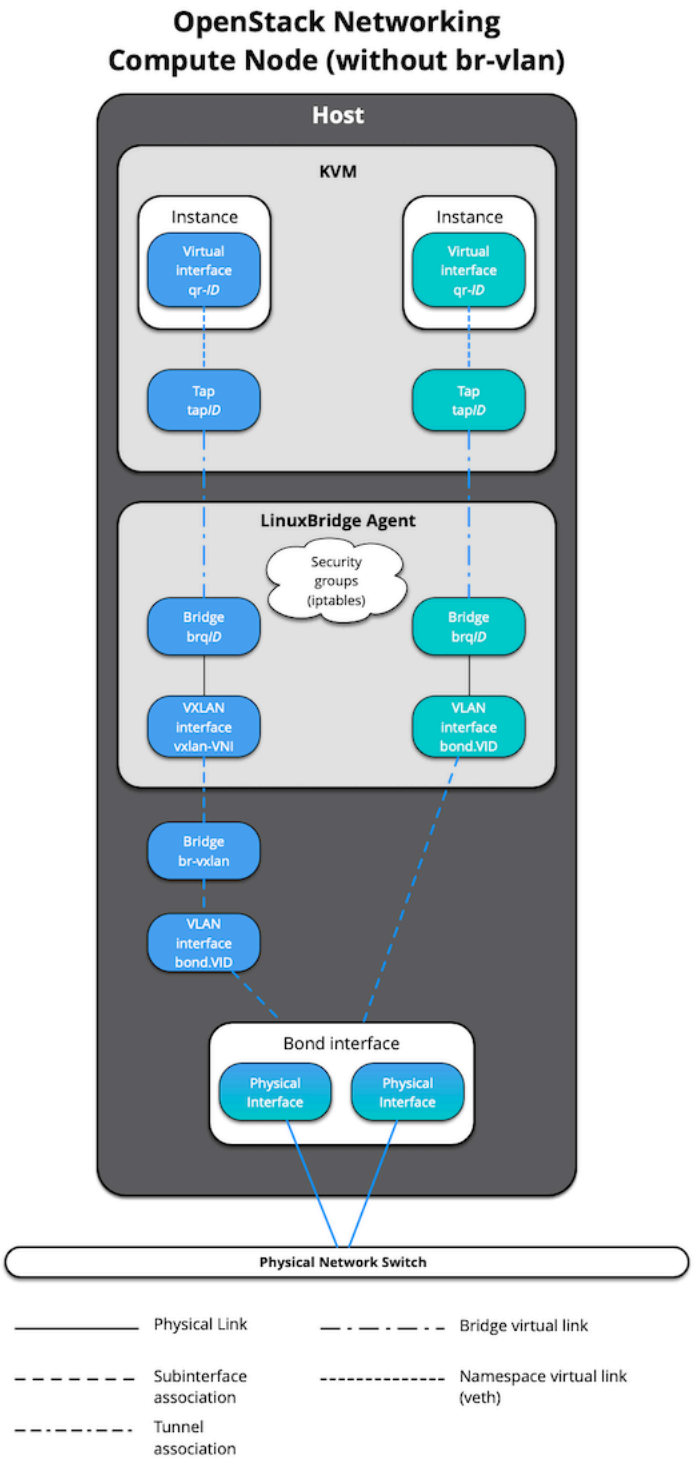
The following diagram shows how virtual machines connect to the `br-vlan` and `br-vxlan` bridges and send traffic to the network outside the host:


OpenStack Networking - Compute
Host
KVM
VM
Virtual interface eth0
VM
Virtual interface eth0
Tap tapID
Tap tapID
Linux Bridge Agent
Security groups (iptables)
Bridge brqID
Bridge brqID
VXLAN network vxlan-ID
VLAN network br-vlan.VID
Bridge br-vxlan
Bridge br-vlan
Logical interface (vlan)
Bond interface
Physical interface
Physical interface
Physical network
Physical link
Bridge virtual link
Subinterface association
Namespace virtual link (veth pair)
Tunnel association

When Neutron agents are deployed on metal on a network node or collapsed infra/network node, the `Neutron Agents` container and respective virtual interfaces are no longer implemented. In addition, use of the `host_bind_override` override when defining provider networks allows Neutron to interface directly with a physical interface or bond instead of the `br-vlan` bridge. The following diagram reflects the differences in the virtual network layout.



The absence of `br-vlan` in-path of instance traffic is also reflected on compute nodes, as shown in the following diagram.



1.4.6 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
```

Systemd nspawn commands

PYTHON MODULE INDEX

d

INDEX

A

D