Kolla Ansible Documentation

Release 12.8.1.dev46

OpenStack Foundation

Jun 27, 2023

CONTENTS

Kollas mission is to provide production-ready containers and deployment tools for operating OpenStack clouds.

Kolla Ansible is highly opinionated out of the box, but allows for complete customization. This permits operators with minimal experience to deploy OpenStack quickly and as experience grows modify the OpenStack configuration to suit the operators exact requirements.

CHAPTER

ONE

RELATED PROJECTS

This documentation is for Kolla Ansible.

For information on building container images for use with Kolla Ansible, please refer to the Kolla image documentation.

Kayobe is a subproject of Kolla that uses Kolla Ansible and Bifrost to deploy an OpenStack control plane to bare metal.

CHAPTER

TWO

SITE NOTES

This documentation is continually updated and may not represent the state of the project at any specific prior release. To access documentation for a previous release of Kolla Ansible, append the OpenStack release name to the URL. For example, to access documentation for the Stein release: https://docs.openstack.org/kolla-ansible/stein

CHAPTER

THREE

RELEASE NOTES

The release notes for the project can be found here: https://docs.openstack.org/releasenotes/kolla-ansible/

CHAPTER

FOUR

ADMINISTRATOR GUIDE

4.1 Admin Guides

4.1.1 Advanced Configuration

Endpoint Network Configuration

When an OpenStack cloud is deployed, the REST API of each service is presented as a series of endpoints. These endpoints are the admin URL, the internal URL, and the external URL.

Kolla offers two options for assigning these endpoints to network addresses: - Combined - Where all three endpoints share the same IP address - Separate - Where the external URL is assigned to an IP address that is different than the IP address shared by the internal and admin URLs

The configuration parameters related to these options are: - kolla_internal_vip_address - network_interface - kolla_external_vip_address - kolla_external_vip_interface

For the combined option, set the two variables below, while allowing the other two to accept their default values. In this configuration all REST API requests, internal and external, will flow over the same network.

```
kolla_internal_vip_address: "10.10.10.254"
network_interface: "eth0"
```

For the separate option, set these four variables. In this configuration the internal and external REST API requests can flow over separate networks.

```
kolla_internal_vip_address: "10.10.10.254"
network_interface: "eth0"
kolla_external_vip_address: "10.10.20.254"
kolla_external_vip_interface: "eth1"
```

Fully Qualified Domain Name Configuration

When addressing a server on the internet, it is more common to use a name, like `www.example.net`, instead of an address like `10.10.10.254`. If you prefer to use names to address the endpoints in your kolla deployment use the variables:

- kolla_internal_fqdn
- kolla_external_fqdn

```
kolla_internal_fqdn: inside.mykolla.example.net
kolla_external_fqdn: mykolla.example.net
```

Provisions must be taken outside of kolla for these names to map to the configured IP addresses. Using a DNS server or the `/etc/hosts` file are two ways to create this mapping.

RabbitMQ Hostname Resolution

RabbitMQ doesnt work with IP address, hence the IP address of `api_interface` should be resolvable by hostnames to make sure that all RabbitMQ Cluster hosts can resolve each others hostname beforehand.

TLS Configuration

Configuration of TLS is now covered *here*.

OpenStack Service Configuration in Kolla

An operator can change the location where custom config files are read from by editing `/etc/kolla/globals.yml` and adding the following line.

```
# The directory to merge custom config files the kolla's config files
node_custom_config: "/etc/kolla/config"
```

Kolla allows the operator to override configuration of services. Kolla will generally look for a file in `/etc/kolla/config/<< config file >>`, `/etc/kolla/config/<< service name >>/<< config file >>` or `/etc/kolla/config/<< service name >>/<< hostname >>/<< config file >>`, but these locations sometimes vary and you should check the config task in the appropriate Ansible role for a full list of supported locations. For example, in the case of `nova.conf` the following locations are supported, assuming that you have services using `nova.conf` running on hosts called `controller-0001`, `controller-0002` and `controller-0003`:

- `/etc/kolla/config/nova.conf`
- `/etc/kolla/config/nova/controller-0001/nova.conf`
- `/etc/kolla/config/nova/controller-0002/nova.conf`
- `/etc/kolla/config/nova/controller-0003/nova.conf`
- `/etc/kolla/config/nova/nova-scheduler.conf`

Using this mechanism, overrides can be configured per-project, per-project-service or per-project-service-on-specified-host.

Overriding an option is as simple as setting the option under the relevant section. For example, to set `override scheduler_max_attempts` in nova scheduler, the operator could create `/etc/kolla/config/nova/nova-scheduler.conf` with content:

```
[DEFAULT]
scheduler_max_attempts = 100
```

If the operator wants to configure compute node cpu and ram allocation ratio on host myhost, the operator needs to create file `/etc/kolla/config/nova/myhost/nova.conf` with content:

```
[DEFAULT]
cpu_allocation_ratio = 16.0
ram_allocation_ratio = 5.0
```

This method of merging configuration sections is supported for all services using Oslo Config, which includes the vast majority of OpenStack services, and in some cases for services using YAML configuration. Since the INI format is an informal standard, not all INI files can be merged in this way. In these cases Kolla supports overriding the entire config file.

Additional flexibility can be introduced by using Jinja conditionals in the config files. For example, you may create Nova cells which are homogeneous with respect to the hypervisor model. In each cell, you may wish to configure the hypervisors differently, for example the following override shows one way of setting the `bandwidth_poll_interval` variable as a function of the cell:

```
[DEFAULT]
{% if 'cell0001' in group_names %}
bandwidth_poll_interval = 100
{% elif 'cell0002' in group_names %}
bandwidth_poll_interval = -1
{% else %}
bandwidth_poll_interval = 300
{% endif %}
```

An alternative to Jinja conditionals would be to define a variable for the `bandwidth_poll_interval` and set it in according to your requirements in the inventory group or host vars:

```
[DEFAULT]
bandwidth_poll_interval = {{ bandwidth_poll_interval }}
```

Kolla allows the operator to override configuration globally for all services. It will look for a file called `/etc/kolla/config/global.conf`.

For example to modify database pool size connection for all services, the operator needs to create `/etc/kolla/config/global.conf` with content:

```
[database]
max_pool_size = 100
```

OpenStack policy customisation

OpenStack services allow customisation of policy. Since the Queens release, default policy configuration is defined within the source code for each service, meaning that operators only need to override rules they wish to change. Projects typically provide documentation on their default policy configuration, for example, Keystone.

Policy can be customised via JSON or YAML files. As of the Wallaby release, the JSON format is deprecated in favour of YAML. One major benefit of YAML is that it allows for the use of comments.

For example, to customise the Neutron policy in YAML format, the operator should add the customised rules in `/etc/kolla/config/neutron/policy.yaml`.

The operator can make these changes after services have been deployed by using the following command:

```
kolla-ansible deploy
```

In order to present a user with the correct interface, Horizon includes policy for other services. Customisations made to those services may need to be replicated in Horizon. For example, to customise the Neutron policy in YAML format for Horizon, the operator should add the customised rules in `/etc/kolla/config/horizon/neutron_policy.yaml`.

IP Address Constrained Environments

If a development environment doesnt have a free IP address available for VIP configuration, the hosts IP address may be used here by disabling HAProxy by adding:

```
enable_haproxy: "no"
```

Note this method is not recommended and generally not tested by the Kolla community, but included since sometimes a free IP is not available in a testing environment.

In this mode it is still necessary to configure `kolla_internal_vip_address`, and it should take the IP address of the `api_interface` interface.

External Elasticsearch/Kibana environment

It is possible to use an external Elasticsearch/Kibana environment. To do this first disable the deployment of the central logging.

```
enable_central_logging: "no"
```

Now you can use the parameter `elasticsearch_address` to configure the address of the external Elasticsearch environment.

Non-default <service> port

It is sometimes required to use a different than default port for service(s) in Kolla. It is possible with setting `<service>_port` in `globals.yml` file. For example:

```
database_port: 3307
```

As `<service>_port` value is saved in different services configuration so its advised to make above change before deploying.

Use an external Syslog server

By default, Fluentd is used as a syslog server to collect Swift and HAProxy logs. When Fluentd is disabled or you want to use an external syslog server, You can set syslog parameters in `globals.yml` file. For example:

```
syslog_server: "172.29.9.145"
syslog_udp_port: "514"
```

You can also set syslog facility names for Swift and HAProxy logs. By default, Swift and HAProxy use `local0` and `local1`, respectively.

```
syslog_swift_facility: "local0"
syslog_haproxy_facility: "local1"
```

If Glance TLS backend is enabled (`glance_enable_tls_backend`), the syslog facility for the `glance_tls_proxy` service uses `local2` by default. This can be set via `syslog_glance_tls_proxy_facility`.

If Neutron TLS backend is enabled (`neutron_enable_tls_backend`), the syslog facility for the `neutron_tls_proxy` service uses `local4` by default. This can be set via `syslog_neutron_tls_proxy_facility`.

Mount additional Docker volumes in containers

It is sometimes useful to be able to mount additional Docker volumes into one or more containers. This may be to integrate 3rd party components into OpenStack, or to provide access to site-specific data such as x.509 certificate bundles.

Additional volumes may be specified at three levels:

- globally
- per-service (e.g. nova)
- per-container (e.g. `nova-api`)

To specify additional volumes globally for all containers, set `default_extra_volumes` in `globals.yml`. For example:

```
default_extra_volumes:
  - "/etc/foo:/etc/foo"
```

To specify additional volumes for all containers in a service, set `<service_name>_extra_volumes` in `globals.yml`. For example:

```
nova_extra_volumes:
  - "/etc/foo:/etc/foo"
```

To specify additional volumes for a single container, set `<container_name>_extra_volumes` in `globals.yml`. For example:

```
nova_libvirt_extra_volumes:
  - "/etc/foo:/etc/foo"
```

4.1.2 TLS

This guide describes how to configure Kolla Ansible to deploy OpenStack with TLS enabled. Enabling TLS on the provided internal and/or external VIP address allows OpenStack clients to authenticate and encrypt network communication with OpenStack services.

When an OpenStack service exposes an API endpoint, Kolla Ansible will configure HAProxy for that service to listen on the internal and/or external VIP address. The HAProxy container load-balances requests on the VIPs to the nodes running the service container.

There are two different layers of TLS configuration for OpenStack APIs:

1. Enabling TLS on the internal and/or external VIP, so communication between an OpenStack client and the HAProxy listening on the VIP is secure.
2. Enabling TLS on the backend network, so communication between HAProxy and the backend API services is secure.

Note: TLS authentication is based on certificates that have been signed by trusted Certificate Authorities. Examples of commercial CAs are Comodo, Symantec, GoDaddy, and GlobalSign. Letsencrypt.org is a CA that will provide trusted certificates at no charge. If using a trusted CA is not possible for your project, you can use a private CA, e.g. Hashicorp Vault, to create a certificate for your domain, or see *Generating a Private Certificate Authority* to use a Kolla Ansible generated private CA.

For details on ACME-enabled CAs, such as letsencrypt.org, please see *ACME http-01 challenge support*.

Quick Start

Note: The certificates generated by Kolla Ansible use a simple Certificate Authority setup and are not suitable for a production deployment. Only certificates signed by a trusted Certificate Authority should be used in a production deployment.

To deploy OpenStack with TLS enabled for the external, internal and backend APIs, configure the following in `globals.yml`:

```
kolla_enable_tls_internal: "yes"
kolla_enable_tls_external: "yes"
kolla_enable_tls_backend: "yes"
kolla_copy_ca_into_containers: "yes"
```

If deploying on Debian or Ubuntu:

```
openstack_cacert: "/etc/ssl/certs/ca-certificates.crt"
```

If on CentOS or RHEL:

```
openstack_cacert: "/etc/pki/tls/certs/ca-bundle.crt"
```

The Kolla Ansible `certificates` command generates a private test Certificate Authority, and then uses the CA to sign the generated certificates for the enabled VIP(s) to test TLS in your OpenStack deployment. Assuming you are using the `multinode` inventory:

```
kolla-ansible -i ~/multinode certificates
```

TLS Configuration for internal/external VIP

The configuration variables that control TLS for the internal and/or external VIP are:

- `kolla_enable_tls_external`
- `kolla_enable_tls_internal`
- `kolla_internal_fqdn_cert`
- `kolla_external_fqdn_cert`

Note: If TLS is enabled only on the internal or external network, then `kolla_internal_vip_address` and `kolla_external_vip_address` must be different.

If there is only a single network configured in your topology (as opposed to separate internal and external networks), TLS can only be enabled using the internal network configuration variables.

The default state for TLS networking is disabled. To enable external TLS encryption:

```
kolla_enable_tls_external: "yes"
```

To enable internal TLS encryption:

```
kolla_enable_tls_internal: "yes"
```

Two certificate files are required to use TLS securely with authentication, which will be provided by your Certificate Authority:

- server certificate with private key
- CA certificate with any intermediate certificates

The combined server certificate and private key needs to be provided to Kolla Ansible, with the path configured via `kolla_external_fqdn_cert` or `kolla_internal_fqdn_cert`. These paths default to `{{ kolla_certificates_dir }}/haproxy.pem` and `{{ kolla_certificates_dir }}/haproxy-internal.pem` respectively, where `kolla_certificates_dir` is `/etc/kolla/certificates` by default.

If the server certificate provided is not already trusted by clients, then the CA certificate file will need to be distributed to the clients. This is discussed in more detail in *Configuring the OpenStack Client for TLS* and *Adding CA Certificates to the Service Containers*.

Configuring the OpenStack Client for TLS

The location for the CA certificate for the `admin-openrc.sh` file is configured with the `kolla_admin_openrc_cacert` variable, which is not set by default. This must be a valid path on all hosts where `admin-openrc.sh` is used.

When TLS is enabled on a VIP, and `kolla_admin_openrc_cacert` is set to `/etc/pki/tls/certs/ca-bundle.crt`, an OpenStack client will have settings similar to this configured by `admin-openrc.sh`:

```
export OS_PROJECT_DOMAIN_NAME=Default
export OS_USER_DOMAIN_NAME=Default
export OS_PROJECT_NAME=admin
export OS_TENANT_NAME=admin
export OS_USERNAME=admin
export OS_PASSWORD=demoPassword
export OS_AUTH_URL=https://mykolla.example.net:5000
export OS_INTERFACE=internal
export OS_ENDPOINT_TYPE=internalURL
export OS_MISTRAL_ENDPOINT_TYPE=internalURL
export OS_IDENTITY_API_VERSION=3
export OS_REGION_NAME=RegionOne
export OS_AUTH_PLUGIN=password
# os_cacert is optional for trusted certificates
export OS_CACERT=/etc/pki/tls/certs/ca-bundle.crt
```

Adding CA Certificates to the Service Containers

To copy CA certificate files to the service containers:

```
kolla_copy_ca_into_containers: "yes"
```

When `kolla_copy_ca_into_containers` is configured to yes, the CA certificate files in `/etc/kolla/certificates/ca` will be copied into service containers to enable trust for those CA certificates. This is required for any certificates that are either self-signed or signed by a private CA, and are not already present in the service image trust store. Kolla will install these certificates in the container system wide trust store when the container starts.

All certificate file names will have the `kolla-customca-` prefix prepended to them when they are copied into the containers. For example, if a certificate file is named `internal.crt`, it will be named `kolla-customca-internal.crt` in the containers.

For Debian and Ubuntu containers, the certificate files will be copied to the `/usr/local/share/ca-certificates/` directory.

For CentOS and RHEL containers, the certificate files will be copied to the `/etc/pki/ca-trust/source/anchors/` directory.

In both cases, valid certificates will be added to the system trust store - `/etc/ssl/certs/ca-certificates.crt` on Debian and Ubuntu, and `/etc/pki/tls/certs/ca-bundle.crt` on CentOS and RHEL.

Configuring a CA bundle

OpenStack services do not always trust CA certificates from the system trust store by default. To resolve this, the `openstack_cacert` variable should be configured with the path to the CA Certificate in the container.

To use the system trust store on Debian or Ubuntu:

```
openstack_cacert: /etc/ssl/certs/ca-certificates.crt
```

For CentOS or RHEL:

```
openstack_cacert: /etc/pki/tls/certs/ca-bundle.crt
```

Back-end TLS Configuration

Enabling TLS on the backend services secures communication between the HAProxy listing on the internal/external VIP and the OpenStack services. It also enables secure end-to-end communication between OpenStack services that support TLS termination. The OpenStack services that support backend TLS termination in Victoria are: Nova, Ironic, Neutron, Keystone, Glance, Heat, Placement, Horizon, Barbican, and Cinder.

The configuration variables that control back-end TLS for service endpoints are:

- `kolla_enable_tls_backend`
- `kolla_tls_backend_cert`
- `kolla_tls_backend_key`
- `haproxy_backend_cacert`
- `haproxy_backend_cacert_dir`

The default state for back-end TLS is disabled. To enable TLS for the back-end communication:

```
kolla_enable_tls_backend: "yes"
```

It is also possible to enable back-end TLS on a per-service basis. For example, to enable back-end TLS for Keystone, set `keystone_enable_tls_backend` to `yes`.

The default values for `haproxy_backend_cacert` and `haproxy_backend_cacert_dir` should suffice if the certificate is in the system trust store. Otherwise, they should be configured to a location of the CA certificate installed in the service containers.

Each backend service requires a certificate and private key. In many cases it is necessary to use a separate certificate and key for each host, or even per-service. The following precedence is used for the certificate:

- `{{ kolla_certificates_dir }}/{{ inventory_hostname }}/{{ project_name }}-cert.pem`
- `{{ kolla_certificates_dir }}/{{ inventory_hostname }}-cert.pem`
- `{{ kolla_certificates_dir }}/{{ project_name }}-cert.pem`
- `{{ kolla_tls_backend_cert }}`

And for the private key:

- {{ kolla_certificates_dir }}/{{ inventory_hostname }}/{{ project_name }}-key.pem
- {{ kolla_certificates_dir }}/{{ inventory_hostname }}-key.pem
- {{ kolla_certificates_dir }}/{{ project_name }}-key.pem
- {{ kolla_tls_backend_key }}

The default for kolla_certificates_dir is /etc/kolla/certificates.

kolla_tls_backend_cert and kolla_tls_backend_key, default to {{ kolla_certificates_dir }}/backend-cert.pem and {{ kolla_certificates_dir }}/backend-key.pem respectively.

project_name is the name of the OpenStack service, e.g. keystone or cinder.

Note: The back-end TLS cert/key can be the same certificate that is used for the VIP, as long as those certificates are configured to allow requests from both the VIP and internal networks.

By default, the TLS certificate will be verified as trustable by the OpenStack services. Although not recommended for production, it is possible to disable verification of the backend certificate:

```
kolla_verify_tls_backend: "no"
```

Generating a Private Certificate Authority

Note: The certificates generated by Kolla Ansible use a simple Certificate Authority setup and are not suitable for a production deployment. Only certificates signed by a trusted Certificate Authority should be used in a production deployment.

Its not always practical to get a certificate signed by a trusted CA. In a development or internal test OpenStack deployment, it can be useful to generate certificates locally to enable TLS.

For convenience, the kolla-ansible command will generate the necessary certificate files based on the information in the globals.yml configuration file and the inventory file:

```
kolla-ansible -i multinode certificates
```

The certificates role performs the following actions:

1. Generates a test root Certificate Authority
2. Generates the internal/external certificates which are signed by the root CA.
3. If back-end TLS is enabled, generate the back-end certificate signed by the root CA.

The combined certificate and key file haproxy.pem (which is the default value for kolla_external_fqdn_cert) will be generated and stored in the /etc/kolla/certificates/ directory, and a copy of the CA certificate (root.crt) will be stored in the /etc/kolla/certificates/ca/ directory.

Generating your certificates without kolla-ansible

If you want to manage your TLS certificates outside kolla-ansible directly on your hosts, you can do it by setting `kolla_externally_managed_cert` to `true`. This will make kolla-ansible ignore any copy of certificate from the operator to kolla-ansible managed hosts and will keep other configuration options for TLS as is.

If using this option, make sure that all certificates are present on the appropriate hosts in the appropriate location.

4.1.3 ACME http-01 challenge support

This guide describes how to configure Kolla Ansible to enable ACME http-01 challenge support. As of Victoria, Kolla Ansible supports configuring HAProxy Horizon frontend to proxy ACME http-01 challenge requests to selected external (not deployed by Kolla Ansible) ACME client servers. These can be ad-hoc or regular servers. This guide assumes general knowledge of ACME.

Do note ACME supports http-01 challenge only over official HTTP(S) ports, that is 80 (for HTTP) and 443 (for HTTPS). Only Horizon is normally deployed on such port with Kolla Ansible (other services use custom ports). This means that, as of now, running Horizon is mandatory to support ACME http-01 challenge.

How To (External ACME client)

You need to determine the IP address (and port) of the ACME client server used for http-01 challenge (e.g. the host you use to run certbot). The default port is usually `80` (HTTP). Assuming the IP address of that host is `192.168.1.1`, the config would look like the following:

```
enable_horizon: "yes"
acme_client_servers:
  - server certbot 192.168.1.1:80
```

`acme_client_servers` is a list of HAProxy backend server directives. The first parameter is the name of the backend server - it can be arbitrary and is used for logging purposes.

After (re)deploying, you can proceed with running the client to host the http-01 challenge files. Please ensure Horizon frontend responds on the domain you request the certificate for.

To use the newly-generated key-cert pair, follow the *TLS* guide.

4.1.4 MariaDB database backup and restore

Kolla Ansible can facilitate either full or incremental backups of data hosted in MariaDB. It achieves this using Mariabackup, a tool designed to allow for hot backups - an approach which means that consistent backups can be taken without any downtime for your database or your cloud.

Note: By default, backups will be performed on the first node in your Galera cluster or on the MariaDB node itself if you just have the one. Backup files are saved to a dedicated Docker volume - `mariadb_backup` - and its the contents of this that you should target for transferring backups elsewhere.

Enabling Backup Functionality

For backups to work, some reconfiguration of MariaDB is required - this is to enable appropriate permissions for the backup client, and also to create an additional database in order to store backup information.

Firstly, enable backups via `globals.yml`:

```
enable_mariabackup: "yes"
```

Then, kick off a reconfiguration of MariaDB:

```
kolla-ansible -i INVENTORY reconfigure -t mariadb
```

Once that has run successfully, you should then be able to take full and incremental backups as described below.

Backup Procedure

To perform a full backup, run the following command:

```
kolla-ansible -i INVENTORY mariadb_backup
```

Or to perform an incremental backup:

```
kolla-ansible -i INVENTORY mariadb_backup --incremental
```

Kolla doesnt currently manage the scheduling of these backups, so youll need to configure an appropriate scheduler (i.e cron) to run these commands on your behalf should you require regular snapshots of your data. A suggested schedule would be:

- Daily full, retained for two weeks
- Hourly incremental, retained for one day

Backups are performed on your behalf on the designated database node using permissions defined during the configuration step; no password is required to invoke these commands.

Furthermore, backup actions can be triggered from a node with a minimal installation of Kolla Ansible, specifically one which doesnt require a copy of `passwords.yml`. This is of note if youre looking to implement automated backups scheduled via a cron job.

Restoring backups

Owing to the way in which Mariabackup performs hot backups, there are some steps that must be performed in order to prepare your data before it can be copied into place for use by MariaDB. This process is currently manual, but the Kolla Mariabackup image includes the tooling necessary to successfully prepare backups. Two examples are given below.

Full

For a full backup, start a new container using the Mariabackup image with the following options on the first database node:

```
docker run --rm -it --volumes-from mariadb --name dbrestore \
   --volume mariadb_backup:/backup \
   kolla/centos-binary-mariadb-server:wallaby \
   /bin/bash
(dbrestore) $ cd /backup
(dbrestore) $ rm -rf /backup/restore
(dbrestore) $ mkdir -p /backup/restore/full
(dbrestore) $ gunzip mysqlbackup-04-10-2018.qp.xbc.xbs.gz
(dbrestore) $ mbstream -x -C /backup/restore/full/ < mysqlbackup-04-10-
↪2018.qp.xbc.xbs
(dbrestore) $ mariabackup --prepare --target-dir /backup/restore/full
```

Stop the MariaDB instance on all nodes:

```
kolla-ansible -i multinode stop -t mariadb --yes-i-really-really-mean-it
```

Delete the old data files (or move them elsewhere), and copy the backup into place, again on the first node:

```
docker run --rm -it --volumes-from mariadb --name dbrestore \
   --volume mariadb_backup:/backup \
   kolla/centos-binary-mariadb-server:wallaby \
   /bin/bash
(dbrestore) $ rm -rf /var/lib/mysql/*
(dbrestore) $ rm -rf /var/lib/mysql/\.[^\.]*
(dbrestore) $ mariabackup --copy-back --target-dir /backup/restore/full
```

Then you can restart MariaDB with the restored data in place.

For single node deployments:

```
docker start mariadb
docker logs mariadb
81004 15:48:27 mysqld_safe WSREP: Running position recovery with --log_
↪error='/var/lib/mysql//wsrep_recovery.BDTAm8' --pid-file='/var/lib/mysql/
↪/scratch-recover.pid'
181004 15:48:30 mysqld_safe WSREP: Recovered position 9388319e-c7bd-11e8-
↪b2ce-6e9ec70d9926:58
```

For multinode deployment restores, a MariaDB recovery role should be run, pointing to the first node of the cluster:

```
kolla-ansible -i multinode mariadb_recovery -e mariadb_recover_inventory_
↪name=controller1
```

The above procedure is valid also for a disaster recovery scenario. In such case, first copy MariaDB backup file from the external source into `mariadb_backup` volume on the first node of the cluster. From there, use the same steps as mentioned in the procedure above.

Incremental

This starts off similar to the full backup restore procedure above, but we must apply the logs from the incremental backups first of all before doing the final preparation required prior to restore. In the example below, I have a full backup - `mysqlbackup-06-11-2018-1541505206.qp.xbc.xbs`, and an incremental backup, `incremental-11-mysqlbackup-06-11-2018-1541505223.qp.xbc.xbs`.

```
docker run --rm -it --volumes-from mariadb --name dbrestore \
  --volume mariadb_backup:/backup --tmpfs /backup/restore \
  kolla/centos-binary-mariadb-server:wallaby \
  /bin/bash
(dbrestore) $ cd /backup
(dbrestore) $ rm -rf /backup/restore
(dbrestore) $ mkdir -p /backup/restore/full
(dbrestore) $ mkdir -p /backup/restore/inc
(dbrestore) $ gunzip mysqlbackup-06-11-2018-1541505206.qp.xbc.xbs.gz
(dbrestore) $ gunzip incremental-11-mysqlbackup-06-11-2018-1541505223.qp.
↪xbc.xbs.gz
(dbrestore) $ mbstream -x -C /backup/restore/full/ < mysqlbackup-06-11-
↪2018-1541505206.qp.xbc.xbs
(dbrestore) $ mbstream -x -C /backup/restore/inc < incremental-11-
↪mysqlbackup-06-11-2018-1541505223.qp.xbc.xbs
(dbrestore) $ mariabackup --prepare --target-dir /backup/restore/full
(dbrestore) $ mariabackup --prepare --incremental-dir=/backup/restore/inc -
↪-target-dir /backup/restore/full
```

At this point the backup is prepared and ready to be copied back into place, as per the previous example.

4.1.5 Production architecture guide

This guide will help with configuring Kolla to suit production needs. It is meant to answer some questions regarding basic configuration options that Kolla requires. This document also contains other useful pointers.

Node types and services running on them

A basic Kolla inventory consists of several types of nodes, known in Ansible as `groups`.

- Control - Cloud controller nodes which host control services like APIs and databases. This group should have odd number of nodes for quorum.
- Network - Network nodes host Neutron agents along with haproxy / keepalived. These nodes will have a floating ip defined in `kolla_internal_vip_address`.
- Compute - Compute nodes for compute services. This is where guest VMs live.
- Storage - Storage nodes for cinder-volume, LVM or Swift.
- Monitoring - Monitor nodes which host monitoring services.

Network configuration

Interface configuration

In Kolla operators should configure following network interfaces:

- `network_interface` - While it is not used on its own, this provides the required default for other interfaces below.
- `api_interface` - This interface is used for the management network. The management network is the network OpenStack services uses to communicate to each other and the databases. There are known security risks here, so its recommended to make this network internal, not accessible from outside. Defaults to `network_interface`.
- `kolla_external_vip_interface` - This interface is public-facing one. Its used when you want HAProxy public endpoints to be exposed in different network than internal ones. It is mandatory to set this option when `kolla_enable_tls_external` is set to yes. Defaults to `network_interface`.
- `storage_interface` - This is the interface that is used by Swift. This can be heavily utilized so its recommended to use a high speed network fabric. Defaults to `network_interface`.
- `swift_storage_interface` - This interface is used by Swift for storage access traffic. This can be heavily utilized so its recommended to use a high speed network fabric. Defaults to `storage_interface`.
- `swift_replication_interface` - This interface is used by Swift for storage replication traffic. This can be heavily utilized so its recommended to use a high speed network fabric. Defaults to `swift_storage_interface`.
- `tunnel_interface` - This interface is used by Neutron for vm-to-vm traffic over tunneled networks (like VxLan). Defaults to `network_interface`.
- `neutron_external_interface` - This interface is required by Neutron. Neutron will put br-ex on it. It will be used for flat networking as well as tagged vlan networks. Has to be set separately.
- `dns_interface` - This interface is required by Designate and Bind9. Is used by public facing DNS requests and queries to bind9 and designate mDNS services. Defaults to `network_interface`.
- `bifrost_network_interface` - This interface is required by Bifrost. Is used to provision bare metal cloud hosts, require L2 connectivity with the bare metal cloud hosts in order to provide DHCP leases with PXE boot options. Defaults to `network_interface`.

Warning: Ansible facts does not recognize interface names containing dashes, in example `br-ex` or `bond-0` cannot be used because ansible will read them as `br_ex` and `bond_0` respectively.

Address family configuration (IPv4/IPv6)

Starting with the Train release, Kolla Ansible allows operators to deploy the control plane using IPv6 instead of IPv4. Each Kolla Ansible network (as represented by interfaces) provides a choice of two address families. Both internal and external VIP addresses can be configured using an IPv6 address as well. IPv6 is tested on all supported platforms.

Warning: While Kolla Ansible Train requires Ansible 2.6 or later, IPv6 support requires Ansible 2.8 or later due to a bug: https://github.com/ansible/ansible/issues/63227

Note: Currently there is no dual stack support. IPv4 can be mixed with IPv6 only when on different networks. This constraint arises from services requiring common single address family addressing.

For example, `network_address_family` accepts either `ipv4` or `ipv6` as its value and defines the default address family for all networks just like `network_interface` defines the default interface. Analogically, `api_address_family` changes the address family for the API network. Current listing of networks is available in `globals.yml` file.

Note: While IPv6 support introduced in Train is broad, some services are known not to work yet with IPv6 or have some known quirks:

- Bifrost does not support IPv6: https://storyboard.openstack.org/#!/story/2006689
- Docker does not allow IPv6 registry address: https://github.com/moby/moby/issues/39033 - the workaround is to use the hostname
- Ironic DHCP server, dnsmasq, is not currently automatically configured to offer DHCPv6: https://bugs.launchpad.net/kolla-ansible/+bug/1848454

Docker configuration

Because Docker is core dependency of Kolla, proper configuration of Docker can change the experience of Kolla significantly. Following section will highlight several Docker configuration details relevant to Kolla operators.

Storage driver

While the default storage driver should be fine for most users, Docker offers more options to consider. For details please refer to Docker documentation.

Volumes

Kolla puts nearly all of persistent data in Docker volumes. These volumes are created in Docker working directory, which defaults to `/var/lib/docker` directory.

We recommend to ensure that this directory has enough space and is placed on fast disk as it will affect performance of builds, deploys as well as database commits and rabbitmq.

This becomes especially relevant when `enable_central_logging` and `openstack_logging_debug` are both set to true, as fully loaded 130 node cluster produced 30-50GB of logs daily.

High Availability (HA) and scalability

HA is an important topic in production systems. HA concerns itself with redundant instances of services so that the overall service can be provided with close-to-zero interruption in case of failure. Scalability often works hand-in-hand with HA to provide load sharing by the use of load balancers.

OpenStack services

Multinode Kolla Ansible deployments provide HA and scalability for services. OpenStack API endpoints are a prime example here: redundant `haproxy` instances provide HA with `keepalived` while the backends are also deployed redundantly to enable both HA and load balancing.

Other core services

The core non-OpenStack components required by most deployments: the SQL database provided by `mariadb` and message queue provided by `rabbitmq` are also deployed in a HA way. Care has to be taken, however, as unlike previously described services, these have more complex HA mechanisms. The reason for that is that they provide the central, persistent storage of information about the cloud that each other service assumes to have a consistent state (aka integrity). This assumption leads to the requirement of quorum establishment (look up the CAP theorem for greater insight).

Quorum needs a majority vote and hence deploying 2 instances of these do not provide (by default) any HA as a failure of one causes a failure of the other one. Hence the recommended number of instances is `3`, where 1 node failure is acceptable. For scaling purposes and better resilience it is possible to use `5` nodes and have 2 failures acceptable. Note, however, that higher numbers usually provide no benefits due to amount of communication between quorum members themselves and the non-zero probability of the communication medium failure happening instead.

4.1.6 Kollas Deployment Philosophy

Overview

Kolla has an objective to replace the inflexible, painful, resource-intensive deployment process of OpenStack with a flexible, painless, inexpensive deployment process. Often to deploy OpenStack at the 100+ nodes scale, small businesses may require building a team of OpenStack professionals to maintain and

manage the OpenStack deployment. Finding people experienced in OpenStack deployment is very difficult and expensive, resulting in a big barrier for OpenStack adoption. Kolla seeks to remedy this set of problems by simplifying the deployment process while enabling flexible deployment models.

Kolla is a highly opinionated deployment tool out of the box. This permits Kolla to be deployable with the simple configuration of three key/value pairs. As an operators experience with OpenStack grows and the desire to customize OpenStack services increases, Kolla offers full capability to override every OpenStack service configuration option in the deployment.

Why not Template Customization?

The Kolla upstream community does not want to place key/value pairs in the Ansible playbook configuration options that are not essential to obtaining a functional deployment. If the Kolla upstream starts down the path of templating configuration options, the Ansible configuration could conceivably grow to hundreds of configuration key/value pairs which is unmanageable. Further, as new versions of Kolla are released, there would be independent customization available for different versions creating an unsupportable and difficult to document environment. Finally, adding key/value pairs for configuration options creates a situation in which development and release cycles are required in order to successfully add new customizations. Essentially templating in configuration options is not a scalable solution and would result in an inability of the project to execute its mission.

Kollas Solution to Customization

Rather than deal with the customization madness of templating configuration options in Kollas Ansible playbooks, Kolla eliminates all the inefficiencies of existing deployment tools through a simple, tidy design: custom configuration sections.

During deployment of an OpenStack service, a basic set of default configuration options are merged with and overridden by custom ini configuration sections. Kolla deployment customization is that simple! This does create a situation in which the Operator must reference the upstream documentation if a customization is desired in the OpenStack deployment. Fortunately the configuration options documentation is extremely mature and well-formulated.

As an example, consider running Kolla in a virtual machine. In order to launch virtual machines from Nova in a virtual environment, it is necessary to use the QEMU hypervisor, rather than the KVM hypervisor. To achieve this result, simply **`mkdir -p /etc/kolla/config`** and modify the file */etc/kolla/config/nova.conf* with the contents

```
[libvirt]
virt_type=qemu
cpu_mode = none
```

After this change Kolla will use an emulated hypervisor with lower performance. Kolla could have templated this commonly modified configuration option. If Kolla starts down this path, the Kolla project could end with hundreds of config options all of which would have to be subjectively evaluated for inclusion or exclusion in the source tree.

Kollas approach yields a solution which enables complete customization without any upstream maintenance burden. Operators dont have to rely on a subjective approval process for configuration options nor rely on a development/test/release cycle to obtain a desired customization. Instead operators have ultimate freedom to make desired deployment choices immediately without the approval of a third party.

CHAPTER

FIVE

USER GUIDE

5.1 User Guides

5.1.1 Quick Start

This guide provides step by step instructions to deploy OpenStack using Kolla Ansible on bare metal servers or virtual machines.

Recommended reading

Its beneficial to learn basics of both Ansible and Docker before running Kolla Ansible.

Host machine requirements

The host machine must satisfy the following minimum requirements:

- 2 network interfaces
- 8GB main memory
- 40GB disk space

See the support matrix for details of supported host Operating Systems.

Install dependencies

Typically commands that use the system package manager in this section must be run with root privileges.

It is generally recommended to use a virtual environment to install Kolla Ansible and its dependencies, to avoid conflicts with the system site packages. Note that this is independent from the use of a virtual environment for remote execution, which is described in Virtual Environments.

1. For Debian or Ubuntu, update the package index.

   ```
   sudo apt update
   ```

2. Install Python build dependencies:

   For CentOS or RHEL 8, run:

```
sudo dnf install git python3-devel libffi-devel gcc openssl-devel␣
↪python3-libselinux
```

For Debian or Ubuntu, run:

```
sudo apt install git python3-dev libffi-dev gcc libssl-dev
```

Install dependencies using a virtual environment

If not installing Kolla Ansible in a virtual environment, skip this section.

1. Install the virtual environment dependencies.

   For CentOS or RHEL 8, you dont need to do anything.

   For Debian or Ubuntu, run:

   ```
   sudo apt install python3-venv
   ```

2. Create a virtual environment and activate it:

   ```
   python3 -m venv /path/to/venv
   source /path/to/venv/bin/activate
   ```

   The virtual environment should be activated before running any commands that depend on packages installed in it.

3. Ensure the latest version of pip is installed:

   ```
   pip install -U pip
   ```

4. Install Ansible. Kolla Ansible requires at least Ansible `2.9` and supports up to `2.10`.

   ```
   pip install 'ansible<3.0'
   ```

Install dependencies not using a virtual environment

If installing Kolla Ansible in a virtual environment, skip this section.

1. Install `pip`.

   For CentOS or RHEL, run:

   ```
   sudo dnf install python3-pip
   ```

   For Debian or Ubuntu, run:

   ```
   sudo apt install python3-pip
   ```

2. Ensure the latest version of pip is installed:

   ```
   sudo pip3 install -U pip
   ```

3. Install Ansible. Kolla Ansible requires at least Ansible `2.9` and supports up to `2.10`.

   For CentOS or RHEL, run:

   ```
   sudo dnf install ansible
   ```

   For Debian or Ubuntu, run:

   ```
   sudo apt install ansible
   ```

   Note: If the installed Ansible version does not meet the requirements, one can use pip: `sudo pip install -U 'ansible<3.0'`. Beware system package upgrades might interfere with that so it is recommended to uninstall the system package first. One might be better off with the virtual environment method to avoid this pitfall.

Install Kolla-ansible

Install Kolla-ansible for deployment or evaluation

1. Install kolla-ansible and its dependencies using `pip`.

   If using a virtual environment:

   ```
   pip install git+https://opendev.org/openstack/kolla-ansible@stable/
   ↪wallaby
   ```

   If not using a virtual environment:

   ```
   sudo pip3 install git+https://opendev.org/openstack/kolla-
   ↪ansible@stable/wallaby
   ```

2. Create the `/etc/kolla` directory.

   ```
   sudo mkdir -p /etc/kolla
   sudo chown $USER:$USER /etc/kolla
   ```

3. Copy `globals.yml` and `passwords.yml` to `/etc/kolla` directory.

   If using a virtual environment:

   ```
   cp -r /path/to/venv/share/kolla-ansible/etc_examples/kolla/* /etc/
   ↪kolla
   ```

   If not using a virtual environment, run:

   ```
   cp -r /usr/local/share/kolla-ansible/etc_examples/kolla/* /etc/kolla
   ```

4. Copy `all-in-one` and `multinode` inventory files to the current directory.

   If using a virtual environment:

   ```
   cp /path/to/venv/share/kolla-ansible/ansible/inventory/* .
   ```

   If not using a virtual environment, run:

```
cp /usr/local/share/kolla-ansible/ansible/inventory/* .
```

Install Kolla for development

1. Clone `kolla-ansible` repository from git.

```
git clone --branch stable/wallaby https://opendev.org/openstack/kolla-
↪ansible
```

2. Install requirements of `kolla` and `kolla-ansible`:

   If using a virtual environment:

```
pip install ./kolla-ansible
```

   If not using a virtual environment:

```
sudo pip3 install ./kolla-ansible
```

3. Create the `/etc/kolla` directory.

```
sudo mkdir -p /etc/kolla
sudo chown $USER:$USER /etc/kolla
```

4. Copy the configuration files to `/etc/kolla` directory. `kolla-ansible` holds the configuration files (`globals.yml` and `passwords.yml`) in `etc/kolla`.

```
cp -r kolla-ansible/etc/kolla/* /etc/kolla
```

5. Copy the inventory files to the current directory. `kolla-ansible` holds inventory files (`all-in-one` and `multinode`) in the `ansible/inventory` directory.

```
cp kolla-ansible/ansible/inventory/* .
```

Configure Ansible

For best results, Ansible configuration should be tuned for your environment. For example, add the following options to the Ansible configuration file `/etc/ansible/ansible.cfg`:

```
[defaults]
host_key_checking=False
pipelining=True
forks=100
```

Further information on tuning Ansible is available here.

Prepare initial configuration

Inventory

The next step is to prepare our inventory file. An inventory is an Ansible file where we specify hosts and the groups that they belong to. We can use this to define node roles and access credentials.

Kolla Ansible comes with `all-in-one` and `multinode` example inventory files. The difference between them is that the former is ready for deploying single node OpenStack on localhost. If you need to use separate host or more than one node, edit `multinode` inventory:

1. Edit the first section of `multinode` with connection details of your environment, for example:

```
[control]
10.0.0.[10:12] ansible_user=ubuntu ansible_password=foobar ansible_
↪become=true
# Ansible supports syntax like [10:12] - that means 10, 11 and 12.
# Become clause means "use sudo".

[network:children]
control
# when you specify group_name:children, it will use contents of group␣
↪specified.

[compute]
10.0.0.[13:14] ansible_user=ubuntu ansible_password=foobar ansible_
↪become=true

[monitoring]
10.0.0.10
# This group is for monitoring node.
# Fill it with one of the controllers' IP address or some others.

[storage:children]
compute

[deployment]
localhost       ansible_connection=local become=true
# use localhost and sudo
```

   To learn more about inventory files, check Ansible documentation.

2. Check whether the configuration of inventory is correct or not, run:

```
ansible -i multinode all -m ping
```

Note: Distributions might not come with Python pre-installed. That will cause errors in the `ping` module. To quickly install Python with Ansible you can run: for Debian or Ubuntu: `ansible -i multinode all -m raw -a "apt -y install python3"`, and for CentOS or RHEL: `ansible -i multinode all -m raw -a "dnf -y install python3"`.

Kolla passwords

Passwords used in our deployment are stored in `/etc/kolla/passwords.yml` file. All passwords are blank in this file and have to be filled either manually or by running random password generator:

For deployment or evaluation, run:

```
kolla-genpwd
```

For development, run:

```
cd kolla-ansible/tools
./generate_passwords.py
```

Kolla globals.yml

`globals.yml` is the main configuration file for Kolla Ansible. There are a few options that are required to deploy Kolla Ansible:

- Image options

  User has to specify images that are going to be used for our deployment. In this guide DockerHub provided pre-built images are going to be used. To learn more about building mechanism, please refer Building Container Images.

  Kolla provides choice of several Linux distributions in containers:

  - CentOS Stream (`centos`)
  - Ubuntu (`ubuntu`)
  - Debian (`debian`)
  - RHEL (`rhel`, deprecated)

  For newcomers, we recommend to use CentOS Stream 8 or Ubuntu 20.04.

  ```
  kolla_base_distro: "centos"
  ```

  Next type of installation needs to be configured. Choices are:

  binary using repositories like apt or dnf

  source using raw source archives, git repositories or local source directory

  Note: This only affects OpenStack services. Infrastructure services are always binary.

  Note: Source builds are proven to be slightly more reliable than binary.

  ```
  kolla_install_type: "source"
  ```

- Networking

Kolla Ansible requires a few networking options to be set. We need to set network interfaces used by OpenStack.

First interface to set is network_interface. This is the default interface for multiple management-type networks.

```
network_interface: "eth0"
```

Second interface required is dedicated for Neutron external (or public) networks, can be vlan or flat, depends on how the networks are created. This interface should be active without IP address. If not, instances wont be able to access to the external networks.

```
neutron_external_interface: "eth1"
```

To learn more about network configuration, refer Network overview.

Next we need to provide floating IP for management traffic. This IP will be managed by keepalived to provide high availability, and should be set to be *not used* address in management network that is connected to our `network_interface`.

```
kolla_internal_vip_address: "10.1.0.250"
```

- Enable additional services

  By default Kolla Ansible provides a bare compute kit, however it does provide support for a vast selection of additional services. To enable them, set `enable_*` to yes. For example, to enable Block Storage service:

  ```
  enable_cinder: "yes"
  ```

  Kolla now supports many OpenStack services, there is a list of available services. For more information about service configuration, Please refer to the Services Reference Guide.

- Multiple globals files

  For a more granular control, enabling any option from the main `globals.yml` file can now be done using multiple yml files. Simply, create a directory called `globals.d` under `/etc/kolla/` and place all the relevant `*.yml` files in there. The `kolla-ansible` script will, automatically, add all of them as arguments to the `ansible-playbook` command.

  An example use case for this would be if an operator wants to enable cinder and all its options, at a later stage than the initial deployment, without tampering with the existing `globals.yml` file. That can be achieved, using a separate `cinder.yml` file, placed under the `/etc/kolla/globals.d/` directory and adding all the relevant options in there.

- Virtual environment

  It is recommended to use a virtual environment to execute tasks on the remote hosts. This is covered Virtual Environments.

Deployment

After configuration is set, we can proceed to the deployment phase. First we need to setup basic host-level dependencies, like docker.

Kolla Ansible provides a playbook that will install all required services in the correct versions.

The following assumes the use of the `multinode` inventory. If using a different inventory, such as `all-in-one`, replace the `-i` argument accordingly.

- For deployment or evaluation, run:
  1. Bootstrap servers with kolla deploy dependencies:

     ```
     kolla-ansible -i ./multinode bootstrap-servers
     ```

  2. Do pre-deployment checks for hosts:

     ```
     kolla-ansible -i ./multinode prechecks
     ```

  3. Finally proceed to actual OpenStack deployment:

     ```
     kolla-ansible -i ./multinode deploy
     ```

- For development, run:
  1. Bootstrap servers with kolla deploy dependencies:

     ```
     cd kolla-ansible/tools
     ./kolla-ansible -i ../../multinode bootstrap-servers
     ```

  2. Do pre-deployment checks for hosts:

     ```
     ./kolla-ansible -i ../../multinode prechecks
     ```

  3. Finally proceed to actual OpenStack deployment:

     ```
     ./kolla-ansible -i ../../multinode deploy
     ```

When this playbook finishes, OpenStack should be up, running and functional! If error occurs during execution, refer to troubleshooting guide.

Using OpenStack

1. Install the OpenStack CLI client:

   ```
   pip install python-openstackclient -c https://releases.openstack.org/
   ↪constraints/upper/wallaby
   ```

2. OpenStack requires an openrc file where credentials for admin user are set. To generate this file:
   - For deployment or evaluation, run:

     ```
     kolla-ansible post-deploy
     . /etc/kolla/admin-openrc.sh
     ```

   - For development, run:

```
cd kolla-ansible/tools
./kolla-ansible post-deploy
. /etc/kolla/admin-openrc.sh
```

3. Depending on how you installed Kolla Ansible, there is a script that will create example networks, images, and so on.

   Warning: You are free to use the following `init-runonce` script for demo purposes but note it does **not** have to be run in order to use your cloud. Depending on your customisations, it may not work, or it may conflict with the resources you want to create. You have been warned.

   - For deployment or evaluation, run:

     If using a virtual environment:

     ```
     /path/to/venv/share/kolla-ansible/init-runonce
     ```

     If not using a virtual environment:

     ```
     /usr/local/share/kolla-ansible/init-runonce
     ```

   - For development, run:

     ```
     kolla-ansible/tools/init-runonce
     ```

5.1.2 Support Matrix

Supported Operating Systems

Kolla Ansible supports the following host Operating Systems (OS):

Note: CentOS 7 is no longer supported as a host OS. The Train release supports both CentOS 7 and 8, and provides a route for migration. See the Kolla Ansible Train documentation for information on migrating to CentOS 8.

Note: CentOS Linux 8 (as opposed to CentOS Stream 8) is no longer supported as a host OS. The Victoria release will in future support both CentOS Linux 8 and CentOS Stream 8, and provides a route for migration.

- CentOS Stream 8
- Debian Bullseye (11)
- RHEL 8 (deprecated)
- Ubuntu Focal (20.04)

Supported container images

For best results, the base container image distribution should match the host OS distribution. The following values are supported for `kolla_base_distro`:

- `centos`
- `debian`
- `rhel` (deprecated)
- `ubuntu`

For details of which images are supported on which distributions, see the Kolla support matrix.

5.1.3 Virtual Environments

Python virtual environments provide a mechanism for isolating python packages from the system site packages and other virtual environments. Kolla-ansible largely avoids this problem by deploying services in Docker containers, however some python dependencies must be installed both on the Ansible control host and the target hosts.

Ansible Control Host

The kolla-ansible python package and its dependencies may be installed in a python virtual environment on the Ansible control host. For example:

```
python3 -m venv /path/to/venv
source /path/to/venv/bin/activate
pip install -U pip
pip install kolla-ansible
pip install 'ansible<2.10'
deactivate
```

To use the virtual environment, it should first be activated:

```
source /path/to/venv/bin/activate
(venv) kolla-ansible --help
```

The virtual environment can be deactivated when necessary:

```
(venv) deactivate
```

Note that the use of a virtual environment on the Ansible control host does not imply that a virtual environment will be used for execution of Ansible modules on the target hosts.

Target Hosts

Ansible supports remote execution of modules in a python virtual environment via the `ansible_python_interpreter` variable. This may be configured to be the path to a python interpreter installed in a virtual environment. For example:

```
ansible_python_interpreter: /path/to/venv/bin/python
```

Note that `ansible_python_interpreter` cannot be templated.

Kolla-ansible provides support for creating a python virtual environment on the target hosts as part of the `bootstrap-servers` command. The path to the virtualenv is configured via the `virtualenv` variable, and access to site-packages is controlled via `virtualenv_site_packages`. Typically we will need to enable use of system site-packages from within this virtualenv, to support the use of modules such as yum, apt, and selinux, which are not available on PyPI.

When executing kolla-ansible commands other than `bootstrap-servers`, the variable `ansible_python_interpreter` should be set to the python interpreter installed in `virtualenv`.

5.1.4 Multinode Deployment of Kolla

Deploy a registry

A Docker registry is a locally hosted registry that replaces the need to pull from the Docker Hub to get images. Kolla can function with or without a local registry, however for a multinode deployment some type of registry is mandatory. Only one registry must be deployed, although HA features exist for registry services.

The Docker registry prior to version 2.3 has extremely bad performance because all container data is pushed for every image rather than taking advantage of Docker layering to optimize push operations. For more information reference pokey registry. The Kolla community recommends using registry 2.3 or later.

The registry may be configured either as a local registry with support for storing images, or as a pull-through cache for Docker hub.

Option 1: local registry

```
docker run -d \
 --name registry \
 --restart=always \
 -p 4000:5000 \
 -v registry:/var/lib/registry \
 registry:2
```

Here we are using port 4000 to avoid a conflict with Keystone. If the registry is not running on the same host as Keystone, the `-p` argument may be omitted.

Edit `globals.yml` and add the following, where `192.168.1.100:4000` is the IP address and port on which the registry is listening:

```
docker_registry: 192.168.1.100:4000
```

Option 2: registry mirror

The Docker registry can be configured as a pull through cache to proxy the official Kolla images hosted in Docker Hub. In order to configure the local registry as a pull through cache, pass the environment variable `REGISTRY_PROXY_REMOTEURL` through to the registry container. Pushing to a registry configured as a pull-through cache is unsupported. For more information, Reference the Docker Documentation.

```
docker run -d \
 --name registry \
 --restart=always \
 -p 4000:5000 \
 -v registry:/var/lib/registry \
 -e REGISTRY_PROXY_REMOTEURL=https://registry-1.docker.io \
 registry:2
```

Edit `globals.yml` and add the following, where `192.168.1.100:4000` is the IP address and port on which the registry is listening:

```
docker_custom_config:
  registry-mirrors:
    - http://192.168.1.100:4000
```

Edit the Inventory File

The ansible inventory file contains all the information needed to determine what services will land on which hosts. Edit the inventory file in the Kolla Ansible directory `ansible/inventory/multinode`. If Kolla Ansible was installed with pip, it can be found in `/usr/share/kolla-ansible`.

Add the IP addresses or hostnames to a group and the services associated with that group will land on that host. IP addresses or hostnames must be added to the groups control, network, compute, monitoring and storage. Also, define additional behavioral inventory parameters such as `ansible_ssh_user`, `ansible_become` and `ansible_private_key_file/ansible_ssh_pass` which controls how ansible interacts with remote hosts.

Note: Ansible uses SSH to connect the deployment host and target hosts. For more information about SSH authentication please reference Ansible documentation.

```
# These initial groups are the only groups required to be modified. The
# additional groups are for more control of the environment.
[control]
# These hostname must be resolvable from your deployment host
control01      ansible_ssh_user=<ssh-username> ansible_become=True ansible_
↪private_key_file=<path/to/private-key-file>
192.168.122.24 ansible_ssh_user=<ssh-username> ansible_become=True ansible_
↪private_key_file=<path/to/private-key-file>
```

Note: Additional inventory parameters might be required according to your environment setup. Reference Ansible Documentation for more information.

For more advanced roles, the operator can edit which services will be associated in with each group. Keep in mind that some services have to be grouped together and changing these around can break your deployment:

```
[kibana:children]
control

[elasticsearch:children]
control

[haproxy:children]
network
```

Host and group variables

Typically, Kolla Ansible configuration is stored in the `globals.yml` file. Variables in this file apply to all hosts. In an environment with multiple hosts, it may become necessary to have different values for variables for different hosts. A common example of this is for network interface configuration, e.g. `api_interface`.

Ansibles host and group variables can be assigned in a variety of ways. Simplest is in the inventory file itself:

```
# Host with a host variable.
[control]
control01 api_interface=eth3

# Group with a group variable.
[control:vars]
api_interface=eth4
```

This can quickly start to become difficult to maintain, so it may be preferable to use `host_vars` or `group_vars` directories containing YAML files with host or group variables:

```
inventory/
  group_vars/
    control
  host_vars/
    control01
  multinode
```

Ansibles variable precedence rules are quite complex, but it is worth becoming familiar with them if using host and group variables. The playbook group variables in `ansible/group_vars/all.yml` define global defaults, and these take precedence over variables defined in an inventory file and inventory `group_vars/all`, but not over inventory `group_vars/*`. Variables in extra files (`globals.yml`) have the highest precedence, so any variables which must differ between hosts must not be in `globals.yml`.

Deploying Kolla

Note: If there are multiple keepalived clusters running within the same layer 2 network, edit the file `/etc/kolla/globals.yml` and specify a `keepalived_virtual_router_id`. The `keepalived_virtual_router_id` should be unique and belong to the range 0 to 255.

Note: If glance is configured to use `file` as backend, only one `glance_api` container will be started. `file` is enabled by default when no other backend is specified in `/etc/kolla/globals.yml`.

First, check that the deployment targets are in a state where Kolla may deploy to them:

```
kolla-ansible prechecks -i <path/to/multinode/inventory/file>
```

Note: RabbitMQ doesnt work with IP addresses, hence the IP address of `api_interface` should be resolvable by hostnames to make sure that all RabbitMQ Cluster hosts can resolve each others hostnames beforehand.

Run the deployment:

```
kolla-ansible deploy -i <path/to/multinode/inventory/file>
```

5.1.5 Multiple Regions Deployment with Kolla

This section describes how to perform a basic multiple region deployment with Kolla. A basic multiple region deployment consists of separate OpenStack installations in two or more regions (RegionOne, RegionTwo,) with a shared Keystone and Horizon. The rest of this documentation assumes Keystone and Horizon are deployed in RegionOne, and other regions have access to the admin endpoint (for example, `kolla_internal_fqdn`) of RegionOne. It also assumes that the operator knows the name of all OpenStack regions in advance, and considers as many Kolla deployments as there are regions.

There is specifications of multiple regions deployment at Multi Region Support for Heat.

Deployment of the first region with Keystone and Horizon

Deployment of the first region results in a typical Kolla deployment whether it is an *all-in-one* or *multinode* deployment (see *Quick Start*). It only requires slight modifications in the `/etc/kolla/globals.yml` configuration file. First of all, ensure that Keystone and Horizon are enabled:

```
enable_keystone: "yes"
enable_horizon: "yes"
```

Then, change the value of `multiple_regions_names` to add names of other regions. In this example, we consider two regions. The current one, formerly known as RegionOne, that is hidden behind `openstack_region_name` variable, and the RegionTwo:

```
openstack_region_name: "RegionOne"
multiple_regions_names:
    - "{{ openstack_region_name }}"
    - "RegionTwo"
```

Note: Kolla uses these variables to create necessary endpoints into Keystone so that services of other regions can access it. Kolla also updates the Horizon `local_settings` to support multiple regions.

Finally, note the value of `kolla_internal_fqdn` and run `kolla-ansible`. The `kolla_internal_fqdn` value will be used by other regions to contact Keystone. For the sake of this example, we assume the value of `kolla_internal_fqdn` is `10.10.10.254`.

Deployment of other regions

Deployment of other regions follows an usual Kolla deployment except that OpenStack services connect to the RegionOnes Keystone. This implies to update the `/etc/kolla/globals.yml` configuration file to tell Kolla how to reach Keystone. In the following, `kolla_internal_fqdn_r1` refers to the value of `kolla_internal_fqdn` in RegionOne:

```
kolla_internal_fqdn_r1: 10.10.10.254

keystone_admin_url: "{{ admin_protocol }}://{{ kolla_internal_fqdn_r1 }}:{
↪{ keystone_admin_port }}"
keystone_internal_url: "{{ internal_protocol }}://{{ kolla_internal_fqdn_
↪r1 }}:{{ keystone_public_port }}"

openstack_auth:
    auth_url: "{{ admin_protocol }}://{{ kolla_internal_fqdn_r1 }}:{{␣
↪keystone_admin_port }}"
    username: "admin"
    password: "{{ keystone_admin_password }}"
    project_name: "admin"
    domain_name: "default"
```

Note: If the `kolla_internal_vip_address` and/or the `kolla_external_vip_address` reside on the same subnet as `kolla_internal_fqdn_r1`, you should set the `keepalived_virtual_router_id` value in the `/etc/kolla/globals.yml` to a unique number.

Configuration files of cinder,nova,neutron,glance have to be updated to contact RegionOnes Keystone. Fortunately, Kolla allows you to override all configuration files at the same time thanks to the `node_custom_config` variable (see *OpenStack Service Configuration in Kolla*). To do so, create a `global.conf` file with the following content:

```
[keystone_authtoken]
www_authenticate_uri = {{ keystone_internal_url }}
auth_url = {{ keystone_admin_url }}
```

The Placement API section inside the nova configuration file also has to be updated to contact RegionOnes Keystone. So create, in the same directory, a `nova.conf` file with below content:

```
[placement]
auth_url = {{ keystone_admin_url }}
```

The Heat section inside the configuration file also has to be updated to contact RegionOnes Keystone. So create, in the same directory, a `heat.conf` file with below content:

```
[trustee]
www_authenticate_uri = {{ keystone_internal_url }}
auth_url = {{ keystone_internal_url }}

[ec2authtoken]
www_authenticate_uri = {{ keystone_internal_url }}

[clients_keystone]
www_authenticate_uri = {{ keystone_internal_url }}
```

The Ceilometer section inside the configuration file also has to be updated to contact RegionOnes Keystone. So create, in the same directory, a `ceilometer.conf` file with below content:

```
[service_credentials]
auth_url = {{ keystone_internal_url }}
```

And link the directory that contains these files into the `/etc/kolla/globals.yml`:

```
node_custom_config: path/to/the/directory/of/global&nova_conf/
```

Also, change the name of the current region. For instance, RegionTwo:

```
openstack_region_name: "RegionTwo"
```

Finally, disable the deployment of Keystone and Horizon that are unnecessary in this region and run `kolla-ansible`:

```
enable_keystone: "no"
enable_horizon: "no"
```

The configuration is the same for any other region.

5.1.6 Operating Kolla

Versioning

Kolla uses the `x.y.z` semver nomenclature for naming versions. Kollas initial Pike release was `5.0.0` and the initial Queens release is `6.0.0`. The Kolla community commits to release z-stream updates every 45 days that resolve defects in the stable version in use and publish those images to the Docker Hub registry.

To prevent confusion, the Kolla community recommends using an alpha identifier `x.y.z.a` where `a` represents any customization done on the part of the operator. For example, if an operator intends to modify one of the Docker files or the repos from the originals and build custom images for the Pike version, the operator should start with version 5.0.0.0 and increase alpha for each release. Alpha tag usage is at discretion of the operator. The alpha identifier could be a number as recommended or a string of the operators choosing.

To customize the version number uncomment `openstack_release` in globals.yml and specify the desired version number or name (e.g. `victoria`, `wallaby`). If `openstack_release` is not specified, Kolla will deploy or upgrade using the version number information contained in the kolla-ansible package.

Upgrade procedure

Note: If you have set `enable_cells` to `yes` then you should read the upgrade notes in the *Nova cells guide*.

Kollas strategy for upgrades is to never make a mess and to follow consistent patterns during deployment such that upgrades from one environment to the next are simple to automate.

Kolla implements a one command operation for upgrading an existing deployment consisting of a set of containers and configuration data to a new deployment.

Limitations and Recommendations

Note: Varying degrees of success have been reported with upgrading the libvirt container with a running virtual machine in it. The libvirt upgrade still needs a bit more validation, but the Kolla community feels confident this mechanism can be used with the correct Docker storage driver.

Note: Because of system technical limitations, upgrade of a libvirt container when using software emulation (`virt_type = qemu` in nova.conf), does not work at all. This is acceptable because KVM is the recommended virtualization driver to use with Nova.

Note: Please note that when the `use_preconfigured_databases` flag is set to `"yes"`, you need to have the `log_bin_trust_function_creators` set to `1` by your database administrator before performing the upgrade.

Note: If you have separate keys for nova and cinder, please be sure to set `ceph_nova_keyring: ceph.client.nova.keyring` and `ceph_nova_user: nova` in `/etc/kolla/globals.yml`

Ubuntu Focal 20.04

The Victoria release adds support for Ubuntu Focal 20.04 as a host operating system. Ubuntu users upgrading from Ussuri should first upgrade OpenStack containers to Victoria, which uses the Ubuntu Focal 20.04 base container image. Hosts should then be upgraded to Ubuntu Focal 20.04.

CentOS Stream 8

The Wallaby release adds support for CentOS Stream 8 as a host operating system. CentOS Stream 8 support will also be added to a Victoria stable release. CentOS Linux users upgrading from Victoria should first migrate hosts and container images from CentOS Linux to CentOS Stream before upgrading to Wallaby.

Preparation

While there may be some cases where it is possible to upgrade by skipping this step (i.e. by upgrading only the `openstack_release` version) - generally, when looking at a more comprehensive upgrade, the kolla-ansible package itself should be upgraded first. This will include reviewing some of the configuration and inventory files. On the operator/master node, a backup of the `/etc/kolla` directory may be desirable.

If upgrading to `wallaby`, upgrade the kolla-ansible package:

```
pip install --upgrade git+https://opendev.org/openstack/kolla-
↪ansible@stable/wallaby
```

If this is a minor upgrade, and you do not wish to upgrade kolla-ansible itself, you may skip this step.

The inventory file for the deployment should be updated, as the newer sample inventory files may have updated layout or other relevant changes. Use the newer `wallaby` one as a starting template, and merge your existing inventory layout into a copy of the one from here:

```
/usr/share/kolla-ansible/ansible/inventory/
```

In addition the `wallaby` sample configuration files should be taken from:

```
# CentOS
/usr/share/kolla-ansible/etc_examples/kolla

# Ubuntu
/usr/local/share/kolla-ansible/etc_examples/kolla
```

At this stage, files that are still at the previous version and need manual updating are:

- `/etc/kolla/globals.yml`
- `/etc/kolla/passwords.yml`

For `globals.yml` relevant changes should be merged into a copy of the new template, and then replace the file in `/etc/kolla` with the updated version. For `passwords.yml`, see the `kolla-mergepwd` instructions in *Tips and Tricks*.

For the kolla docker images, the `openstack_release` is updated to `wallaby`:

```
openstack_release: wallaby
```

Once the kolla release, the inventory file, and the relevant configuration files have been updated in this way, the operator may first want to pull down the images to stage the `wallaby` versions. This can be done safely ahead of time, and does not impact the existing services. (optional)

Run the command to pull the `wallaby` images for staging:

```
kolla-ansible pull
```

At a convenient time, the upgrade can now be run (it will complete more quickly if the images have been staged ahead of time).

Perform the Upgrade

To perform the upgrade:

```
kolla-ansible upgrade
```

After this command is complete the containers will have been recreated from the new images.

Tips and Tricks

Kolla Ansible CLI

When running the `kolla-ansible` CLI, additional arguments may be passed to `ansible-playbook` via the `EXTRA_OPTS` environment variable.

`kolla-ansible -i INVENTORY deploy` is used to deploy and start all Kolla containers.

`kolla-ansible -i INVENTORY destroy` is used to clean up containers and volumes in the cluster.

`kolla-ansible -i INVENTORY mariadb_recovery` is used to recover a completely stopped mariadb cluster.

`kolla-ansible -i INVENTORY prechecks` is used to check if all requirements are meet before deploy for each of the OpenStack services.

`kolla-ansible -i INVENTORY post-deploy` is used to do post deploy on deploy node to get the admin openrc file.

`kolla-ansible -i INVENTORY pull` is used to pull all images for containers.

`kolla-ansible -i INVENTORY reconfigure` is used to reconfigure OpenStack service.

`kolla-ansible -i INVENTORY upgrade` is used to upgrades existing OpenStack Environment.

`kolla-ansible -i INVENTORY check` is used to do post-deployment smoke tests.

`kolla-ansible -i INVENTORY stop` is used to stop running containers.

`kolla-ansible -i INVENTORY deploy-containers` is used to check and if necessary update containers, without generating configuration.

`kolla-ansible -i INVENTORY prune-images` is used to prune orphaned Docker images on hosts.

Note: In order to do smoke tests, requires `kolla_enable_sanity_checks=yes`.

Passwords

The following commands manage the Kolla Ansible passwords file.

`kolla-mergepwd --old OLD_PASSWDS --new NEW_PASSWDS --final FINAL_PASSWDS` is used to merge passwords from old installation with newly generated passwords during upgrade of Kolla release. The workflow is:

1. Save old passwords from `/etc/kolla/passwords.yml` into `passwords.yml.old`.
2. Generate new passwords via `kolla-genpwd` as `passwords.yml.new`.
3. Merge `passwords.yml.old` and `passwords.yml.new` into `/etc/kolla/passwords.yml`.

For example:

```
mv /etc/kolla/passwords.yml passwords.yml.old
cp kolla-ansible/etc/kolla/passwords.yml passwords.yml.new
kolla-genpwd -p passwords.yml.new
kolla-mergepwd --old passwords.yml.old --new passwords.yml.new --final /
↪etc/kolla/passwords.yml
```

Note: `kolla-mergepwd`, by default, keeps old, unused passwords intact. To alter this behavior, and remove such entries, use the `--clean` argument when invoking `kolla-mergepwd`.

Tools

Kolla ships with several utilities intended to facilitate ease of operation.

`tools/cleanup-containers` is used to remove deployed containers from the system. This can be useful when you want to do a new clean deployment. It will preserve the registry and the locally built images in the registry, but will remove all running Kolla containers from the local Docker daemon. It also removes the named volumes.

`tools/cleanup-host` is used to remove remnants of network changes triggered on the Docker host when the neutron-agents containers are launched. This can be useful when you want to do a new clean deployment, particularly one changing the network topology.

`tools/cleanup-images --all` is used to remove all Docker images built by Kolla from the local Docker cache.

5.1.7 Adding and removing hosts

This page discusses how to add and remove nodes from an existing cluster. The procedure differs depending on the type of nodes being added or removed, which services are running, and how they are configured. Here we will consider two types of nodes - controllers and compute nodes. Other types of nodes will need consideration.

Any procedure being used should be tested before being applied in a production environment.

Adding new hosts

Adding new controllers

The *bootstrap-servers command* can be used to prepare the new hosts that are being added to the system. It adds an entry to `/etc/hosts` for the new hosts, and some services, such as RabbitMQ, require entries to exist for all controllers on every controller. If using a `--limit` argument, ensure that all controllers are included, e.g. via `--limit control`. Be aware of the *potential issues* with running `bootstrap-servers` on an existing system.

```
kolla-ansible -i <inventory> bootstrap-servers [ --limit <limit> ]
```

Pull down container images to the new hosts. The `--limit` argument may be used and only needs to include the new hosts.

```
kolla-ansible -i <inventory> pull [ --limit <limit> ]
```

Deploy containers to the new hosts. If using a `--limit` argument, ensure that all controllers are included, e.g. via `--limit control`.

```
kolla-ansible -i <inventory> deploy [ --limit <limit> ]
```

The new controllers are now deployed. It is recommended to perform testing of the control plane at this point to verify that the new controllers are functioning correctly.

Some resources may not be automatically balanced onto the new controllers. It may be helpful to manually rebalance these resources onto the new controllers. Examples include networks hosted by Neutron DHCP agent, and routers hosted by Neutron L3 agent. The *removing-existing-controllers* section provides an example of how to do this.

Adding new compute nodes

The *bootstrap-servers command*, can be used to prepare the new hosts that are being added to the system. Be aware of the *potential issues* with running `bootstrap-servers` on an existing system.

```
kolla-ansible -i <inventory> bootstrap-servers [ --limit <limit> ]
```

Pull down container images to the new hosts. The `--limit` argument may be used and only needs to include the new hosts.

```
kolla-ansible -i <inventory> pull [ --limit <limit> ]
```

Deploy containers on the new hosts. The `--limit` argument may be used and only needs to include the new hosts.

```
kolla-ansible -i <inventory> deploy [ --limit <limit> ]
```

The new compute nodes are now deployed. It is recommended to perform testing of the compute nodes at this point to verify that they are functioning correctly.

Server instances are not automatically balanced onto the new compute nodes. It may be helpful to live migrate some server instances onto the new hosts.

```
openstack server migrate <server> --live-migration --host <target host> --
↪os-compute-api-version 2.30
```

Alternatively, a service such as Watcher may be used to do this automatically.

Removing existing hosts

Removing existing controllers

When removing controllers or other hosts running clustered services, consider whether enough hosts remain in the cluster to form a quorum. For example, in a system with 3 controllers, only one should be removed at a time. Consider also the effect this will have on redundancy.

Before removing existing controllers from a cluster, it is recommended to move resources they are hosting. Here we will cover networks hosted by Neutron DHCP agent and routers hosted by Neutron L3 agent. Other actions may be necessary, depending on your environment and configuration.

For each host being removed, find Neutron routers on that host and move them. Disable the L3 agent. For example:

```
l3_id=$(openstack network agent list --host <host> --agent-type l3 -f␣
↪value -c ID)
target_l3_id=$(openstack network agent list --host <target host> --agent-
↪type l3 -f value -c ID)
openstack router list --agent $l3_id -f value -c ID | while read router; do
  openstack network agent remove router $l3_id $router --l3
  openstack network agent add router $target_l3_id $router --l3
done
openstack network agent set $l3_id --disable
```

Repeat for DHCP agents:

```
dhcp_id=$(openstack network agent list --host <host> --agent-type dhcp -f␣
↪value -c ID)
target_dhcp_id=$(openstack network agent list --host <target host> --agent-
↪type dhcp -f value -c ID)
openstack network list --agent $dhcp_id -f value -c ID | while read␣
↪network; do
  openstack network agent remove network $dhcp_id $network --dhcp
  openstack network agent add network $target_dhcp_id $network --dhcp
done
```

Stop all services running on the hosts being removed:

```
kolla-ansible -i <inventory> stop --yes-i-really-really-mean-it [ --limit
↪<limit> ]
```

Remove the hosts from the Ansible inventory.

Reconfigure the remaining controllers to update the membership of clusters such as MariaDB and RabbitMQ. Use a suitable limit, such as `--limit control`.

```
kolla-ansible -i <inventory> deploy [ --limit <limit> ]
```

Perform testing to verify that the remaining cluster hosts are operating correctly.

For each host, clean up its services:

```
openstack network agent list --host <host> -f value -c ID | while read id;␣
↪do
  openstack network agent delete $id
done

openstack compute service list --os-compute-api-version 2.53 --host <host>␣
↪-f value -c ID | while read id; do
  openstack compute service delete --os-compute-api-version 2.53 $id
done
```

Removing existing compute nodes

When removing compute nodes from a system, consider whether there is capacity to host the running workload on the remaining compute nodes. Include overhead for failures that may occur.

Before removing compute nodes from a system, it is recommended to migrate or destroy any instances that they are hosting.

For each host, disable the compute service to ensure that no new instances are scheduled to it.

```
openstack compute service set <host> nova-compute --disable
```

If possible, live migrate instances to another host.

```
openstack server list --host <host> -f value -c ID | while read server; do
  openstack server migrate --live-migration $server
done
```

Verify that the migrations were successful.

Stop all services running on the hosts being removed:

```
kolla-ansible -i <inventory> stop --yes-i-really-really-mean-it [ --limit
↪<limit> ]
```

Remove the hosts from the Ansible inventory.

Perform testing to verify that the remaining cluster hosts are operating correctly.

For each host, clean up its services:

```
openstack network agent list --host <host> -f value -c ID | while read id;␣
↪do
  openstack network agent delete $id
done

openstack compute service list --os-compute-api-version 2.53 --host <host>␣
↪-f value -c ID | while read id; do
  openstack compute service delete --os-compute-api-version 2.53 $id
done
```

5.1.8 Kolla Security

Non Root containers

The OpenStack services, with a few exceptions, run as non root inside of Kollas containers. Kolla uses the Docker provided `USER` flag to set the appropriate user for each service.

SELinux

The state of SELinux in Kolla is a work in progress. The short answer is you must disable it until selinux polices are written for the Docker containers.

To understand why Kolla needs to set certain selinux policies for services that you wouldnt expect to need them (rabbitmq, mariadb, glance and so on) we must take a step back and talk about Docker.

Docker has not had the concept of persistent containerized data until recently. This means when a container is run the data it creates is destroyed when the container goes away, which is obviously no good in the case of upgrades.

It was suggested data containers could solve this issue by only holding data if they were never recreated, leading to a scary state where you could lose access to your data if the wrong command was executed. The real answer to this problem came in Docker 1.9 with the introduction of named volumes. You could now address volumes directly by name removing the need for so called **data containers** all together.

Another solution to the persistent data issue is to use a host bind mount which involves making, for sake of example, host directory `var/lib/mysql` available inside the container at `var/lib/mysql`. This absolutely solves the problem of persistent data, but it introduces another security issue, permissions. With this host bind mount solution the data in `var/lib/mysql` will be owned by the mysql user in the container. Unfortunately, that mysql user in the container could have any UID/GID and thats who will own the data outside the container introducing a potential security risk. Additionally, this method dirties the host and requires host permissions to the directories to bind mount.

The solution Kolla chose is named volumes.

Why does this matter in the case of selinux? Kolla does not run the process. It is launching as root in most cases. So glance-api is run as the glance user, and mariadb is run as the mysql user, and so on. When mounting a named volume in the location that the persistent data will be stored it will be owned by the root user and group. The mysql user has no permissions to write to this folder now. What Kolla does is allow a select few commands to be run with sudo as the mysql user. This allows the mysql user to chown a specific, explicit directory and store its data in a named volume without the security risk and other downsides of host bind mounts. The downside to this is selinux blocks those sudo commands and it will do so until we make explicit policies to allow those operations.

Kolla-ansible users

Prior to Queens, when users want to connect using non-root user, they must add extra option `ansible_become=True` which is inconvenient and add security risk. In Queens, almost all services have support for escalation for only necessary tasks. In Rocky, all services have this capability, so users do not need to add `ansible_become` option if connection user has passwordless sudo capability.

Prior to Rocky, `ansible_user` (the user which Ansible uses to connect via SSH) is default configuration owner and group in target nodes. From Rocky release, Kolla support connection using any user which has passwordless sudo capability. For setting custom owner user and group, user can set `config_owner_user` and `config_owner_group` in `globals.yml`.

5.1.9 Ansible tuning

In this section we cover some options for tuning Ansible for performance and scale.

SSH pipelining

SSH pipelining is disabled in Ansible by default, but is generally safe to enable, and provides a reasonable performance improvement.

Listing 1: `ansible.cfg`

```
[ssh_connection]
pipelining = True
```

Forks

By default Ansible executes tasks using a fairly conservative 5 process forks. This limits the parallelism that allows Ansible to scale. Most Ansible control hosts will be able to handle far more forks than this. You will need to experiment to find out the CPU, memory and IO limits of your machine.

For example, to increase the number of forks to 20:

Listing 2: `ansible.cfg`

```
[defaults]
forks = 20
```

Fact caching

By default, Ansible gathers facts for each host at the beginning of every play, unless `gather_facts` is set to `false`. With a large number of hosts this can result in a significant amount of time spent gathering facts.

One way to improve this is through Ansibles support for fact caching. In order to make this work with Kolla Ansible, it is necessary to change Ansibles gathering configuration option to `smart`.

Example

In the following example we configure Kolla Ansible to use fact caching using the jsonfile cache plugin.

Listing 3: `ansible.cfg`

```
[defaults]
gathering = smart
fact_caching = jsonfile
fact_caching_connection = /tmp/ansible-facts
```

You may also wish to set the expiration timeout for the cache via `[defaults] fact_caching_timeout`.

Fact variable injection

By default, Ansible injects a variable for every fact, prefixed with `ansible_`. This can result in a large number of variables for each host, which at scale can incur a performance penalty. Ansible provides a configuration option that can be set to `False` to prevent this injection of facts. In this case, facts should be referenced via `ansible_facts.<fact>`. In recent releases of Kolla Ansible, facts are referenced via `ansible_facts`, allowing users to disable fact variable injection.

Listing 4: `ansible.cfg`

```
[defaults]
inject_facts_as_vars = False
```

Fact filtering

Ansible facts filtering can be used to speed up Ansible. Environments with many network interfaces on the network and compute nodes can experience very slow processing with Kolla Ansible. This happens due to the processing of the large per-interface facts with each task. To avoid storing certain facts, we can use the `kolla_ansible_setup_filter` variable, which is used as the `filter` argument to the `setup` module. For example, to avoid collecting facts for virtual interfaces beginning with q or t:

```
kolla_ansible_setup_filter: "ansible_[!qt]*"
```

This causes Ansible to collect but not store facts matching that pattern, which includes the virtual interface facts. Currently we are not referencing other facts matching the pattern within Kolla Ansible. Note that including the `ansible_` prefix causes meta facts `module_setup` and `gather_subset` to be filtered, but this seems to be the only way to get a good match on the interface facts.

The exact improvement will vary, but has been reported to be as large as 18x on systems with many virtual interfaces.

Fact gathering subsets

It is also possible to configure which subsets of facts are gathered, via `kolla_ansible_setup_gather_subset`, which is used as the `gather_subset` argument to the `setup` module. For example, if one wants to avoid collecting facts via facter:

```
kolla_ansible_setup_gather_subset: "all,!facter"
```

5.1.10 Troubleshooting Guide

Failures

If Kolla fails, often it is caused by a CTRL-C during the deployment process or a problem in the `globals.yml` configuration.

Note: In some countries like China, Kolla might fail due to unable to pull images from Docker Hub. There is a workround to solve this issue:

```
mkdir -p /etc/docker
tee /etc/docker/daemon.json <<-'EOF'
{
  "registry-mirrors": ["https://registry.docker-cn.com"]
}
EOF
systemctl restart docker
```

To correct the problem where Operators have a misconfigured environment, the Kolla community has added a precheck feature which ensures the deployment targets are in a state where Kolla may deploy to them. To run the prechecks:

```
kolla-ansible prechecks
```

If a failure during deployment occurs it nearly always occurs during evaluation of the software. Once the Operator learns the few configuration options required, it is highly unlikely they will experience a failure in deployment.

Deployment may be run as many times as desired, but if a failure in a bootstrap task occurs, a further deploy action will not correct the problem. In this scenario, Kollas behavior is undefined.

The fastest way during to recover from a deployment failure is to remove the failed deployment:

```
kolla-ansible destroy -i <<inventory-file>>
```

Any time the tags of a release change, it is possible that the container implementation from older versions wont match the Ansible playbooks in a new version. If running multinode from a registry, each nodes Docker image cache must be refreshed with the latest images before a new deployment can occur. To refresh the docker cache from the local Docker registry:

```
kolla-ansible pull
```

Debugging Kolla

The status of containers after deployment can be determined on the deployment targets by executing:

```
docker ps -a
```

If any of the containers exited, this indicates a bug in the container. Please seek help by filing a launchpad bug or contacting the developers via IRC.

The logs can be examined by executing:

```
docker exec -it fluentd bash
```

The logs from all services in all containers may be read from `/var/log/kolla/SERVICE_NAME`

If the stdout logs are needed, please run:

```
docker logs <container-name>
```

Note that most of the containers dont log to stdout so the above command will provide no information.

To learn more about Docker command line operation please refer to Docker documentation.

The log volume kolla_logs is linked to `/var/log/kolla` on the host. You can find all kolla logs in there.

```
readlink -f /var/log/kolla
/var/lib/docker/volumes/kolla_logs/_data
```

When `enable_central_logging` is enabled, to view the logs in a web browser using Kibana, go to `http://<kolla_internal_vip_address>:<kibana_server_port>` or `http://<kolla_external_vip_address>:<kibana_server_port>`. Authenticate using `<kibana_user>` and `<kibana_password>`.

The values `<kolla_internal_vip_address>`, `<kolla_external_vip_address>` `<kibana_server_port>` and `<kibana_user>` can be found in `<kolla_install_path>/kolla/ansible/group_vars/all.yml` or if the default values are overridden, in `/etc/kolla/globals.yml`. The value of `<kibana_password>` can be found in `/etc/kolla/passwords.yml`.

CHAPTER

SIX

REFERENCE

6.1 Projects Deployment Configuration Reference

6.1.1 Compute

This section describes configuring nova hypervisors and compute services.

Libvirt - Nova Virtualisation Driver

Overview

Libvirt is the most commonly used virtualisation driver in OpenStack. It uses libvirt, backed by QEMU and when available, KVM. Libvirt is executed in the `nova_libvirt` container.

Hardware Virtualisation

Two values are supported for `nova_compute_virt_type` with libvirt - `kvm` and `qemu`, with `kvm` being the default.

For optimal performance, `kvm` is preferable, since many aspects of virtualisation can be offloaded to hardware. If it is not possible to enable hardware virtualisation (e.g. Virtualisation Technology (VT) BIOS configuration on Intel systems), `qemu` may be used to provide less performant software-emulated virtualisation.

SASL Authentication

The default configuration of Kolla Ansible is to run libvirt over TCP, authenticated with SASL. This should not be considered as providing a secure, encrypted channel, since the username/password SASL mechanisms available for TCP are no longer considered cryptographically secure. However, it does at least provide some authentication for the libvirt API. For a more secure encrypted channel, use *libvirt TLS*.

SASL is enabled according to the `libvirt_enable_sasl` flag, which defaults to `true`.

The username is configured via `libvirt_sasl_authname`, and defaults to `nova`. The password is configured via `libvirt_sasl_password`, and is generated with other passwords using `kolla-mergepwd` and `kolla-genpwd` and stored in `passwords.yml`.

The list of enabled authentication mechanisms is configured via `libvirt_sasl_mech_list`, and defaults to `["SCRAM-SHA-256"]` if libvirt TLS is enabled, or `["DIGEST-MD5"]` otherwise.

Libvirt TLS

The default configuration of Kolla Ansible is to run libvirt over TCP, with SASL authentication. As long as one takes steps to protect who can access the network this works well. However, in a less trusted environment one may want to use encryption when accessing the libvirt API. To do this we can enable TLS for libvirt and make nova use it. Mutual TLS is configured, providing authentication of clients via certificates. SASL authentication provides a further level of security.

Using libvirt TLS

Libvirt TLS can be enabled in Kolla Ansible by setting the following option in `/etc/kolla/globals.yml`:

```
libvirt_tls: "yes"
```

Creation of the TLS certificates is currently out-of-scope for Kolla Ansible. You will need to either use an existing Internal CA or you will need to generate your own offline CA. For the TLS communication to work correctly you will have to supply Kolla Ansible the following pieces of information:

- cacert.pem
  - This is the CAs public certificate that all of the client and server certificates are signed with. Libvirt and nova-compute will need this so they can verify that all the certificates being used were signed by the CA and should be trusted.
- serverkey.pem
  - This is the private key for the server, and is no different than the private key of a TLS certificate. It should be carefully protected, just like the private key of a TLS certificate.
- servercert.pem
  - This is the public certificate for the server. Libvirt will present this certificate to any connection made to the TLS port. This is no different than the public certificate part of a standard TLS certificate/key bundle.
- clientkey.pem
  - This is the client private key, which nova-compute/libvirt will use when it is connecting to libvirt. Think of this as an SSH private key and protect it in a similar manner.
- clientcert.pem
  - This is the client certificate that nova-compute/libvirt will present when it is connecting to libvirt. Think of this as the public side of an SSH key.

Kolla Ansible will search for these files for each compute node in the following locations and order on the host where Kolla Ansible is executed:

- `/etc/kolla/config/nova/nova-libvirt/<hostname>/`
- `/etc/kolla/config/nova/nova-libvirt/`

In most cases you will want to have a unique set of server and client certificates and keys per hypervisor and with a common CA certificate. In this case you would place each of the server/client certificate and key PEM files under `/etc/kolla/config/nova/nova-libvirt/<hostname>/` and the CA certificate under `/etc/kolla/config/nova/nova-libvirt/`.

However, it is possible to make use of wildcard server certificate and a single client certificate that is shared by all servers. This will allow you to generate a single client certificate and a single server certificate that is shared across every hypervisor. In this case you would store everything under `/etc/kolla/config/nova/nova-libvirt/`.

Externally managed certificates

One more option for deployers who already have automation to get TLS certs onto servers is to disable certificate management under `/etc/kolla/globals.yaml`:

```
libvirt_tls_manage_certs: "no"
```

With this option disabled Kolla Ansible will simply assume that certificates and keys are already installed in their correct locations. Deployers will be responsible for making sure that the TLS certificates/keys get placed in to the correct container configuration directories on the servers so that they can get copied into the nova-compute and nova-libvirt containers. With this option disabled you will also be responsible for restarting the nova-compute and nova-libvirt containers when the certs are updated, as kolla-ansible will not be able to tell when the files have changed.

Masakari - Virtual Machines High Availability

Overview

Masakari provides Instances High Availability Service for OpenStack clouds by automatically recovering failed Instances. Currently, Masakari can recover KVM-based Virtual Machine(VM)s from failure events such as VM process down, provisioning process down, and nova-compute host failure. Masakari also provides an API service to manage and control the automated rescue mechanism.

Kolla deploys Masakari API, Masakari Engine and Masakari Monitor containers which are the main Masakari components only if `enable_masakari` is set in `/etc/kolla/globals.yml`. By default, both the Masakari Host Monitor and Masakari Instance Monitor containers are enabled. The deployment of each type of monitors can be controlled individually via `enable_masakari_instancemonitor` and `enable_masakari_hostmonitor`.

Nova Cells

Overview

Nova cells V2 is a feature that allows Nova deployments to be scaled out to a larger size than would otherwise be possible. This is achieved through sharding of the compute nodes into pools known as *cells*, with each cell having a separate message queue and database.

Further information on cells can be found in the Nova documentation here and here. This document assumes the reader is familiar with the concepts of cells.

Cells: deployment perspective

From a deployment perspective, nova cell support involves separating the Nova services into two sets - global services and per-cell services.

Global services:

- `nova-api`
- `nova-scheduler`
- `nova-super-conductor` (in multi-cell mode)

Per-cell control services:

- `nova-compute-ironic` (for Ironic cells)
- `nova-conductor`
- `nova-novncproxy`
- `nova-serialproxy`
- `nova-spicehtml5proxy`

Per-cell compute services:

- `nova-compute`
- `nova-libvirt`
- `nova-ssh`

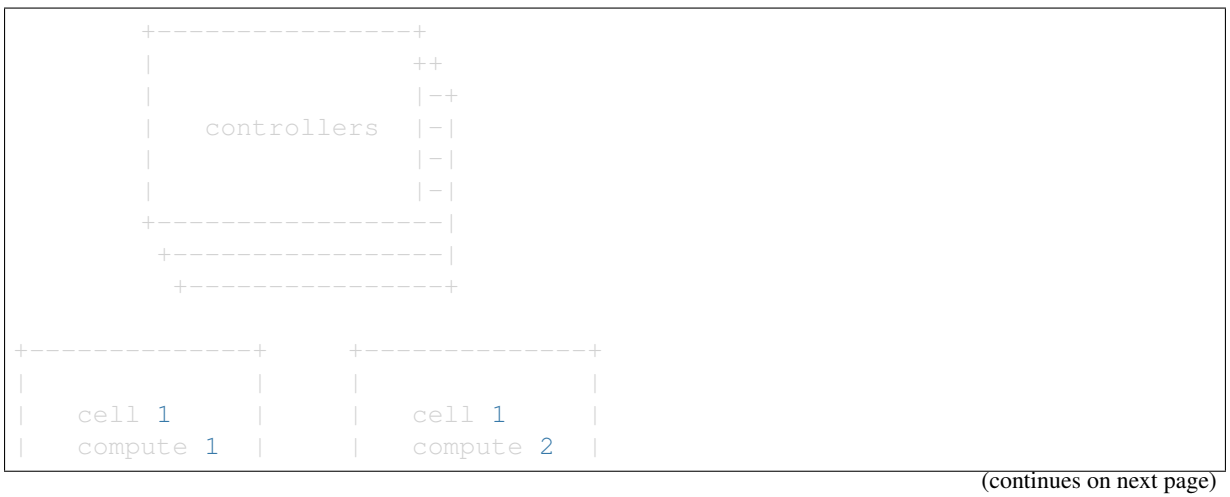
Another consideration is the database and message queue clusters that the cells depend on. This will be discussed later.

Service placement

There are a number of ways to place services in a multi-cell environment.

Single cell topology

The single cell topology is used by default, and is limited to a single cell:



(continues on next page)

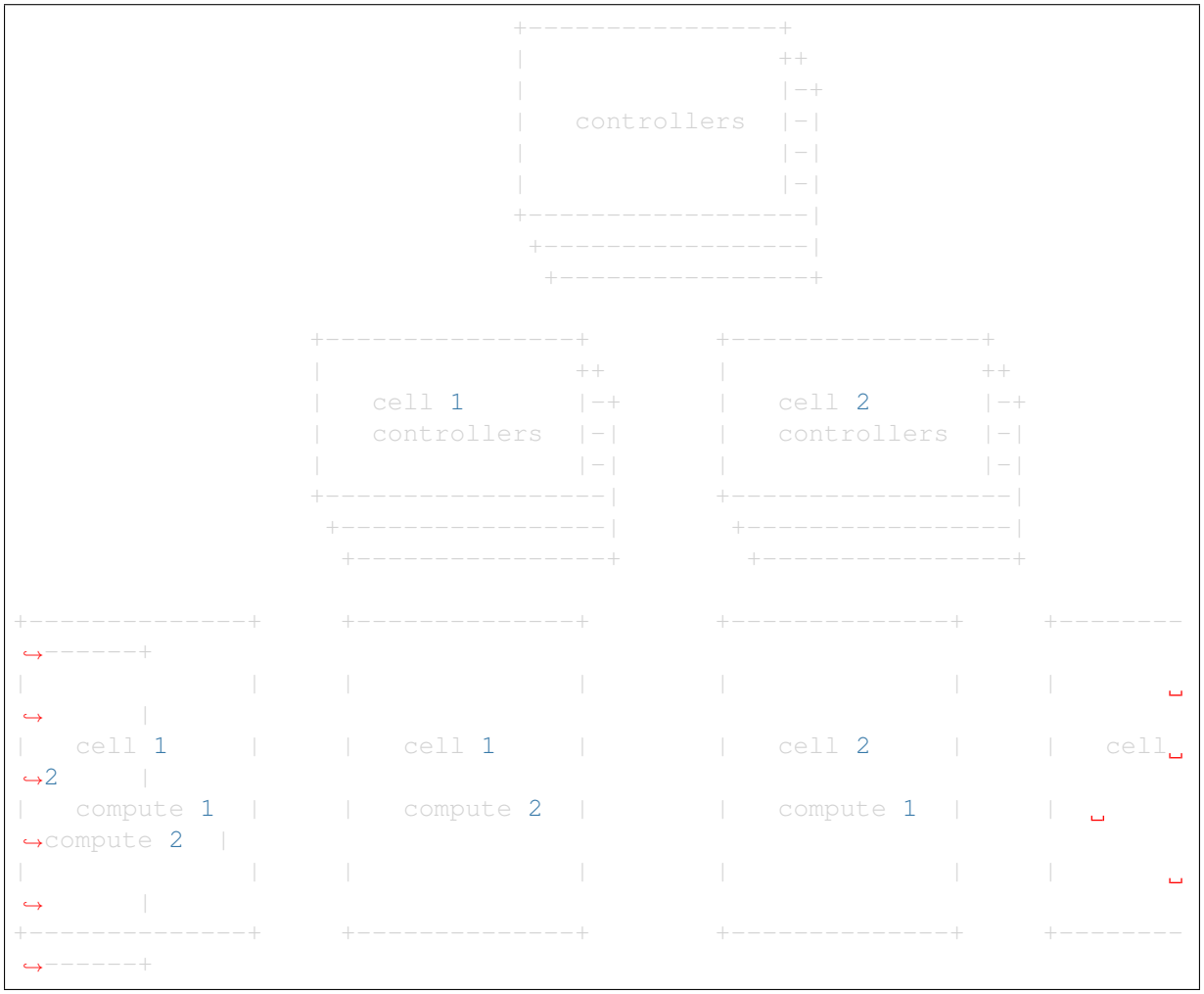
(continued from previous page)

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

All control services run on the controllers, and there is no superconductor.

Dedicated cell controller topology

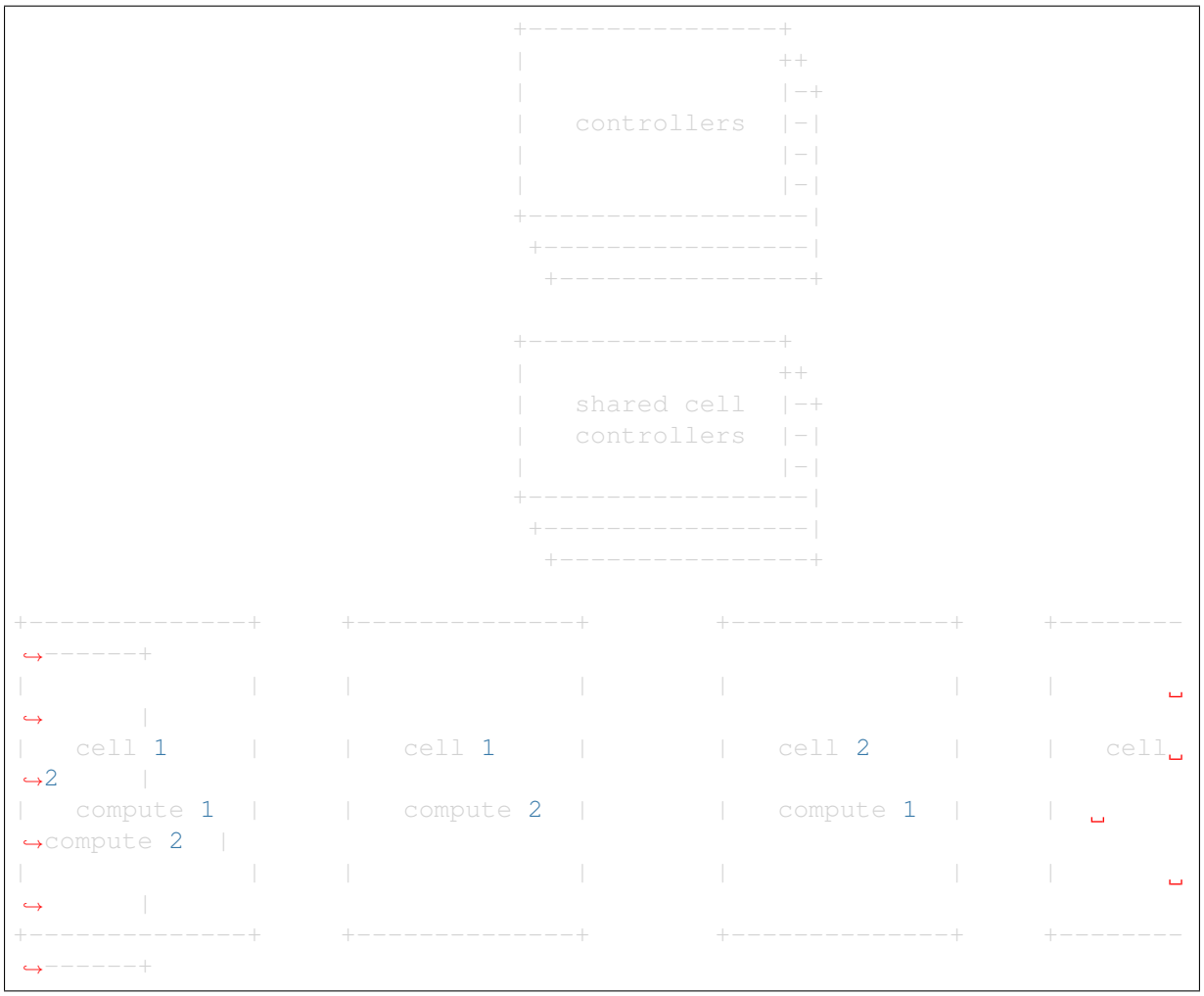
In this topology, each cell has a dedicated group of controllers to run cell control services. The following diagram shows the topology for a cloud with two cells:



Shared cell controller topology

Note: This topology is not yet supported by Kolla Ansible.

An alternative configuration is to place the cell control services for multiple cells on a single shared group of cell controllers. This might allow for more efficient use of hardware where the control services for a single cell do not fully consume the resources of a set of cell controllers:



Databases & message queues

The global services require access to a database for the API and cell0 databases, in addition to a message queue. Each cell requires its own database and message queue instance. These could be separate database and message queue clusters, or shared database and message queue clusters partitioned via database names and virtual hosts. Currently Kolla Ansible supports deployment of shared database cluster and message queue clusters.

Configuration

See also:

Configuring Kolla Ansible for deployment of multiple cells typically requires use of *inventory host and group variables*.

Enabling multi-cell support

Support for deployment of multiple cells is disabled by default - nova is deployed in single conductor mode.

Deployment of multiple cells may be enabled by setting `enable_cells` to `yes` in `globals.yml`. This deploys nova in superconductor mode, with separate conductors for each cell.

Naming cells

By default, all cell services are deployed in a single unnamed cell. This behaviour is backwards compatible with previous releases of Kolla Ansible.

To deploy hosts in a different cell, set the `nova_cell_name` variable for the hosts in the cell. This can be done either using host variables or group variables.

Groups

In a single cell deployment, the following Ansible groups are used to determine the placement of services:

- `compute`: `nova-compute`, `nova-libvirt`, `nova-ssh`
- `nova-compute-ironic`: `nova-compute-ironic`
- `nova-conductor`: `nova-conductor`
- `nova-novncproxy`: `nova-novncproxy`
- `nova-serialproxy`: `nova-serialproxy`
- `nova-spicehtml5proxy`: `nova-spicehtml5proxy`

In a multi-cell deployment, this is still necessary - compute hosts must be in the `compute` group. However, to provide further control over where cell services are placed, the following variables are used:

- `nova_cell_compute_group`
- `nova_cell_compute_ironic_group`
- `nova_cell_conductor_group`
- `nova_cell_novncproxy_group`
- `nova_cell_serialproxy_group`
- `nova_cell_spicehtml5proxy_group`

For backwards compatibility, these are set by default to the original group names. For a multi-cell deployment, they should be set to the name of a group containing only the compute hosts in that cell.

Example

In the following example we have two cells, `cell1` and `cell2`. Each cell has two compute nodes and a cell controller.

Inventory:

```
[compute:children]
compute-cell1
compute-cell2

[nova-conductor:children]
cell-control-cell1
cell-control-cell2

[nova-novncproxy:children]
cell-control-cell1
cell-control-cell2

[nova-spicehtml5proxy:children]
cell-control-cell1
cell-control-cell2

[nova-serialproxy:children]
cell-control-cell1
cell-control-cell2

[cell1:children]
compute-cell1
cell-control-cell1

[cell2:children]
compute-cell2
cell-control-cell2

[compute-cell1]
compute01
compute02

[compute-cell2]
compute03
compute04

[cell-control-cell1]
cell-control01

[cell-control-cell2]
cell-control02
```

Cell1 group variables (`group_vars/cell1`):

```
nova_cell_name: cell1
nova_cell_compute_group: compute-cell1
```

(continues on next page)

(continued from previous page)

```
nova_cell_conductor_group: cell-control-cell1
nova_cell_novncproxy_group: cell-control-cell1
nova_cell_serialproxy_group: cell-control-cell1
nova_cell_spicehtml5proxy_group: cell-control-cell1
```

Cell2 group variables (`group_vars/cell2`):

```
nova_cell_name: cell2
nova_cell_compute_group: compute-cell2
nova_cell_conductor_group: cell-control-cell2
nova_cell_novncproxy_group: cell-control-cell2
nova_cell_serialproxy_group: cell-control-cell2
nova_cell_spicehtml5proxy_group: cell-control-cell2
```

Note that these example cell group variables specify groups for all console proxy services for completeness. You will need to ensure that there are no port collisions. For example, if in both cell1 and cell2, you use the default `novncproxy` console proxy, you could add `nova_novncproxy_port: 6082` to the cell2 group variables to prevent a collision with cell1.

Databases

The database connection for each cell is configured via the following variables:

- `nova_cell_database_name`
- `nova_cell_database_user`
- `nova_cell_database_password`
- `nova_cell_database_address`
- `nova_cell_database_port`

By default the MariaDB cluster deployed by Kolla Ansible is used. For an unnamed cell, the `nova` database is used for backwards compatibility. For a named cell, the database is named `nova_<cell name>`.

Message queues

The RPC message queue for each cell is configured via the following variables:

- `nova_cell_rpc_user`
- `nova_cell_rpc_password`
- `nova_cell_rpc_port`
- `nova_cell_rpc_group_name`
- `nova_cell_rpc_transport`
- `nova_cell_rpc_vhost`

And for notifications:

- `nova_cell_notify_user`

- nova_cell_notify_password
- nova_cell_notify_port
- nova_cell_notify_group_name
- nova_cell_notify_transport
- nova_cell_notify_vhost

By default the message queue cluster deployed by Kolla Ansible is used. For an unnamed cell, the / virtual host used by all OpenStack services is used for backwards compatibility. For a named cell, a virtual host named nova_<cell name> is used.

Conductor & API database

By default the cell conductors are configured with access to the API database. This is currently necessary for some operations in Nova which require an *upcall*.

If those operations are not required, it is possible to prevent cell conductors from accessing the API database by setting nova_cell_conductor_has_api_database to no.

Console proxies

General information on configuring console access in Nova is available *here*. For deployments with multiple cells, the console proxies for each cell must be accessible by a unique endpoint. We achieve this by adding an HAProxy frontend for each cell that forwards to the console proxies for that cell. Each frontend must use a different port. The port may be configured via the following variables:

- nova_novncproxy_port
- nova_spicehtml5proxy_port
- nova_serialproxy_port

Ironic

Currently all Ironic-based instances are deployed in a single cell. The name of that cell is configured via nova_cell_ironic_cell_name, and defaults to the unnamed cell. nova_cell_compute_ironic_group can be used to set the group that the nova-compute-ironic services are deployed to.

Deployment

Deployment in a multi-cell environment does not need to be done differently than in a single-cell environment - use the kolla-ansible deploy command.

Scaling out

A common operational task in large scale environments is to add new compute resources to an existing deployment. In a multi-cell environment it is likely that these will all be added to one or more new or existing cells. Ideally we would not risk affecting other cells, or even the control hosts, when deploying these new resources.

The Nova cells support in Kolla Ansible has been built such that it is possible to add new cells or extend existing ones without affecting the rest of the cloud. This is achieved via the `--limit` argument to `kolla-ansible`. For example, if we are adding a new cell `cell03` to an existing cloud, and all hosts for that cell (control and compute) are in a `cell03` group, we could use this as our limit:

```
kolla-ansible deploy --limit cell03
```

When adding a new cell, we also need to ensure that HAProxy is configured for the console proxies in that cell:

```
kolla-ansible deploy --tags haproxy
```

Another benefit of this approach is that it should be faster to complete, as the number of hosts Ansible manages is reduced.

Upgrades

Similar to deploys, upgrades in a multi-cell environment can be performed in the same way as single-cell environments, via `kolla-ansible upgrade`.

Staged upgrades

Note: Staged upgrades are not applicable when `nova_safety_upgrade` is `yes`.

In large environments the risk involved with upgrading an entire site can be significant, and the ability to upgrade one cell at a time is crucial. This is very much an advanced procedure, and operators attempting this should be familiar with the Nova upgrade documentation.

Here we use Ansible tags and limits to control the upgrade process. We will only consider the Nova upgrade here. It is assumed that all dependent services have been upgraded (see `ansible/site.yml` for correct ordering).

The first step, which may be performed in advance of the upgrade, is to perform the database schema migrations.

```
kolla-ansible upgrade --tags nova-bootstrap
```

Next, we upgrade the global services.

```
kolla-ansible upgrade --tags nova-api-upgrade
```

Now the cell services can be upgraded. This can be performed in batches of one or more cells at a time, using `--limit`. For example, to upgrade services in `cell03`:

```
kolla-ansible upgrade --tags nova-cell-upgrade --limit cell03
```

At this stage, we might wish to perform testing of the new services, to check that they are functioning correctly before proceeding to other cells.

Once all cells have been upgraded, we can reload the services to remove RPC version pinning, and perform online data migrations.

```
kolla-ansible upgrade --tags nova-reload,nova-online-data-migrations
```

The nova upgrade is now complete, and upgrading of other services may continue.

Nova Fake Driver

One common question from OpenStack operators is that how does the control plane (for example, database, messaging queue, nova-scheduler) scales?. To answer this question, operators setup Rally to drive workload to the OpenStack cloud. However, without a large number of nova-compute nodes, it becomes difficult to exercise the control performance.

Given the built-in feature of Docker container, Kolla enables standing up many of Compute nodes with nova fake driver on a single host. For example, we can create 100 nova-compute containers on a real host to simulate the 100-hypervisor workload to the `nova-conductor` and the messaging queue.

Use nova-fake driver

Nova fake driver can not work with all-in-one deployment. This is because the fake `neutron-openvswitch-agent` for the fake `nova-compute` container conflicts with `neutron-openvswitch-agent` on the Compute nodes. Therefore, in the inventory the network node must be different than the Compute node.

By default, Kolla uses libvirt driver on the Compute node. To use nova-fake driver, edit the following parameters in `/etc/kolla/globals.yml` or in the command line options.

```
enable_nova_fake: "yes"
num_nova_fake_per_node: 5
```

Each Compute node will run 5 `nova-compute` containers and 5 `neutron-plugin-agent` containers. When booting instance, there will be no real instances created. But **`nova list`** shows the fake instances.

Nova - Compute Service

Nova is a core service in OpenStack, and provides compute services. Typically this is via Virtual Machines (VMs), but may also be via bare metal servers if Nova is coupled with Ironic.

Nova is enabled by default, but may be disabled by setting `enable_nova` to `no` in `globals.yml`.

Virtualisation Drivers

The virtualisation driver may be selected via `nova_compute_virt_type` in `globals.yml`. Supported options are `qemu`, `kvm`, and `vmware`. The default is `kvm`.

Libvirt

Information on the libvirt-based drivers `kvm` and `qemu` can be found in *Libvirt - Nova Virtualisation Driver*.

VMware

Information on the VMware-based driver `vmware` can be found in *VMware - Nova Virtualisation Driver*.

Bare Metal

Information on using Nova with Ironic to deploy compute instances to bare metal can be found in *Ironic - Bare Metal provisioning*.

Fake Driver

The fake driver can be used for testing Novas scaling properties without requiring access to a large amount of hardware resources. It is covered in *Nova Fake Driver*.

Consoles

The console driver may be selected via `nova_console` in `globals.yml`. Valid options are `none`, `novnc` and `spice`. Additionally, serial console support can be enabled by setting `enable_nova_serialconsole_proxy` to `yes`.

Cells

Information on using Nova Cells V2 to scale out can be found in *Nova Cells*.

Failure handling

Compute service registration

During deployment, Kolla Ansible waits for Nova compute services to register themselves. By default, if a compute service does not register itself before the timeout, that host will be marked as failed in the Ansible run. This behaviour is useful at scale, where failures are more frequent.

Alternatively, to fail all hosts in a cell when any compute service fails to register, set `nova_compute_registration_fatal` to `true`.

VMware - Nova Virtualisation Driver

Overview

Kolla can deploy the Nova and Neutron Service(s) for VMware vSphere. Depending on the network architecture (NsxV or DVS) you choose, Kolla deploys the following OpenStack services for VMware vSphere:

For VMware NsxV:

- nova-compute
- neutron-server

For VMware DVS:

- nova-compute
- neutron-server
- neutron-dhcp-agent
- neutron-metadata-agent

Kolla can deploy the Glance and Cinder services using VMware datastore as their backend. Ceilometer metering for vSphere is also supported.

Because the vmware-nsx drivers for neutron use completely different architecture than other types of virtualization, vmware-nsx drivers cannot coexist with other type of virtualization in one region. In neutron vmware-nsx drivers, neutron-server acts like an agent to translate OpenStack actions into what vSphere/NSX Manager API can understand. Neutron does not directly takes control of the Open vSwitch inside the VMware environment but through the API exposed by vSphere/NSX Manager.

For VMware DVS, the Neutron DHCP agent does not attaches to Open vSwitch inside VMware environment, but attach to the Open vSwitch bridge called `br-dvs` on the OpenStack side and replies to/receives DHCP packets through VLAN. Similar to what the DHCP agent does, Neutron metadata agent attaches to `br-dvs` bridge and works through VLAN.

Note: VMware NSX-DVS plugin does not support tenant networks, so all VMs should attach to Provider VLAN/Flat networks.

VMware NSX-V

Preparation

You should have a working NSX-V environment, this part is out of scope of Kolla. For more information, please see VMware NSX-V documentation.

Note: In addition, it is important to modify the firewall rule of vSphere to make sure that VNC is accessible from outside VMware environment.

On every VMware host, edit `/etc/vmware/firewall/vnc.xml` as below:

```
<!-- FirewallRule for VNC Console -->
<ConfigRoot>
<service>
<id>VNC</id>
<rule id = '0000'>
<direction>inbound</direction>
<protocol>tcp</protocol>
<porttype>dst</porttype>
<port>
<begin>5900</begin>
<end>5999</end>
</port>
</rule>
<rule id = '0001'>
<direction>outbound</direction>
<protocol>tcp</protocol>
<porttype>dst</porttype>
<port>
<begin>0</begin>
<end>65535</end>
</port>
</rule>
<enabled>true</enabled>
<required>false</required>
</service>
</ConfigRoot>
```

Then refresh the firewall config by:

```
# esxcli network firewall refresh
```

Verify that the firewall config is applied:

```
# esxcli network firewall ruleset list
```

Deployment

Enable VMware nova-compute plugin and NSX-V neutron-server plugin in `/etc/kolla/globals.yml`:

```
nova_compute_virt_type: "vmware"
neutron_plugin_agent: "vmware_nsxv"
```

Note: VMware NSX-V also supports Neutron FWaaS and VPNaaS services, you can enable them by setting these options in `globals.yml`:

- enable_neutron_vpnaas: yes
- enable_neutron_fwaas: yes

If you want to set VMware datastore as cinder backend, enable it in `/etc/kolla/globals.yml`:

```
enable_cinder: "yes"
cinder_backend_vmwarevc_vmdk: "yes"
vmware_datastore_name: "TestDatastore"
```

If you want to set VMware datastore as glance backend, enable it in `/etc/kolla/globals.yml`:

```
glance_backend_vmware: "yes"
vmware_vcenter_name: "TestDatacenter"
vmware_datastore_name: "TestDatastore"
```

VMware options are required in `/etc/kolla/globals.yml`, these options should be configured correctly according to your NSX-V environment.

Options for `nova-compute` and `ceilometer`:

```
vmware_vcenter_host_ip: "127.0.0.1"
vmware_vcenter_host_username: "admin"
vmware_vcenter_cluster_name: "cluster-1"
vmware_vcenter_insecure: "True"
vmware_vcenter_datastore_regex: ".*"
```

Note: The VMware vCenter password has to be set in `/etc/kolla/passwords.yml`.

```
vmware_vcenter_host_password: "admin"
```

Options for Neutron NSX-V support:

```
vmware_nsxv_user: "nsx_manager_user"
vmware_nsxv_manager_uri: "https://127.0.0.1"
vmware_nsxv_cluster_moid: "TestCluster"
vmware_nsxv_datacenter_moid: "TestDataCeter"
vmware_nsxv_resource_pool_id: "TestRSGroup"
vmware_nsxv_datastore_id: "TestDataStore"
vmware_nsxv_external_network: "TestDVSPort-Ext"
vmware_nsxv_vdn_scope_id: "TestVDNScope"
vmware_nsxv_dvs_id: "TestDVS"
vmware_nsxv_backup_edge_pool: "service:compact:1:2"
vmware_nsxv_spoofguard_enabled: "false"
vmware_nsxv_metadata_initializer: "false"
vmware_nsxv_edge_ha: "false"
```

Note: If you want to set secure connections to VMware, set `vmware_vcenter_insecure` to false. Secure connections to vCenter requires a CA file, copy the vCenter CA file to `/etc/kolla/config/vmware_ca`.

Note: The VMware NSX-V password has to be set in `/etc/kolla/passwords.yml`.

```
vmware_nsxv_password: "nsx_manager_password"
```

Then you should start **`kolla-ansible`** deployment normally as KVM/QEMU deployment.

VMware NSX-DVS

Preparation

Before deployment, you should have a working VMware vSphere environment. Create a cluster and a vSphere Distributed Switch with all the host in the cluster attached to it.

For more information, please see Setting Up Networking with vSphere Distributed Switches.

Deployment

Enable VMware nova-compute plugin and NSX-V neutron-server plugin in `/etc/kolla/globals.yml`:

```
nova_compute_virt_type: "vmware"
neutron_plugin_agent: "vmware_dvs"
```

If you want to set VMware datastore as Cinder backend, enable it in `/etc/kolla/globals.yml`:

```
enable_cinder: "yes"
cinder_backend_vmwarevc_vmdk: "yes"
vmware_datastore_name: "TestDatastore"
```

If you want to set VMware datastore as Glance backend, enable it in `/etc/kolla/globals.yml`:

```
glance_backend_vmware: "yes"
vmware_vcenter_name: "TestDatacenter"
vmware_datastore_name: "TestDatastore"
```

VMware options are required in `/etc/kolla/globals.yml`, these options should be configured correctly according to the vSphere environment you installed before. All option for nova, cinder, glance are the same as VMware-NSX, except the following options.

Options for Neutron NSX-DVS support:

```
vmware_dvs_host_ip: "192.168.1.1"
vmware_dvs_host_port: "443"
vmware_dvs_host_username: "admin"
vmware_dvs_dvs_name: "VDS-1"
vmware_dvs_dhcp_override_mac: ""
```

Note: The VMware NSX-DVS password has to be set in `/etc/kolla/passwords.yml`.

```
vmware_dvs_host_password: "password"
```

Then you should start **`kolla-ansible`** deployment normally as KVM/QEMU deployment.

For more information on OpenStack vSphere, see VMware vSphere, VMware-NSX package.

Zun - Container service

Zun is an OpenStack Container service. It aims to provide an OpenStack API for provisioning and managing containerized workload on OpenStack. For more details about Zun, see OpenStack Zun Documentation.

Preparation and Deployment

By default Zun and its dependencies are disabled. In order to enable Zun, you need to edit globals.yml and set the following variables:

```
enable_zun: "yes"
enable_kuryr: "yes"
enable_etcd: "yes"
docker_configure_for_zun: "yes"
containerd_configure_for_zun: "yes"
```

Currently Kuryr does not support Docker 23 and later due to dropped cluster-store option (bug bug).You need to cap docker by setting the following variables in globals.yml.

```
docker_apt_package_pin: "5:20.*"
docker_yum_package_pin: "20.*"
```

Docker reconfiguration requires rebootstrapping before deploy.

Make sure you understand the consequences of restarting Docker. Please see *Subsequent bootstrap considerations* for details. If its initial deploy, then there is nothing to worry about because its initial bootstrapping as well and there are no running services to affect.

```
$ kolla-ansible bootstrap-servers
```

Finally deploy:

```
$ kolla-ansible deploy
```

Verification

1. Generate the credentials file:

   ```
   $ kolla-ansible post-deploy
   ```

2. Source credentials file:

   ```
   $ . /etc/kolla/admin-openrc.sh
   ```

3. Download and create a glance container image:

   ```
   $ docker pull cirros
   $ docker save cirros | openstack image create cirros --public \
     --container-format docker --disk-format raw
   ```

4. Create zun container:

```
$ zun create --name test --net network=demo-net cirros ping -c4 8.8.8.
↪8
```

Note: Kuryr does not support networks with DHCP enabled, disable DHCP in the subnet used for zun containers.

```
$ openstack subnet set --no-dhcp <subnet>
```

5. Verify container is created:

```
$ zun list

+--------------------------------------+------+---------------+-------
↪--+------------+------------+-------+
| uuid                                 | name | image         |␣
↪status  | task_state | addresses  | ports |
+--------------------------------------+------+---------------+-------
↪--+------------+------------+-------+
| 3719a73e-5f86-47e1-bc5f-f4074fc749f2 | test | cirros        |␣
↪Created | None       | 172.17.0.3 | []    |
+--------------------------------------+------+---------------+-------
↪--+------------+------------+-------+
```

6. Start container:

```
$ zun start test
Request to start container test has been accepted.
```

7. Verify container:

```
$ zun logs test
PING 8.8.8.8 (8.8.8.8): 56 data bytes
64 bytes from 8.8.8.8: seq=0 ttl=45 time=96.396 ms
64 bytes from 8.8.8.8: seq=1 ttl=45 time=96.504 ms
64 bytes from 8.8.8.8: seq=2 ttl=45 time=96.721 ms
64 bytes from 8.8.8.8: seq=3 ttl=45 time=95.884 ms

--- 8.8.8.8 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 95.884/96.376/96.721 ms
```

For more information about how zun works, see zun, OpenStack Container service.

6.1.2 Bare Metal

This section describes configuring bare metal provisioning such as Ironic.

Ironic - Bare Metal provisioning

Overview

Ironic is the OpenStack service for handling bare metal, i.e., the physical machines. It can work standalone as well as with other OpenStack services (notably, Neutron and Nova).

Pre-deployment Configuration

Enable Ironic in `/etc/kolla/globals.yml`:

```
enable_ironic: "yes"
```

In the same file, define a network interface as the default NIC for dnsmasq and a range of IP addresses that will be available for use by Ironic inspector. The optional netmask of the network should be provided in case when DHCP-relay is used. Finally, define a network to be used for the Ironic cleaning network:

```
ironic_dnsmasq_interface: "eth1"
ironic_dnsmasq_dhcp_range: "192.168.5.100,192.168.5.110,255.255.255.0"
ironic_cleaning_network: "public1"
```

In the same file, optionally a default gateway to be used for the Ironic Inspector inspection network:

```
ironic_dnsmasq_default_gateway: 192.168.5.1
```

In the same file, specify the PXE bootloader file for Ironic Inspector. The file is relative to the `/tftpboot` directory. The default is `pxelinux.0`, and should be correct for x86 systems. Other platforms may require a different value, for example aarch64 on Debian requires `debian-installer/arm64/bootnetaa64.efi`.

```
ironic_dnsmasq_boot_file: pxelinux.0
```

Ironic inspector also requires a deploy kernel and ramdisk to be placed in `/etc/kolla/config/ironic/`. The following example uses coreos which is commonly used in Ironic deployments, though any compatible kernel/ramdisk may be used:

```
$ curl https://tarballs.opendev.org/openstack/ironic-python-agent/dib/
↪files/ipa-centos8-stable-wallaby.kernel \
  -o /etc/kolla/config/ironic/ironic-agent.kernel

$ curl https://tarballs.opendev.org/openstack/ironic-python-agent/dib/
↪files/ipa-centos8-stable-wallaby.initramfs \
  -o /etc/kolla/config/ironic/ironic-agent.initramfs
```

You may optionally pass extra kernel parameters to the inspection kernel using:

```
ironic_inspector_kernel_cmdline_extras: ['ipa-lldp-timeout=90.0', 'ipa-
↪collect-lldp=1']
```

in /etc/kolla/globals.yml.

Enable iPXE booting (optional)

You can optionally enable booting via iPXE by setting enable_ironic_ipxe to true in /etc/kolla/globals.yml:

```
enable_ironic_ipxe: "yes"
```

When iPXE booting is enabled, the ironic_ipxe container is used to serve the iPXE boot images as described below. Regardless of the setting above, the same container is used to support the direct deploy interface.

The port used for the iPXE webserver is controlled via ironic_ipxe_port in /etc/kolla/globals.yml:

```
ironic_ipxe_port: "8089"
```

The following changes will occur if iPXE booting is enabled:

- Ironic will be configured with the ipxe_enabled configuration option set to true
- The inspection ramdisk and kernel will be loaded via iPXE
- The DHCP servers will be configured to chainload iPXE from an existing PXE environment. You may also boot directly to iPXE by some other means e.g by burning it to the option rom of your ethernet card.

Note that due to a limitation in Kolla Ansible, PXE and iPXE cannot be used together in a single deployment.

In order to enable the iPXE driver in Ironic, set the [DEFAULT] enabled_boot_interfaces option in /etc/kolla/config/ironic.conf:

```
[DEFAULT]
enabled_boot_interfaces = ipxe
```

Attach ironic to external keystone (optional)

In multi-regional deployment keystone could be installed in one region (lets say region 1) and ironic - in another region (lets say region 2). In this case we dont install keystone together with ironic in region 2, but have to configure ironic to connect to existing keystone in region 1. To deploy ironic in this way we have to set variable enable_keystone to "no".

```
enable_keystone: "no"
```

It will prevent keystone from being installed in region 2.

To add keystone-related sections in ironic.conf, it is also needed to set variable ironic_enable_keystone_integration to "yes"

```
ironic_enable_keystone_integration: "yes"
```

Deployment

Run the deploy as usual:

```
$ kolla-ansible deploy
```

Post-deployment configuration

The Ironic documentation describes how to create the deploy kernel and ramdisk and register them with Glance. In this example were reusing the same images that were fetched for the Inspector:

```
openstack image create --disk-format aki --container-format aki --public \
  --file /etc/kolla/config/ironic/ironic-agent.kernel deploy-vmlinuz

openstack image create --disk-format ari --container-format ari --public \
  --file /etc/kolla/config/ironic/ironic-agent.initramfs deploy-initrd
```

The Ironic documentation describes how to create Nova flavors for bare metal. For example:

```
openstack flavor create my-baremetal-flavor \
  --ram 512 --disk 1 --vcpus 1 \
  --property resources:CUSTOM_BAREMETAL_RESOURCE_CLASS=1 \
  --property resources:VCPU=0 \
  --property resources:MEMORY_MB=0 \
  --property resources:DISK_GB=0
```

The Ironic documentation describes how to enroll baremetal nodes and ports. In the following example ensure to substitute correct values for the kernel, ramdisk, and MAC address for your baremetal node.

```
openstack baremetal node create --driver ipmi --name baremetal-node \
  --driver-info ipmi_port=6230 --driver-info ipmi_username=admin \
  --driver-info ipmi_password=password \
  --driver-info ipmi_address=192.168.5.1 \
  --resource-class baremetal-resource-class --property cpus=1 \
  --property memory_mb=512 --property local_gb=1 \
  --property cpu_arch=x86_64 \
  --driver-info deploy_kernel=15f3c95f-d778-43ad-8e3e-9357be09ca3d \
  --driver-info deploy_ramdisk=9b1e1ced-d84d-440a-b681-39c216f24121

openstack baremetal port create 52:54:00:ff:15:55 \
  --node 57aa574a-5fea-4468-afcf-e2551d464412 \
  --physical-network physnet1
```

Make the baremetal node available to nova:

```
openstack baremetal node manage 57aa574a-5fea-4468-afcf-e2551d464412
openstack baremetal node provide 57aa574a-5fea-4468-afcf-e2551d464412
```

It may take some time for the node to become available for scheduling in nova. Use the following commands to wait for the resources to become available:

```
openstack hypervisor stats show
openstack hypervisor show 57aa574a-5fea-4468-afcf-e2551d464412
```

Booting the baremetal

Assuming you have followed the examples above and created the demo resources as shown in the *Quick Start*, you can now use the following example command to boot the baremetal instance:

```
openstack server create --image cirros --flavor my-baremetal-flavor \
  --key-name mykey --network public1 demo1
```

In other cases you will need to adapt the command to match your environment.

Notes

Debugging DHCP

The following *tcpdump* command can be useful when debugging why dhcp requests may not be hitting various pieces of the process:

```
tcpdump -i <interface> port 67 or port 68 or port 69 -e -n
```

Configuring the Web Console

Configuration based off upstream Node web console.

Serial speed must be the same as the serial configuration in the BIOS settings. Default value: 115200bps, 8bit, non-parity.If you have different serial speed.

Set ironic_console_serial_speed in `/etc/kolla/globals.yml`:

```
ironic_console_serial_speed: 9600n8
```

Deploying using virtual baremetal (vbmc + libvirt)

See https://brk3.github.io/post/kolla-ironic-libvirt/

6.1.3 Storage

This section describes configuration of the different storage backends supported by kolla.

External Ceph

Kolla Ansible does not provide support for provisioning and configuring a Ceph cluster directly. Instead, administrators should use a tool dedicated to this purpose, such as:

- ceph-ansible
- cephadm

The desired pool(s) and keyrings should then be created via the Ceph CLI or similar.

Requirements

- An existing installation of Ceph
- Existing Ceph storage pools
- Existing credentials in Ceph for OpenStack services to connect to Ceph (Glance, Cinder, Nova, Gnocchi, Manila)

Refer to https://docs.ceph.com/en/latest/rbd/rbd-openstack/ for details on creating the pool and keyrings with appropriate permissions for each service.

Configuring External Ceph

Ceph integration is configured for different OpenStack services independently.

Glance

Ceph RBD can be used as a storage backend for Glance images. Configuring Glance for Ceph includes the following steps:

1. Enable Glance Ceph backend in `globals.yml`:

```
glance_backend_ceph: "yes"
```

2. Configure Ceph authentication details in `/etc/kolla/globals.yml`:
   - `ceph_glance_keyring` (default: `ceph.client.glance.keyring`)
   - `ceph_glance_user` (default: `glance`)
   - `ceph_glance_pool_name` (default: `images`)
3. Copy Ceph configuration file to `/etc/kolla/config/glance/ceph.conf`

```
[global]
fsid = 1d89fec3-325a-4963-a950-c4afedd37fe3
mon_initial_members = ceph-0
mon_host = 192.168.0.56
auth_cluster_required = cephx
auth_service_required = cephx
auth_client_required = cephx
```

4. Copy Ceph keyring to `/etc/kolla/config/glance/<ceph_glance_keyring>`

Cinder

Ceph RBD can be used as a storage backend for Cinder volumes. Configuring Cinder for Ceph includes following steps:

1. When using external Ceph, there may be no nodes defined in the storage group. This will cause Cinder and related services relying on this group to fail. In this case, operator should add some nodes to the storage group, all the nodes where `cinder-volume` and `cinder-backup` will run:

```
[storage]
control01
```

2. Enable Cinder Ceph backend in `globals.yml`:

```
cinder_backend_ceph: "yes"
```

3. Configure Ceph authentication details in `/etc/kolla/globals.yml`:
   - `ceph_cinder_keyring` (default: `ceph.client.cinder.keyring`)
   - `ceph_cinder_user` (default: `cinder`)
   - `ceph_cinder_pool_name` (default: `volumes`)
   - `ceph_cinder_backup_keyring` (default: `ceph.client.cinder-backup.keyring`)
   - `ceph_cinder_backup_user` (default: `cinder-backup`)
   - `ceph_cinder_backup_pool_name` (default: `backups`)
4. Copy Ceph configuration file to `/etc/kolla/config/cinder/ceph.conf`

   Separate configuration options can be configured for cinder-volume and cinder-backup by adding ceph.conf files to `/etc/kolla/config/cinder/cinder-volume` and `/etc/kolla/config/cinder/cinder-backup` respectively. They will be merged with `/etc/kolla/config/cinder/ceph.conf`.
5. Copy Ceph keyring files to:
   - `/etc/kolla/config/cinder/cinder-volume/<ceph_cinder_keyring>`
   - `/etc/kolla/config/cinder/cinder-backup/<ceph_cinder_keyring>`
   - `/etc/kolla/config/cinder/cinder-backup/<ceph_cinder_backup_keyring>`

Note: `cinder-backup` requires two keyrings for accessing volumes and backup pool.

Nova must also be configured to allow access to Cinder volumes:

1. Configure Ceph authentication details in `/etc/kolla/globals.yml`:
   - `ceph_cinder_keyring` (default: `ceph.client.cinder.keyring`)
2. Copy Ceph keyring file(s) to:
   - `/etc/kolla/config/nova/<ceph_cinder_keyring>`

Nova

Ceph RBD can be used as a storage backend for Nova instance ephemeral disks. This avoids the requirement for local storage for instances on compute nodes. It improves the performance of migration, since instances ephemeral disks do not need to be copied between hypervisors.

Configuring Nova for Ceph includes following steps:

1. Enable Nova Ceph backend in `globals.yml`:

```
nova_backend_ceph: "yes"
```

2. Configure Ceph authentication details in `/etc/kolla/globals.yml`:
   - `ceph_nova_keyring` (by default its the same as `ceph_cinder_keyring`)
   - `ceph_nova_user` (by default its the same as `ceph_cinder_user`)
   - `ceph_nova_pool_name` (default: `vms`)
3. Copy Ceph configuration file to `/etc/kolla/config/nova/ceph.conf`
4. Copy Ceph keyring file(s) to:
   - `/etc/kolla/config/nova/<ceph_nova_keyring>`

Note: If you are using a Ceph deployment tool that generates separate Ceph keys for Cinder and Nova, you will need to override `ceph_nova_keyring` and `ceph_nova_user` to match.

Gnocchi

Ceph object storage can be used as a storage backend for Gnocchi metrics. Configuring Gnocchi for Ceph includes following steps:

1. Enable Gnocchi Ceph backend in `globals.yml`:

```
gnocchi_backend_storage: "ceph"
```

2. Configure Ceph authentication details in `/etc/kolla/globals.yml`:
   - `ceph_gnocchi_keyring` (default: `ceph.client.gnocchi.keyring`)
   - `ceph_gnocchi_user` (default: `gnocchi`)
   - `ceph_gnocchi_pool_name` (default: `gnocchi`)
3. Copy Ceph configuration file to `/etc/kolla/config/gnocchi/ceph.conf`
4. Copy Ceph keyring to `/etc/kolla/config/gnocchi/<ceph_gnocchi_keyring>`

Manila

CephFS can be used as a storage backend for Manila shares. Configuring Manila for Ceph includes following steps:

1. Enable Manila Ceph backend in `globals.yml`:

```
enable_manila_backend_cephfs_native: "yes"
```

2. Configure Ceph authentication details in `/etc/kolla/globals.yml`:
   - `ceph_manila_keyring` (default: `ceph.client.manila.keyring`)
   - `ceph_manila_user` (default: `manila`)

Note: Required Ceph identity caps for manila user are documented in CephFS Native driver.

Important: CephFS driver in the Wallaby (or later) release requires a Ceph identity with a different set of Ceph capabilities when compared to the driver in a pre-Wallaby release - please refer to Manila CephFS Native driver Documentation.

---

3. Copy Ceph configuration file to `/etc/kolla/config/manila/ceph.conf`
4. Copy Ceph keyring to `/etc/kolla/config/manila/<ceph_manila_keyring>`
5. If using multiple filesystems (Ceph Pacific+), set `manila_cephfs_filesystem_name` in `/etc/kolla/globals.yml` to the name of the Ceph filesystem Manila should use. By default, Manila will use the first filesystem returned by the `ceph fs volume ls` command.
6. Setup Manila in the usual way

For more details on the rest of the Manila setup, such as creating the share type `default_share_type`, please see *Manila in Kolla*.

For more details on the CephFS Native driver, please see CephFS Native driver.

Cinder - Block storage

Overview

Cinder can be deployed using Kolla and supports the following storage backends:

- ceph
- hnas_nfs
- iscsi
- lvm
- nfs

LVM

When using the `lvm` backend, a volume group should be created on each storage node. This can either be a real physical volume or a loopback mounted file for development. Use `pvcreate` and `vgcreate` to create the volume group. For example with the devices `/dev/sdb` and `/dev/sdc`:

```
<WARNING ALL DATA ON /dev/sdb and /dev/sdc will be LOST!>

pvcreate /dev/sdb /dev/sdc
vgcreate cinder-volumes /dev/sdb /dev/sdc
```

During development, it may be desirable to use file backed block storage. It is possible to use a file and mount it as a block device via the loopback system.

```
free_device=$(losetup -f)
fallocate -l 20G /var/lib/cinder_data.img
losetup $free_device /var/lib/cinder_data.img
pvcreate $free_device
vgcreate cinder-volumes $free_device
```

Enable the `lvm` backend in `/etc/kolla/globals.yml`:

```
enable_cinder_backend_lvm: "yes"
```

Note: There are currently issues using the LVM backend in a multi-controller setup, see _bug 1571211 for more info.

NFS

To use the `nfs` backend, configure `/etc/exports` to contain the mount where the volumes are to be stored:

```
/kolla_nfs 192.168.5.0/24(rw,sync,no_root_squash)
```

In this example, `/kolla_nfs` is the directory on the storage node which will be `nfs` mounted, `192.168.5.0/24` is the storage network, and `rw,sync,no_root_squash` means make the share read-write, synchronous, and prevent remote root users from having access to all files.

Then start `nfsd`:

```
systemctl start nfs
```

On the deploy node, create `/etc/kolla/config/nfs_shares` with an entry for each storage node:

```
storage01:/kolla_nfs
storage02:/kolla_nfs
```

Finally, enable the `nfs` backend in `/etc/kolla/globals.yml`:

```
enable_cinder_backend_nfs: "yes"
```

Validation

Create a volume as follows:

```
openstack volume create --size 1 steak_volume
<bunch of stuff printed>
```

Verify it is available. If it says error, then something went wrong during LVM creation of the volume.

```
openstack volume list

+--------------------------------------+--------------+-----------+------+-
↪-------------+
| ID                                   | Display Name | Status    | Size |␣
↪Attached to |
+--------------------------------------+--------------+-----------+------+-
↪-------------+
| 0069c17e-8a60-445a-b7f0-383a8b89f87e | steak_volume | available |    1 |␣
↪             |
+--------------------------------------+--------------+-----------+------+-
↪-------------+
```

Attach the volume to a server using:

```
openstack server add volume steak_server 0069c17e-8a60-445a-b7f0-
↪383a8b89f87e
```

Check the console log to verify the disk addition:

```
openstack console log show steak_server
```

A `/dev/vdb` should appear in the console log, at least when booting cirros. If the disk stays in the available state, something went wrong during the iSCSI mounting of the volume to the guest VM.

Cinder LVM2 backend with iSCSI

As of Newton-1 milestone, Kolla supports LVM2 as cinder backend. It is accomplished by introducing two new containers `tgtd` and `iscsid`. `tgtd` container serves as a bridge between cinder-volume process and a server hosting Logical Volume Groups (LVG). `iscsid` container serves as a bridge between nova-compute process and the server hosting LVG.

In order to use Cinders LVM backend, a LVG named `cinder-volumes` should exist on the server and following parameter must be specified in `globals.yml`:

```
enable_cinder_backend_lvm: "yes"
```

For Ubuntu and LVM2/iSCSI

`iscsd` process uses configfs which is normally mounted at `/sys/kernel/config` to store discovered targets information, on centos/rhel type of systems this special file system gets mounted automatically, which is not the case on debian/ubuntu. Since `iscsid` container runs on every nova compute node, the following steps must be completed on every Ubuntu server targeted for nova compute role.

- Add configfs module to `/etc/modules`
- Rebuild initramfs using: `update-initramfs -u` command
- Stop `open-iscsi` system service due to its conflicts with iscsid container.

  Ubuntu 16.04 (systemd): `systemctl stop open-iscsi; systemctl stop iscsid`
- Make sure configfs gets mounted during a server boot up process. There are multiple ways to accomplish it, one example:

```
mount -t configfs /etc/rc.local /sys/kernel/config
```

> **Note:** There is currently an issue with the folder /sys/kernel/config as it is either empty or does not exist in several operating systems, see _bug 1631072 for more info

Cinder backend with external iSCSI storage

In order to use external storage system (like the ones from EMC or NetApp) the following parameter must be specified in `globals.yml`:

```
enable_cinder_backend_iscsi: "yes"
```

Also `enable_cinder_backend_lvm` should be set to `no` in this case.

Skip Cinder prechecks for Custom backends

In order to use custom storage backends which currently not yet implemented in Kolla, the following parameter must be specified in `globals.yml`:

```
skip_cinder_backend_check: True
```

All configuration for custom NFS backend should be performed via `cinder.conf` in config overrides directory.

Hitachi NAS Platform iSCSI and NFS drives for OpenStack

Overview

The Block Storage service provides persistent block storage resources that Compute instances can consume. This includes secondary attached storage similar to the Amazon Elastic Block Storage (EBS) offering. In addition, you can write images to a Block Storage device for Compute to use as a bootable persistent instance.

Requirements

- Hitachi NAS Platform Models 3080, 3090, 4040, 4060, 4080, and 4100.
- HNAS/SMU software version is 12.2 or higher.
- HNAS configuration and management utilities to create a storage pool (span) and an EVS.
  - GUI (SMU).
  - SSC CLI.
- You must set an iSCSI domain to EVS

Supported shared file systems and operations

The NFS and iSCSI drivers support these operations:

- Create, delete, attach, and detach volumes.
- Create, list, and delete volume snapshots.
- Create a volume from a snapshot.
- Copy an image to a volume.
- Copy a volume to an image.
- Clone a volume.
- Extend a volume.
- Get volume statistics.
- Manage and unmanage a volume.
- Manage and unmanage snapshots (HNAS NFS only).

Configuration example for Hitachi NAS Platform NFS

NFS backend

Enable cinder hnas backend nfs in `/etc/kolla/globals.yml`

```
enable_cinder_backend_hnas_nfs: "yes"
```

Create or modify the file `/etc/kolla/config/cinder.conf` and add the contents:

```
[DEFAULT]
enabled_backends = hnas-nfs

[hnas-nfs]
volume_driver = cinder.volume.drivers.hitachi.hnas_nfs.HNASNFSDriver
volume_nfs_backend = hnas_nfs_backend
hnas_nfs_username = supervisor
hnas_nfs_mgmt_ip0 = <hnas_ip>
hnas_chap_enabled = True

hnas_nfs_svc0_volume_type = nfs_gold
hnas_nfs_svc0_hdp = <svc0_ip>/<export_name>
```

Then set password for the backend in `/etc/kolla/passwords.yml`:

```
hnas_nfs_password: supervisor
```

Configuration on Kolla deployment

Enable Shared File Systems service and HNAS driver in `/etc/kolla/globals.yml`

```
enable_cinder: "yes"
```

Configuration on HNAS

Create the data HNAS network in Kolla OpenStack:

List the available tenants:

```
openstack project list
```

Create a network to the given tenant (service), providing the tenant ID, a name for the network, the name of the physical network over which the virtual network is implemented, and the type of the physical mechanism by which the virtual network is implemented:

```
neutron net-create --tenant-id <SERVICE_ID> hnas_network \
--provider:physical_network=physnet2 --provider:network_type=flat
```

Create a subnet to the same tenant (service), the gateway IP of this subnet, a name for the subnet, the network ID created before, and the CIDR of subnet:

```
neutron subnet-create --tenant-id <SERVICE_ID> --gateway <GATEWAY> \
--name hnas_subnet <NETWORK_ID> <SUBNET_CIDR>
```

Add the subnet interface to a router, providing the router ID and subnet ID created before:

```
neutron router-interface-add <ROUTER_ID> <SUBNET_ID>
```

Create volume

Create a non-bootable volume.

```
openstack volume create --size 1 my-volume
```

Verify Operation.

```
cinder show my-volume

+--------------------------------+--------------------------------------+
| Property                       | Value                                |
+--------------------------------+--------------------------------------+
| attachments                    | []                                   |
| availability_zone              | nova                                 |
| bootable                       | false                                |
| consistencygroup_id            | None                                 |
| created_at                     | 2017-01-17T19:02:45.000000           |
| description                    | None                                 |
| encrypted                      | False                                |
| id                             | 4f5b8ae8-9781-411e-8ced-de616ae64cfd |
| metadata                       | {}                                   |
| migration_status               | None                                 |
| multiattach                    | False                                |
| name                           | my-volume                            |
| os-vol-host-attr:host          | compute@hnas-iscsi#iscsi_gold        |
| os-vol-mig-status-attr:migstat | None                                 |
| os-vol-mig-status-attr:name_id | None                                 |
| os-vol-tenant-attr:tenant_id   | 16def9176bc64bd283d419ac2651e299     |
| replication_status             | disabled                             |
| size                           | 1                                    |
| snapshot_id                    | None                                 |
| source_volid                   | None                                 |
| status                         | available                            |
| updated_at                     | 2017-01-17T19:02:46.000000           |
| user_id                        | fb318b96929c41c6949360c4ccdbf8c0     |
| volume_type                    | None                                 |
+--------------------------------+--------------------------------------+

nova volume-attach INSTANCE_ID VOLUME_ID auto

+----------+--------------------------------------+
| Property | Value                                |
+----------+--------------------------------------+
| device   | /dev/vdc                             |
| id       | 4f5b8ae8-9781-411e-8ced-de616ae64cfd |
| serverId | 3bf5e176-be05-4634-8cbd-e5fe491f5f9c |
```

(continues on next page)

(continued from previous page)

```
| volumeId | 4f5b8ae8-9781-411e-8ced-de616ae64cfd |
+----------+--------------------------------------+

openstack volume list

+--------------------------------------+--------------+-----------------+--
↪----+----------------------------------------------+
| ID                                   | Display Name | Status          |␣
↪Size | Attached to                                  |
+--------------------------------------+--------------+-----------------+--
↪----+----------------------------------------------+
| 4f5b8ae8-9781-411e-8ced-de616ae64cfd | my-volume    | in-use          | ␣
↪  1 | Attached to private-instance on /dev/vdb     |
+--------------------------------------+--------------+-----------------+--
↪----+----------------------------------------------+
```

For more information about how to manage volumes, see the Manage volumes.

For more information about how HNAS driver works, see Hitachi NAS Platform iSCSI and NFS drives for OpenStack.

Quobyte Storage for OpenStack

Quobyte Cinder Driver

To use the `Quobyte` Cinder backend, enable and configure the `Quobyte` Cinder driver in `/etc/kolla/globals.yml`.

```
enable_cinder_backend_quobyte: "yes"
```

Also set values for `quobyte_storage_host` and `quobyte_storage_volume` in `/etc/kolla/globals.yml` to the hostname or IP address of the Quobyte registry and the Quobyte volume respectively.

Since `Quobyte` is proprietary software that requires a license, the use of this backend requires the `Quobyte` Client software package to be installed in the `cinder-volume` and `nova-compute` containers. To do this follow the steps outlined in the Building Container Images, particularly the `Package Customisation` and `Custom Repos` sections. The repository information is available in the `Quobyte` customer portal.

Manila - Shared filesystems service

Overview

Currently, Kolla can deploy following manila services:

- manila-api
- manila-data
- manila-scheduler
- manila-share

The OpenStack Shared File Systems service (Manila) provides file storage to a virtual machine. The Shared File Systems service provides an infrastructure for managing and provisioning of file shares. The service also enables management of share types as well as share snapshots if a driver supports them.

Important

For simplicity, this guide describes configuring the Shared File Systems service to use the `generic` back end with the driver handles share server mode (DHSS) enabled that uses Compute (nova), Networking (neutron) and Block storage (cinder) services. Networking service configuration requires the capability of networks being attached to a public router in order to create shared networks.

Before you proceed, ensure that Compute, Networking and Block storage services are properly working.

Preparation and Deployment

Cinder is required, enable it in `/etc/kolla/globals.yml`:

```
enable_cinder: "yes"
```

Enable Manila and generic back end in `/etc/kolla/globals.yml`:

```
enable_manila: "yes"
enable_manila_backend_generic: "yes"
```

By default Manila uses instance flavor id 100 for its file systems. For Manila to work, either create a new nova flavor with id 100 (use *nova flavor-create*) or change *service_instance_flavor_id* to use one of the default nova flavor ids. Ex: *service_instance_flavor_id = 2* to use nova default flavor `m1.small`.

Create or modify the file `/etc/kolla/config/manila-share.conf` and add the contents:

```
[generic]
service_instance_flavor_id = 2
```

Verify Operation

Verify operation of the Shared File Systems service. List service components to verify successful launch of each process:

```
# manila service-list

+------------------+----------------+------+---------+-------+-------------
↪---------------+-----------------+
|      Binary      |    Host        | Zone |  Status | State |
↪Updated_at          | Disabled Reason |
+------------------+----------------+------+---------+-------+-------------
↪---------------+-----------------+
| manila-scheduler | controller     | nova | enabled |   up  | 2014-10-
↪18T01:30:54.000000 |       None      |
| manila-share     | share1@generic | nova | enabled |   up  | 2014-10-
↪18T01:30:57.000000 |       None      |
+------------------+----------------+------+---------+-------+-------------
↪---------------+-----------------+
```

Launch an Instance

Before being able to create a share, the manila with the generic driver and the DHSS mode enabled requires the definition of at least an image, a network and a share-network for being used to create a share server. For that back end configuration, the share server is an instance where NFS/CIFS shares are served.

Determine the configuration of the share server

Create a default share type before running manila-share service:

```
# manila type-create default_share_type True

+--------------------------------------+--------------------+------------+-
↪-----------+--------------------------------------+-----------------------
↪--+
| ID                                   | Name               | Visibility |␣
↪is_default | required_extra_specs                 | optional_extra_specs ␣
↪   |
+--------------------------------------+--------------------+------------+-
↪-----------+--------------------------------------+-----------------------
↪--+
| 8a35da28-0f74-490d-afff-23664ecd4f01 | default_share_type | public     |␣
↪-          | driver_handles_share_servers : True | snapshot_support :␣
↪True |
+--------------------------------------+--------------------+------------+-
↪-----------+--------------------------------------+-----------------------
↪--+
```

Create a manila share server image to the Image service:

```
# wget https://tarballs.opendev.org/openstack/manila-image-elements/images/
↪manila-service-image-master.qcow2
# glance image-create --name "manila-service-image" \
  --file manila-service-image-master.qcow2 \
  --disk-format qcow2 --container-format bare \
  --visibility public --progress

[=============================>] 100%
+------------------+--------------------------------------+
| Property         | Value                                |
+------------------+--------------------------------------+
| checksum         | 48a08e746cf0986e2bc32040a9183445     |
| container_format | bare                                 |
| created_at       | 2016-01-26T19:52:24Z                 |
| disk_format      | qcow2                                |
| id               | 1fc7f29e-8fe6-44ef-9c3c-15217e83997c |
| min_disk         | 0                                    |
| min_ram          | 0                                    |
| name             | manila-service-image                 |
| owner            | e2c965830ecc4162a002bf16ddc91ab7     |
| protected        | False                                |
| size             | 306577408                            |
| status           | active                               |
| tags             | []                                   |
```

(continues on next page)

(continued from previous page)

```
| updated_at        | 2016-01-26T19:52:28Z                 |
| virtual_size      | None                                 |
| visibility        | public                               |
+-------------------+--------------------------------------+
```

List available networks to get id and subnets of the private network:

```
+--------------------------------------+---------+--------------------------
↪----------------------------+
| id                                   | name    | subnets                  ⌊
↪                            |
+--------------------------------------+---------+--------------------------
↪----------------------------+
| 0e62efcd-8cee-46c7-b163-d8df05c3c5ad | public  | 5cc70da8-4ee7-4565-be53-
↪b9c011fca011 10.3.31.0/24   |
| 7c6f9b37-76b4-463e-98d8-27e5686ed083 | private | 3482f524-8bff-4871-80d4-
↪5774c2730728 172.16.1.0/24 |
+--------------------------------------+---------+--------------------------
↪----------------------------+
```

Create a shared network

```
# manila share-network-create --name demo-share-network1 \
  --neutron-net-id PRIVATE_NETWORK_ID \
  --neutron-subnet-id PRIVATE_NETWORK_SUBNET_ID

+-------------------+--------------------------------------+
| Property          | Value                                |
+-------------------+--------------------------------------+
| name              | demo-share-network1                  |
| segmentation_id   | None                                 |
| created_at        | 2016-01-26T20:03:41.877838           |
| neutron_subnet_id | 3482f524-8bff-4871-80d4-5774c2730728 |
| updated_at        | None                                 |
| network_type      | None                                 |
| neutron_net_id    | 7c6f9b37-76b4-463e-98d8-27e5686ed083 |
| ip_version        | None                                 |
| nova_net_id       | None                                 |
| cidr              | None                                 |
| project_id        | e2c965830ecc4162a002bf16ddc91ab7     |
| id                | 58b2f0e6-5509-4830-af9c-97f525a31b14 |
| description       | None                                 |
+-------------------+--------------------------------------+
```

Create a flavor (**Required** if you not defined *manila_instance_flavor_id* in `/etc/kolla/config/manila-share.conf` file)

```
# nova flavor-create manila-service-flavor 100 128 0 1
```

Create a share

Create a NFS share using the share network:

```
# manila create NFS 1 --name demo-share1 --share-network demo-share-
↪network1

+-----------------------------+--------------------------------------+
| Property                    | Value                                |
+-----------------------------+--------------------------------------+
| status                      | None                                 |
| share_type_name             | None                                 |
| description                 | None                                 |
| availability_zone           | None                                 |
| share_network_id            | None                                 |
| export_locations            | []                                   |
| host                        | None                                 |
| snapshot_id                 | None                                 |
| is_public                   | False                                |
| task_state                  | None                                 |
| snapshot_support            | True                                 |
| id                          | 016ca18f-bdd5-48e1-88c0-782e4c1aa28c |
| size                        | 1                                    |
| name                        | demo-share1                          |
| share_type                  | None                                 |
| created_at                  | 2016-01-26T20:08:50.502877           |
| export_location             | None                                 |
| share_proto                 | NFS                                  |
| consistency_group_id        | None                                 |
| source_cgsnapshot_member_id | None                                 |
| project_id                  | 48e8c35b2ac6495d86d4be61658975e7     |
| metadata                    | {}                                   |
+-----------------------------+--------------------------------------+
```

After some time, the share status should change from `creating` to `available`:

```
# manila list

+--------------------------------------+-------------+------+-------------
↪+-----------+-----------+--------------------------------------+---------
↪--------------------+-------------------+
| ID                                   | Name        | Size | Share Proto
↪| Status    | Is Public | Share Type Name                      | Host
↪                    | Availability Zone |
+--------------------------------------+-------------+------+-------------
↪+-----------+-----------+--------------------------------------+---------
↪--------------------+-------------------+
| e1e06b14-ba17-48d4-9e0b-ca4d59823166 | demo-share1 | 1    | NFS
↪| available | False     | default_share_type                   |
↪share1@generic#GENERIC      | nova              |
+--------------------------------------+-------------+------+-------------
↪+-----------+-----------+--------------------------------------+---------
↪--------------------+-------------------+
```

Configure user access to the new share before attempting to mount it via the network:

```
# manila access-allow demo-share1 ip INSTANCE_PRIVATE_NETWORK_IP
```

Mount the share from an instance

Get export location from share

```
# manila show demo-share1

+-----------------------------+-----------------------------------------
↪---------------------------+
| Property                    | Value                                  ␣
↪                           |
+-----------------------------+-----------------------------------------
↪---------------------------+
| status                      | available                              ␣
↪                           |
| share_type_name             | default_share_type                     ␣
↪                           |
| description                 | None                                   ␣
↪                           |
| availability_zone           | nova                                   ␣
↪                           |
| share_network_id            | fa07a8c3-598d-47b5-8ae2-120248ec837f   ␣
↪                           |
| export_locations            |                                        ␣
↪                           |
|                             | path = 10.254.0.3:/shares/share-422dc546-
↪8f37-472b-ac3c-d23fe410d1b6 |
|                             | preferred = False                      ␣
↪                           |
|                             | is_admin_only = False                  ␣
↪                           |
|                             | id = 5894734d-8d9a-49e4-b53e-7154c9ce0882 ␣
↪                           |
|                             | share_instance_id = 422dc546-8f37-472b-
↪ac3c-d23fe410d1b6           |
| share_server_id             | 4782feef-61c8-4ffb-8d95-69fbcc380a52   ␣
↪                           |
| host                        | share1@generic#GENERIC                 ␣
↪                           |
| access_rules_status         | active                                 ␣
↪                           |
| snapshot_id                 | None                                   ␣
↪                           |
| is_public                   | False                                  ␣
↪                           |
| task_state                  | None                                   ␣
↪                           |
| snapshot_support            | True                                   ␣
↪                           |
| id                          | e1e06b14-ba17-48d4-9e0b-ca4d59823166   ␣
↪                           |
| size                        | 1                                      ␣
↪                           |
| name                        | demo-share1                            ␣
↪                           |
```

(continues on next page)

(continued from previous page)

```
| share_type                  | 6e1e803f-1c37-4660-a65a-c1f2b54b6e17
↪                             |
| has_replicas                | False
↪                             |
| replication_type            | None
↪                             |
| created_at                  | 2016-03-15T18:59:12.000000
↪                             |
| share_proto                 | NFS
↪                             |
| consistency_group_id        | None
↪                             |
| source_cgsnapshot_member_id | None
↪                             |
| project_id                  | 9dc02df0f2494286ba0252b3c81c01d0
↪                             |
| metadata                    | {}
↪                             |
+-----------------------------+--------------------------------------
↪----------------------------+
```

Create a folder where the mount will be placed:

```
# mkdir ~/test_folder
```

Mount the NFS share in the instance using the export location of the share:

```
# mount -v 10.254.0.3:/shares/share-422dc546-8f37-472b-ac3c-d23fe410d1b6 ~/
↪test_folder
```

Share Migration

As administrator, you can migrate a share with its data from one location to another in a manner that is transparent to users and workloads. You can use manila client commands to complete a share migration.

For share migration, is needed modify `manila.conf` and set a ip in the same provider network for `data_node_access_ip`.

Modify the file `/etc/kolla/config/manila.conf` and add the contents:

```
[DEFAULT]
data_node_access_ip = 10.10.10.199
```

Note: Share migration requires have more than one back end configured. For details, see *Configure multiple back ends*.

Use the manila migration command, as shown in the following example:

```
# manila migration-start --preserve-metadata True|False \
  --writable True|False --force_host_assisted_migration True|False \
  --new_share_type share_type --new_share_network share_network \
  shareID destinationHost
```

- `--force-host-copy`: Forces the generic host-based migration mechanism and bypasses any driver optimizations.
- `destinationHost`: Is in this format `host#pool` which includes destination host and pool.
- `--writable` and `--preserve-metadata`: Are only for driver assisted.
- `--new_share_network`: Only if driver supports shared network.
- `--new_share_type`: Choose share type compatible with destinationHost.

Checking share migration progress

Use the **`manila migration-get-progress shareID`** command to check progress.

```
# manila migration-get-progress demo-share1

+----------------+-----------------------+
| Property       | Value                 |
+----------------+-----------------------+
| task_state     | data_copying_starting |
| total_progress | 0                     |
+----------------+-----------------------+

# manila migration-get-progress demo-share1
+----------------+-------------------------+
| Property       | Value                   |
+----------------+-------------------------+
| task_state     | data_copying_completing |
| total_progress | 100                     |
+----------------+-------------------------+
```

Use the **`manila migration-complete shareID`** command to complete share migration process.

For more information about how to manage shares, see the Manage shares.

GlusterFS

We have support for enabling Manila to provide users access to volumes from an external GlusterFS. For more details on the GlusterfsShareDriver, please see: https://docs.openstack.org/manila/latest/admin/glusterfs_driver.html

Kolla-ansible supports using the GlusterFS shares with NFS. To enable this backend, add the following to `/etc/kolla/globals.yml`:

```
enable_manila_backend_glusterfs_nfs: "yes"
```

Layouts

A layout is a strategy of allocating storage from GlusterFS backends for shares. Currently there are two layouts implemented:

volume mapped layout

You will also need to add the following configuration options to ensure the driver can connect to GlusterFS and exposes the correct subset of existing volumes in the system by adding the following in `/etc/kolla/globals.yml`:

```
manila_glusterfs_servers:
  - glusterfs1.example.com
  - glusterfs2.example.com
manila_glusterfs_ssh_user: "root"
manila_glusterfs_ssh_password: "<glusterfs ssh password>"
manila_glusterfs_volume_pattern: "manila-share-volume-\\d+$"
```

The `manila_glusterfs_ssh_password` and `manila_glusterfs_ssh_user` configuration options are only required when the GlusterFS server runs remotely rather than on the system running the Manila share service.

directory mapped layout

You will also need to add the following configuration options to ensure the driver can connect to GlusterFS and exposes the correct subset of existing volumes in the system by adding the following in `/etc/kolla/globals.yml`:

```
manila_glusterfs_share_layout: "layout_directory.
↪GlusterfsDirectoryMappedLayout"
manila_glusterfs_target: "root@10.0.0.1:/volume"
manila_glusterfs_ssh_password: "<glusterfs ssh password>"
manila_glusterfs_mount_point_base: "$state_path/mnt"
```

- `manila_glusterfs_target`: If its of the format <username>@<glustervolserver>:/<glustervolid>, then we ssh to <username>@<glustervolserver> to execute gluster (<username> is supposed to have administrative privileges on <glustervolserver>).
- `manila_glusterfs_ssh_password`: configuration options are only required when the GlusterFS server runs remotely rather than on the system running the Manila share service.

Hitachi NAS Platform File Services Driver for OpenStack

Overview

The Hitachi NAS Platform File Services Driver for OpenStack provides NFS Shared File Systems to OpenStack.

Requirements

- Hitachi NAS Platform Models 3080, 3090, 4040, 4060, 4080, and 4100.
- HNAS/SMU software version is 12.2 or higher.
- HNAS configuration and management utilities to create a storage pool (span) and an EVS.
  - GUI (SMU).
  - SSC CLI.

Supported shared file systems and operations

The driver supports CIFS and NFS shares.

The following operations are supported:

- Create a share.
- Delete a share.
- Allow share access.
- Deny share access.
- Create a snapshot.
- Delete a snapshot.
- Create a share from a snapshot.
- Extend a share.
- Shrink a share.
- Manage a share.
- Unmanage a share.

Preparation and Deployment

Note: The manila-share node only requires the HNAS EVS data interface if you plan to use share migration.

Important: It is mandatory that HNAS management interface is reachable from the Shared File System node through the admin network, while the selected EVS data interface is reachable from OpenStack Cloud, such as through Neutron flat networking.

Configuration on Kolla deployment

Enable Shared File Systems service and HNAS driver in `/etc/kolla/globals.yml`

```
enable_manila: "yes"
enable_manila_backend_hnas: "yes"
```

Configure the OpenStack networking so it can reach HNAS Management interface and HNAS EVS Data interface.

To configure two physical networks, physnet1 and physnet2, with ports eth1 and eth2 associated respectively:

In `/etc/kolla/globals.yml` set:

```
neutron_bridge_name: "br-ex,br-ex2"
neutron_external_interface: "eth1,eth2"
```

Note: `eth1` is used to Neutron external interface and `eth2` is used to HNAS EVS data interface.

HNAS back end configuration

In `/etc/kolla/globals.yml` uncomment and set:

```
hnas_ip: "172.24.44.15"
hnas_user: "supervisor"
hnas_password: "supervisor"
hnas_evs_id: "1"
hnas_evs_ip: "10.0.1.20"
hnas_file_system_name: "FS-Manila"
```

Configuration on HNAS

Create the data HNAS network in Kolla OpenStack:

List the available tenants:

```
$ openstack project list
```

Create a network to the given tenant (service), providing the tenant ID, a name for the network, the name of the physical network over which the virtual network is implemented, and the type of the physical mechanism by which the virtual network is implemented:

```
$ neutron net-create --tenant-id <SERVICE_ID> hnas_network \
  --provider:physical_network=physnet2 --provider:network_type=flat
```

Optional - List available networks:

```
$ neutron net-list
```

Create a subnet to the same tenant (service), the gateway IP of this subnet, a name for the subnet, the network ID created before, and the CIDR of subnet:

```
$ neutron subnet-create --tenant-id <SERVICE_ID> --gateway <GATEWAY> \
  --name hnas_subnet <NETWORK_ID> <SUBNET_CIDR>
```

Optional - List available subnets:

```
$ neutron subnet-list
```

Add the subnet interface to a router, providing the router ID and subnet ID created before:

```
$ neutron router-interface-add <ROUTER_ID> <SUBNET_ID>
```

Create a file system on HNAS. See the Hitachi HNAS reference.

Important: Make sure that the filesystem is not created as a replication target. Refer official HNAS administration guide.

Prepare the HNAS EVS network.

Create a route in HNAS to the tenant network:

```
$ console-context --evs <EVS_ID_IN_USE> route-net-add --gateway <FLAT_
↪NETWORK_GATEWAY> \
  <TENANT_PRIVATE_NETWORK>
```

Important: Make sure multi-tenancy is enabled and routes are configured per EVS.

```
$ console-context --evs 3 route-net-add --gateway 192.168.1.1 \
  10.0.0.0/24
```

Create a share

Create a default share type before running manila-share service:

```
$ manila type-create default_share_hitachi False

+--------------------------------------+-----------------------+-----------
↪-+------------+--------------------------------------+--------------------
↪------+
| ID                                   | Name                  |␣
↪visibility | is_default | required_extra_specs                 |␣
↪optional_extra_specs     |
+--------------------------------------+-----------------------+-----------
↪-+------------+--------------------------------------+--------------------
↪------+
| 3e54c8a2-1e50-455e-89a0-96bb52876c35 | default_share_hitachi | public    ␣
↪ | -          | driver_handles_share_servers : False | snapshot_support␣
↪: True |
+--------------------------------------+-----------------------+-----------
↪-+------------+--------------------------------------+--------------------
↪------+
```

(continues on next page)

(continued from previous page)

Create a NFS share using the HNAS back end:

```
$ manila create NFS 1 \
  --name mysharehnas \
  --description "My Manila share" \
  --share-type default_share_hitachi
```

Verify Operation:

```
$ manila list

+--------------------------------------+----------------+------+-----------
↪--+-----------+-----------+----------------------+----------------------
↪---+-------------------+
| ID                                   | Name           | Size | Share␣
↪Proto | Status    | Is Public | Share Type Name      | Host                ␣
↪       | Availability Zone |
+--------------------------------------+----------------+------+-----------
↪--+-----------+-----------+----------------------+----------------------
↪---+-------------------+
| 721c0a6d-eea6-41af-8c10-72cd98985203 | mysharehnas    | 1    | NFS       ␣
↪  | available | False     | default_share_hitachi | control@hnas1#HNAS1 ␣
↪   | nova              |
+--------------------------------------+----------------+------+-----------
↪--+-----------+-----------+----------------------+----------------------
↪---+-------------------+
```

```
$ manila show mysharehnas

+-----------------------------+-----------------------------------------------
↪----------------------+
| Property                    | Value                                         ␣
↪                      |
+-----------------------------+-----------------------------------------------
↪----------------------+
| status                      | available                                     ␣
↪                      |
| share_type_name             | default_share_hitachi                         ␣
↪                      |
| description                 | My Manila share                               ␣
↪                      |
| availability_zone           | nova                                          ␣
↪                      |
| share_network_id            | None                                          ␣
↪                      |
| export_locations            |                                               ␣
↪                      |
|                             | path = 172.24.53.1:/shares/45ed6670-688b-
↪4cf0-bfe7-34956648fb84 |
|                             | preferred = False                             ␣
↪                      |
|                             | is_admin_only = False                         ␣
↪                      |
|                             | id = e81e716f-f1bd-47b2-8a56-2c2f9e33a98e     ␣
↪                      |
```

(continues on next page)

(continued from previous page)

```
|                              | share_instance_id = 45ed6670-688b-4cf0-
↪bfe7-34956648fb84        |
| share_server_id              | None                                    ␣
↪                        |
| host                         | control@hnas1#HNAS1                     ␣
↪                        |
| access_rules_status          | active                                  ␣
↪                        |
| snapshot_id                  | None                                    ␣
↪                        |
| is_public                    | False                                   ␣
↪                        |
| task_state                   | None                                    ␣
↪                        |
| snapshot_support             | True                                    ␣
↪                        |
| id                           | 721c0a6d-eea6-41af-8c10-72cd98985203    ␣
↪                        |
| size                         | 1                                       ␣
↪                        |
| user_id                      | ba7f6d543713488786b4b8cb093e7873        ␣
↪                        |
| name                         | mysharehnas                             ␣
↪                        |
| share_type                   | 3e54c8a2-1e50-455e-89a0-96bb52876c35    ␣
↪                        |
| has_replicas                 | False                                   ␣
↪                        |
| replication_type             | None                                    ␣
↪                        |
| created_at                   | 2016-10-14T14:50:47.000000              ␣
↪                        |
| share_proto                  | NFS                                     ␣
↪                        |
| consistency_group_id         | None                                    ␣
↪                        |
| source_cgsnapshot_member_id  | None                                    ␣
↪                        |
| project_id                   | c3810d8bcc3346d0bdc8100b09abbbf1        ␣
↪                        |
| metadata                     | {}                                      ␣
↪                        |
+------------------------------+-----------------------------------------
↪----------------------+
```

Configure multiple back ends

An administrator can configure an instance of Manila to provision shares from one or more back ends. Each back end leverages an instance of a vendor-specific implementation of the Manila driver API.

The name of the back end is declared as a configuration option share_backend_name within a particular configuration stanza that contains the related configuration options for that back end.

So, in the case of an multiple back ends deployment, it is necessary to change the default share backends before deployment.

Modify the file `/etc/kolla/config/manila.conf` and add the contents:

```
[DEFAULT]
enabled_share_backends = generic,hnas1,hnas2
```

Modify the file `/etc/kolla/config/manila-share.conf` and add the contents:

```
[generic]
share_driver = manila.share.drivers.generic.GenericShareDriver
interface_driver = manila.network.linux.interface.OVSInterfaceDriver
driver_handles_share_servers = True
service_instance_password = manila
service_instance_user = manila
service_image_name = manila-service-image
share_backend_name = GENERIC

[hnas1]
share_backend_name = HNAS1
share_driver = manila.share.drivers.hitachi.hnas.driver.HitachiHNASDriver
driver_handles_share_servers = False
hitachi_hnas_ip = <hnas_ip>
hitachi_hnas_user = <user>
hitachi_hnas_password = <password>
hitachi_hnas_evs_id = <evs_id>
hitachi_hnas_evs_ip = <evs_ip>
hitachi_hnas_file_system_name = FS-Manila1

[hnas2]
share_backend_name = HNAS2
share_driver = manila.share.drivers.hitachi.hnas.driver.HitachiHNASDriver
driver_handles_share_servers = False
hitachi_hnas_ip = <hnas_ip>
hitachi_hnas_user = <user>
hitachi_hnas_password = <password>
hitachi_hnas_evs_id = <evs_id>
hitachi_hnas_evs_ip = <evs_ip>
hitachi_hnas_file_system_name = FS-Manila2
```

For more information about how to manage shares, see the Manage shares.

For more information about how HNAS driver works, see Hitachi NAS Platform File Services Driver for OpenStack.

Swift - Object storage service

Overview

Kolla can deploy a full working Swift setup in either a **all-in-one** or **multinode** setup.

Networking

The following networks are used by Swift:

External API network (`kolla_external_vip_interface`) This network is used by users to access the Swift public API.

Internal API network (`api_interface`) This network is used by users to access the Swift internal API. It is also used by HAProxy to access the Swift proxy servers.

Swift Storage network (`swift_storage_interface`) This network is used by the Swift proxy server to access the account, container and object servers. Defaults to `storage_interface`.

Swift replication network (`swift_replication_network`) This network is used for Swift storage replication traffic. This is optional as the default configuration uses the `swift_storage_interface` for replication traffic.

Disks with a partition table (recommended)

Swift requires block devices to be available for storage. To prepare a disk for use as a Swift storage device, a special partition name and filesystem label need to be added.

The following should be done on each storage node, the example is shown for three disks:

> **Warning:** ALL DATA ON DISK will be LOST!

```
index=0
for d in sdc sdd sde; do
    parted /dev/${d} -s -- mklabel gpt mkpart KOLLA_SWIFT_DATA 1 -1
    sudo mkfs.xfs -f -L d${index} /dev/${d}1
    (( index++ ))
done
```

For evaluation, loopback devices can be used in lieu of real disks:

```
index=0
for d in sdc sdd sde; do
    free_device=$(losetup -f)
    fallocate -l 1G /tmp/$d
    losetup $free_device /tmp/$d
    parted $free_device -s -- mklabel gpt mkpart KOLLA_SWIFT_DATA 1 -1
    sudo mkfs.xfs -f -L d${index} ${free_device}p1
    (( index++ ))
done
```

Disks without a partition table

Kolla also supports unpartitioned disk (filesystem on `/dev/sdc` instead of `/dev/sdc1`) detection purely based on filesystem label. This is generally not a recommended practice but can be helpful for Kolla to take over Swift deployment already using disk like this.

Given hard disks with labels swd1, swd2, swd3, use the following settings in `ansible/roles/swift/defaults/main.yml`.

```
swift_devices_match_mode: "prefix"
swift_devices_name: "swd"
```

Rings

Before running Swift we need to generate **rings**, which are binary compressed files that at a high level let the various Swift services know where data is in the cluster. We hope to automate this process in a future release.

The following example commands should be run from the `operator` node to generate rings for a demo setup. The commands work with **disks with partition table** example listed above. Please modify accordingly if your setup is different.

If using a separate replication network it is necessary to add the replication network IP addresses to the rings. See the Swift documentation for details on how to do that.

Prepare for Rings generating

To prepare for Swift Rings generating, run the following commands to initialize the environment variable and create `/etc/kolla/config/swift` directory:

```
STORAGE_NODES=(192.168.0.2 192.168.0.3 192.168.0.4)
KOLLA_SWIFT_BASE_IMAGE="kolla/centos-source-swift-base:4.0.0"
mkdir -p /etc/kolla/config/swift
```

Generate Object Ring

To generate Swift object ring, run the following commands:

```
docker run \
  --rm \
  -v /etc/kolla/config/swift/:/etc/kolla/config/swift/ \
  $KOLLA_SWIFT_BASE_IMAGE \
  swift-ring-builder \
    /etc/kolla/config/swift/object.builder create 10 3 1

for node in ${STORAGE_NODES[@]}; do
    for i in {0..2}; do
      docker run \
        --rm \
        -v /etc/kolla/config/swift/:/etc/kolla/config/swift/ \
        $KOLLA_SWIFT_BASE_IMAGE \
```

(continues on next page)

(continued from previous page)

```
        swift-ring-builder \
          /etc/kolla/config/swift/object.builder add r1z1-${node}:6000/d$
↪{i} 1;
    done
done
```

Generate Account Ring

To generate Swift account ring, run the following commands:

```
docker run \
  --rm \
  -v /etc/kolla/config/swift/:/etc/kolla/config/swift/ \
  $KOLLA_SWIFT_BASE_IMAGE \
  swift-ring-builder \
    /etc/kolla/config/swift/account.builder create 10 3 1

for node in ${STORAGE_NODES[@]}; do
    for i in {0..2}; do
      docker run \
        --rm \
        -v /etc/kolla/config/swift/:/etc/kolla/config/swift/ \
        $KOLLA_SWIFT_BASE_IMAGE \
        swift-ring-builder \
          /etc/kolla/config/swift/account.builder add r1z1-${node}:6001/d$
↪{i} 1;
    done
done
```

Generate Container Ring

To generate Swift container ring, run the following commands:

```
docker run \
  --rm \
  -v /etc/kolla/config/swift/:/etc/kolla/config/swift/ \
  $KOLLA_SWIFT_BASE_IMAGE \
  swift-ring-builder \
    /etc/kolla/config/swift/container.builder create 10 3 1

for node in ${STORAGE_NODES[@]}; do
    for i in {0..2}; do
      docker run \
        --rm \
        -v /etc/kolla/config/swift/:/etc/kolla/config/swift/ \
        $KOLLA_SWIFT_BASE_IMAGE \
        swift-ring-builder \
          /etc/kolla/config/swift/container.builder add r1z1-${node}:6002/d
↪${i} 1;
    done
done
```

Rebalance

To rebalance the ring files:

```
for ring in object account container; do
  docker run \
    --rm \
    -v /etc/kolla/config/swift/:/etc/kolla/config/swift/ \
    $KOLLA_SWIFT_BASE_IMAGE \
    swift-ring-builder \
      /etc/kolla/config/swift/${ring}.builder rebalance;
done
```

For more information, see the Swift documentation.

Deploying

Enable Swift in `/etc/kolla/globals.yml`:

```
enable_swift : "yes"
```

If you are to deploy multiple policies, override the variable `swift_extra_ring_files` with the list of your custom ring files, .builder and .ring.gz all together. This will append them to the list of default rings.

```
swift_extra_ring_files:
   - object-1.builder
   - object-1.ring.gz
```

Once the rings are in place, deploying Swift is the same as any other Kolla Ansible service:

```
# kolla-ansible deploy -i <path/to/inventory-file>
```

Verification

A very basic smoke test:

```
$ openstack container create mycontainer

+---------------------------------------+--------------+------------------
↪-----------------+
| account                               | container    | x-trans-id
↪                 |
+---------------------------------------+--------------+------------------
↪-----------------+
| AUTH_7b938156dba44de7891f311c751f91d8 | mycontainer  |
↪txb7f05fa81f244117ac1b7-005a0e7803 |
+---------------------------------------+--------------+------------------
↪-----------------+

$ openstack object create mycontainer README.rst
```

(continues on next page)

(continued from previous page)

```
+---------------+--------------+----------------------------------+
| object        | container    | etag                             |
+---------------+--------------+----------------------------------+
| README.rst    | mycontainer  | 2634ecee0b9a52ba403a503cc7d8e988 |
+---------------+--------------+----------------------------------+

$ openstack container show mycontainer

+--------------+------------------------------------------+
| Field        | Value                                    |
+--------------+------------------------------------------+
| account      | AUTH_7b938156dba44de7891f311c751f91d8 |
| bytes_used   | 6684                                     |
| container    | mycontainer                              |
| object_count | 1                                        |
+--------------+------------------------------------------+

$ openstack object store account show

+------------+------------------------------------------+
| Field      | Value                                    |
+------------+------------------------------------------+
| Account    | AUTH_7b938156dba44de7891f311c751f91d8 |
| Bytes      | 6684                                     |
| Containers | 1                                        |
| Objects    | 1                                        |
+------------+------------------------------------------+
```

S3 API

The Swift S3 API can be enabled by setting `enable_swift_s3api` to `true` in `globals.yml`. It is disabled by default. In order to use this API it is necessary to obtain EC2 credentials from Keystone. See the the Swift documentation for details.

Swift Recon

Enable Swift Recon in `/etc/kolla/globals.yml`:

```
enable_swift_recon : "yes"
```

The Swift role in Kolla Ansible is still using the old role format. Unlike many other Kolla Ansible roles, it wont automatically add the new volume to the containers in existing deployments when running *kolla-ansible reconfigure*. Instead we must use the *kolla-ansible upgrade* command, which will remove the existing containers and then put them back again.

Example usage:

```
$ sudo docker exec swift_object_server swift-recon --all`
```

For more information, see the Swift documentation.

6.1.4 Networking

Kolla deploys Neutron by default as OpenStack networking component. This section describes configuring and running Neutron extensions like Networking-SFC, QoS, and so on.

Designate - DNS service

Overview

Designate provides DNSaaS services for OpenStack:

- REST API for domain/record management
- Multi-tenant
- Integrated with Keystone for authentication
- Framework in place to integrate with Nova and Neutron notifications (for auto-generated records)
- Support for Bind9 and Infoblox out of the box

Configuration on Kolla deployment

Enable Designate service in `/etc/kolla/globals.yml`

```
enable_designate: "yes"
```

Configure Designate options in `/etc/kolla/globals.yml`

Important: Designate MDNS node requires the `dns_interface` to be reachable from public network.

```
dns_interface: "eth1"
designate_ns_record:
  - "ns1.sample.openstack.org"
```

Important: If multiple nodes are assigned to be Designate workers, then you must enable a supported coordination backend, currently only `redis` is supported. The backend choice can be overridden via the `designate_coordination_backend` variable. It defaults to `redis` when `redis` is enabled (`enable_redis` is set to `yes`).

The following additional variables are required depending on which backend you intend to use:

Bind9 Backend

Configure Designate options in `/etc/kolla/globals.yml`

```
designate_backend: "bind9"
```

Infoblox Backend

Important: When using Infoblox as the Designate backend the MDNS node requires the container to listen on port 53. As this is a privilaged port you will need to build your designate-mdns container to run as the user root rather than designate.

Configure Designate options in `/etc/kolla/globals.yml`

```
designate_backend: "infoblox"
designate_backend_infoblox_nameservers: "192.168.1.1,192.168.1.2"
designate_infoblox_host: "192.168.1.1"
designate_infoblox_wapi_url: "https://infoblox.example.com/wapi/v2.1/"
designate_infoblox_auth_username: "username"
designate_infoblox_ns_group: "INFOBLOX"
```

Configure Designate options in `/etc/kolla/passwords.yml`

```
designate_infoblox_auth_password: "password"
```

For more information about how the Infoblox backend works, see Infoblox backend.

Neutron and Nova Integration

Create default Designate Zone for Neutron:

```
openstack zone create --email admin@sample.openstack.org sample.openstack.
↪org.
```

Create designate-sink custom configuration folder:

```
mkdir -p /etc/kolla/config/designate/
```

Append Designate Zone ID in `/etc/kolla/config/designate/designate-sink.conf`

```
[handler:nova_fixed]
zone_id = <ZONE_ID>
[handler:neutron_floatingip]
zone_id = <ZONE_ID>
```

Reconfigure Designate:

```
kolla-ansible reconfigure -i <INVENTORY_FILE> --tags designate,neutron,nova
```

Verify operation

List available networks:

```
openstack network list
```

Associate a domain to a network:

```
openstack network set <NETWORK_ID> --dns-domain sample.openstack.org.
```

Start an instance:

```
openstack server create \
  --image cirros \
  --flavor m1.tiny \
  --key-name mykey \
  --nic net-id=${NETWORK_ID} \
  my-vm
```

Check DNS records in Designate:

```
openstack recordset list sample.openstack.org.

+--------------------------------------+--------------------------------------
↪----+------+----------------------------------------------+--------+------
↪--+
| id                                   | name
↪     | type | records                                      | status |
↪action |
+--------------------------------------+--------------------------------------
↪----+------+----------------------------------------------+--------+------
↪--+
| 5aec6f5b-2121-4a2e-90d7-9e4509f79506 | sample.openstack.org.
↪     | SOA  | sample.openstack.org.                        | ACTIVE | NONE
↪   |
|                                      |
↪     |      | admin.sample.openstack.org. 1485266928 3514 |        |
↪   |
|                                      |
↪     |      | 600 86400 3600                               |        |
↪   |
| 578dc94a-df74-4086-a352-a3b2db9233ae | sample.openstack.org.
↪     | NS   | sample.openstack.org.                        | ACTIVE | NONE
↪   |
| de9ff01e-e9ef-4a0f-88ed-6ec5ecabd315 | 192-168-190-232.sample.openstack.
↪org. | A    | 192.168.190.232                              | ACTIVE |
↪NONE   |
| f67645ee-829c-4154-a988-75341050a8d6 | my-vm.None.sample.openstack.org.
↪     | A    | 192.168.190.232                              | ACTIVE | NONE
↪   |
| e5623d73-4f9f-4b54-9045-b148e0c3342d | my-vm.sample.openstack.org.
↪     | A    | 192.168.190.232                              | ACTIVE | NONE
↪   |
+--------------------------------------+--------------------------------------
↪----+------+----------------------------------------------+--------+------
↪--+
```

Query instance DNS information to Designate `dns_interface` IP address:

```
dig +short -p 5354 @<DNS_INTERFACE_IP> my-vm.sample.openstack.org. A
192.168.190.232
```

For more information about how Designate works, see Designate, a DNSaaS component for OpenStack.

DPDK

Introduction

Open vSwitch (ovs) is an open source software virtual switch developed and distributed via openvswitch.org. The Data Plane Development Kit (dpdk) is a collection of userspace libraries and tools that facilitate the development of high-performance userspace networking applications.

As of the ovs 2.2 release, the ovs netdev datapath has supported integration with dpdk for accelerated userspace networking. As of the pike release of kolla support for deploying ovs with dpdk (ovs-dpdk) has been added to kolla ansible. The ovs-dpdk role introduced in the pike release has been tested on centos 7 and ubuntu 16.04 hosts, however, ubuntu is recommended due to conflicts with the cgroup configuration created by the default systemd version shipped with centos 7.

Prerequisites

DPDK is a high-performance userspace networking library, as such it has several requirements to function correctly that are not required when deploying ovs without dpdk.

To function efficiently one of the mechanisms dpdk uses to accelerate memory access is the utilisation of kernel hugepages. The use of hugepage memory minimises the chance of a translation lookaside buffer(TLB) miss when translating virtual to physical memory as it increases the total amount of addressable memory that can be cached via the TLB. Hugepage memory pages are unswappable contiguous blocks of memory of typically 2MiB or 1GiB in size, that can be used to facilitate efficient sharing of memory between guests and a vSwitch or DMA mapping between physical nics and the userspace ovs datapath.

To deploy ovs-dpdk on a platform a proportion of system memory should be allocated hugepages. While it is possible to allocate hugepages at runtime it is advised to allocate them via the kernel command line instead to prevent memory fragmentation. This can be achieved by adding the following to the grub config and regenerating your grub file.

```
default_hugepagesz=2M hugepagesz=2M hugepages=25000
```

As dpdk is a userspace networking library it requires userspace compatible drivers to be able to control the physical interfaces on the platform. dpdk technically support 3 kernel drivers `igb_uio`, `uio_pci_generic` and `vfio_pci`. While it is technically possible to use all 3 only `uio_pci_generic` and `vfio_pci` are recommended for use with kolla. `igb_uio` is BSD licenced and distributed as part of the dpdk library. While it has some advantages over `uio_pci_generic` loading the `igb_uio` module will taint the kernel and possibly invalidate distro support. To successfully deploy `ovs-dpdk`, `vfio_pci` or `uio_pci_generic` kernel module must be present on the platform. Most distros include `vfio_pci` or `uio_pci_generic` as part of the default kernel though on some distros you may need to install `kernel-modules-extra` or the distro equivalent prior to running **`kolla-ansible deploy`**.

Installation

To enable ovs-dpdk, add the following configuration to `/etc/kolla/globals.yml` file:

```
ovs_datapath: "netdev"
enable_ovs_dpdk: yes
enable_openvswitch: yes
tunnel_interface: "dpdk_bridge"
neutron_bridge_name: "dpdk_bridge"
```

Note: Kolla doesnt support ovs-dpdk for RHEL-based distros due to the lack of a suitable package.

Unlike standard Open vSwitch deployments, the interface specified by neutron_external_interface should have an ip address assigned. The ip address assigned to neutron_external_interface will be moved to the dpdk_bridge as part of deploy action. When using ovs-dpdk the tunnel_interface must be an ovs bridge with a physical interfaces attached for tunnelled traffic to be accelerated by dpdk. Note that due to a limitation in ansible variable names which excluded the use of - in a variable name it is not possible to use the default br-ex name for the neutron_bridge_name or tunnel_interface.

At present, the tunnel interface ip is configured using network manager on on ubuntu and systemd on centos family operating systems. systemd is used to work around a limitation of the centos network manager implementation which does not consider the creation of an ovs bridge to be a hotplug event. In the future, a new config option will be introduced to allow systemd to be used on all host distros for those who do not wish to enable the network manager service on ubuntu.

Limitations

Reconfiguration from kernel ovs to ovs dpdk is currently not supported. Changing ovs datapaths on a deployed node requires neutron config changes and libvirt xml changes for all running instances including a hard reboot of the vm.

When upgrading ovs-dpdk it should be noted that this will always involve a dataplane outage. Unlike kernel OVS the dataplane for ovs-dpdk executes in the ovs-vswitchd process. This means the lifetime of the dpdk dataplane is tied to the lifetime of the ovsdpdk_vswitchd container. As such it is recommended to always evacuate all vm workloads from a node running ovs-dpdk prior to upgrading.

On ubuntu network manager is required for tunnel networking. This requirement will be removed in the future.

Neutron - Networking Service

Preparation and deployment

Neutron is enabled by default in `/etc/kolla/globals.yml`:

```
#enable_neutron: "{{ enable_openstack_core | bool }}"
```

Network interfaces

Neutron external interface is used for communication with the external world, for example provider networks, routers and floating IPs. For setting up the neutron external interface modify `/etc/kolla/globals.yml` setting `neutron_external_interface` to the desired interface name. This interface is used by hosts in the `network` group. It is also used by hosts in the `compute` group if `enable_neutron_provider_networks` is set or DVR is enabled.

The interface is plugged into a bridge (Open vSwitch or Linux Bridge, depending on the driver) defined by `neutron_bridge_name`, which defaults to `br-ex`. The default Neutron physical network is `physnet1`.

Example: single interface

In the case where we have only a single Neutron external interface, configuration is simple:

```
neutron_external_interface: "eth1"
```

Example: multiple interfaces

In some cases it may be necessary to have multiple external network interfaces. This may be achieved via comma-separated lists:

```
neutron_external_interface: "eth1,eth2"
neutron_bridge_name: "br-ex1,br-ex2"
```

These two lists are zipped together, such that `eth1` is plugged into the `br-ex1` bridge, and `eth2` is plugged into the `br-ex2` bridge. Kolla Ansible maps these interfaces to Neutron physical networks `physnet1` and `physnet2` respectively.

Example: shared interface

Sometimes an interface used for Neutron external networking may also be used for other traffic. Plugging an interface directly into a bridge would prevent us from having a usable IP address on the interface. One solution to this issue is to use an intermediate Linux bridge and virtual Ethernet pair, then assign IP addresses on the Linux bridge. This setup is supported by Kayobe. It is out of scope here, as it is non-trivial to set up in a persistent manner.

Provider networks

Provider networks allow to connect compute instances directly to physical networks avoiding tunnels. This is necessary for example for some performance critical applications. Only administrators of OpenStack can create such networks.

To use provider networks in instances you also need to set the following in `/etc/kolla/globals.yml`:

```
enable_neutron_provider_networks: yes
```

For provider networks, compute hosts must have an external bridge created and configured by Ansible (this is also necessary when Neutron Distributed Virtual Routing (DVR) mode is enabled). In this case, ensure `neutron_external_interface` is configured correctly for hosts in the `compute` group.

OpenvSwitch (ml2/ovs)

By default `kolla-ansible` uses `openvswitch` as its underlying network mechanism, you can change that using the `neutron_plugin_agent` variable in `/etc/kolla/globals.yml`:

```
neutron_plugin_agent: "openvswitch"
```

When using Open vSwitch on a compatible kernel (4.3+ upstream, consult the documentation of your distribution for support details), you can switch to using the native OVS firewall driver by employing a configuration override (see *OpenStack Service Configuration in Kolla*). You can set it in `/etc/kolla/config/neutron/openvswitch_agent.ini`:

```
[securitygroup]
firewall_driver = openvswitch
```

OVN (ml2/ovn)

In order to use `OVN` as mechanism driver for `neutron`, you need to set the following:

```
neutron_plugin_agent: "ovn"
```

When using OVN - Kolla Ansible will not enable distributed floating ip functionality (not enable external bridges on computes) by default. To change this behaviour you need to set the following:

```
neutron_ovn_distributed_fip: "yes"
```

Similarly - in order to have Neutron DHCP agents deployed in OVN networking scenario, use:

```
neutron_ovn_dhcp_agent: "yes"
```

This might be desired for example when Ironic bare metal nodes are used as a compute service. Currently OVN is not able to answer DHCP queries on port type external, this is where Neutron agent helps.

Mellanox Infiniband (ml2/mlnx)

In order to add `mlnx_infiniband` to the list of mechanism driver for `neutron` to support Infiniband virtual funtions, you need to set the following (assuming neutron SR-IOV agent is also enabled using `enable_neutron_sriov` flag):

```
enable_neutron_mlnx: "yes"
```

Additionally, you will also need to provide physnet:interface mappings via `neutron_mlnx_physnet_mappings` which is presented to `neutron_mlnx_agent` container via `mlnx_agent.ini` and `neutron_eswitchd` container via `eswitchd.conf`:

```
neutron_mlnx_physnet_mappings:
  ibphysnet: "ib0"
```

Neutron Extensions

Networking-SFC

Preparation and deployment

Modify the `/etc/kolla/globals.yml` file as the following example shows:

```
enable_neutron_sfc: "yes"
```

Verification

For setting up a testbed environment and creating a port chain, please refer to networking-sfc documentation.

Neutron VPNaaS (VPN-as-a-Service)

Preparation and deployment

Modify the `/etc/kolla/globals.yml` file as the following example shows:

```
enable_neutron_vpnaas: "yes"
```

Verification

VPNaaS is a complex subject, hence this document provides directions for a simple smoke test to verify the service is up and running.

On the network node(s), the `neutron_vpnaas_agent` should be up (image naming and versioning may differ depending on deploy configuration):

```
# docker ps --filter name=neutron_vpnaas_agent

CONTAINER ID   IMAGE                                                                                ␣
↪         COMMAND         CREATED          STATUS        PORTS  NAMES
97d25657d55e   operator:5000/kolla/centos-source-neutron-vpnaas-agent:4.0.
↪0   "kolla_start"   44 minutes ago   Up 44 minutes         neutron_vpnaas_
↪agent
```

Warning: You are free to use the following `init-runonce` script for demo purposes but note it does **not** have to be run in order to use your cloud. Depending on your customisations, it may not work, or it may conflict with the resources you want to create. You have been warned.

> Similarly, the `init-vpn` script does **not** have to be run unless you want to follow this particular demo.

Kolla Ansible includes a small script that can be used in tandem with `tools/init-runonce` to verify the VPN using two routers and two Nova VMs:

```
tools/init-runonce
tools/init-vpn
```

Verify both VPN services are active:

```
# neutron vpn-service-list

+--------------------------------------+----------+--------------------------
↪--------------+--------+
| id                                   | name     | router_id                ⌴
↪              | status |
+--------------------------------------+----------+--------------------------
↪--------------+--------+
| ad941ec4-5f3d-4a30-aae2-1ab3f4347eb1 | vpn_west | 051f7ce3-4301-43cc-
↪bfbd-7ffd59af539e | ACTIVE |
| edce15db-696f-46d8-9bad-03d087f1f682 | vpn_east | 058842e0-1d01-4230-
↪af8d-0ba6d0da8b1f | ACTIVE |
+--------------------------------------+----------+--------------------------
↪--------------+--------+
```

Two VMs can now be booted, one on vpn_east, the other on vpn_west, and encrypted ping packets observed being sent from one to the other.

For more information on this and VPNaaS in Neutron refer to the Neutron VPNaaS Testing and the OpenStack wiki.

Trunking

The network trunk service allows multiple networks to be connected to an instance using a single virtual NIC (vNIC). Multiple networks can be presented to an instance by connecting it to a single port.

Modify the `/etc/kolla/globals.yml` file as the following example shows:

```
enable_neutron_trunk: "yes"
```

Octavia

Octavia provides load balancing as a service. This guide covers configuration of Octavia for the Amphora driver. See the Octavia documentation for full details. The installation guide is a useful reference.

Enabling Octavia

Enable the octavia service in `globals.yml`:

```
enable_octavia: "yes"
```

Certificates

Octavia requires various TLS certificates for operation. Since the Victoria release, Kolla Ansible supports generating these certificates automatically.

Option 1: Automatically generating Certificates

Kolla Ansible provides default values for the certificate issuer and owner fields. You can customize this via `globals.yml`, for example:

```
octavia_certs_country: US
octavia_certs_state: Oregon
octavia_certs_organization: OpenStack
octavia_certs_organizational_unit: Octavia
```

Generate octavia certificates:

```
kolla-ansible octavia-certificates
```

The certificates and keys will be generated under `/etc/kolla/config/octavia`.

Option 2: Manually generating certificates

Follow the octavia documentation to generate certificates for Amphorae. These should be copied to the Kolla Ansible configuration as follows:

```
cp client_ca/certs/ca.cert.pem /etc/kolla/config/octavia/client_ca.cert.pem
cp server_ca/certs/ca.cert.pem /etc/kolla/config/octavia/server_ca.cert.pem
cp server_ca/private/ca.key.pem /etc/kolla/config/octavia/server_ca.key.pem
cp client_ca/private/client.cert-and-key.pem /etc/kolla/config/octavia/
↪client.cert-and-key.pem
```

The following option should be set in `passwords.yml`, matching the password used to encrypt the CA key:

```
octavia_ca_password: <CA key password>
```

Networking

Octavia worker and health manager nodes must have access to the Octavia management network for communication with Amphorae.

If using a VLAN for the Octavia management network, enable Neutron provider networks:

```
enable_neutron_provider_networks: yes
```

Configure the name of the network interface on the controllers used to access the Octavia management network. If using a VLAN provider network, ensure that the traffic is also bridged to Open vSwitch on the controllers.

```
octavia_network_interface: <network interface on controllers>
```

This interface should have an IP address on the Octavia management subnet.

Registering OpenStack resources

Since the Victoria release, there are two ways to configure Octavia.

1. Kolla Ansible automatically registers resources for Octavia during deployment
2. Operator registers resources for Octavia after it is deployed

The first option is simpler, and is recommended for new users. The second option provides more flexibility, at the cost of complexity for the operator.

Option 1: Automatic resource registration (default, recommended)

For automatic resource registration, Kolla Ansible will register the following resources:

- Nova flavor
- Nova SSH keypair
- Neutron network and subnet
- Neutron security groups

The configuration for these resources may be customised before deployment.

Note that for this to work access to the Nova and Neutron APIs is required. This is true also for the `kolla-ansible genconfig` command and when using Ansible check mode.

Customize Amphora flavor

The default amphora flavor is named `amphora` with 1 VCPUs, 1GB RAM and 5GB disk. you can customize this flavor by changing `octavia_amp_flavor` in `globals.yml`.

See the `os_nova_flavor` Ansible module for details. Supported parameters are:

- `disk`
- `ephemeral` (optional)
- `extra_specs` (optional)
- `flavorid` (optional)
- `is_public` (optional)
- `name`
- `ram`
- `swap` (optional)
- `vcpus`

The following defaults are used:

```
octavia_amp_flavor:
  name: "amphora"
  is_public: no
  vcpus: 1
  ram: 1024
  disk: 5
```

Customise network and subnet

Configure Octavia management network and subnet with `octavia_amp_network` in `globals.yml`. This must be a network that is *accessible from the controllers*. Typically a VLAN provider network is used.

See the `os_network` and `os_subnet` Ansible modules for details. Supported parameters:

The network parameter has the following supported parameters:

- `external` (optional)
- `mtu` (optional)
- `name`
- `provider_network_type` (optional)
- `provider_physical_network` (optional)
- `provider_segmentation_id` (optional)
- `shared` (optional)
- `subnet`

The subnet parameter has the following supported parameters:

- allocation_pool_start (optional)
- allocation_pool_end (optional)
- cidr
- enable_dhcp (optional)
- gateway_ip (optional)
- name
- no_gateway_ip (optional)
- ip_version (optional)
- ipv6_address_mode (optional)
- ipv6_ra_mode (optional)

For example:

```
octavia_amp_network:
  name: lb-mgmt-net
  provider_network_type: vlan
  provider_segmentation_id: 1000
  provider_physical_network: physnet1
  external: false
  shared: false
  subnet:
    name: lb-mgmt-subnet
    cidr: "10.1.2.0/24"
    allocation_pool_start: "10.1.2.100"
    allocation_pool_end: "10.1.2.200"
    gateway_ip: "10.1.2.1"
    enable_dhcp: yes
```

Deploy Octavia with Kolla Ansible:

```
kolla-ansible -i <inventory> deploy --tags common,horizon,octavia
```

Once the installation is completed, you need to *register an amphora image in glance*.

Option 2: Manual resource registration

In this case, Kolla Ansible will not register resources for Octavia. Set octavia_auto_configure to no in globals.yml:

```
octavia_auto_configure: no
```

All resources should be registered in the service project. This can be done as follows:

```
. /etc/kolla/octavia-openrc.sh
```

Note: Ensure that you have executed kolla-ansible post-deploy and set enable_octavia to yes in global.yml

Note: In Train and earlier releases, resources should be registered in the `admin` project. This is configured via `octavia_service_auth_project`, and may be set to `service` to avoid a breaking change when upgrading to Ussuri. Changing the project on an existing system requires at a minimum registering a new security group in the new project. Ideally the flavor and network should be recreated in the new project, although this will impact existing Amphorae.

Amphora flavor

Register the flavor in Nova:

```
openstack flavor create --vcpus 1 --ram 1024 --disk 2 "amphora" --private
```

Make a note of the ID of the flavor, or specify one via `--id`.

Keypair

Register the keypair in Nova:

```
openstack keypair create --public-key <path to octavia public key> octavia_
↪ssh_key
```

Network and subnet

Register the management network and subnet in Neutron. This must be a network that is *accessible from the controllers*. Typically a VLAN provider network is used.

```
OCTAVIA_MGMT_SUBNET=192.168.43.0/24
OCTAVIA_MGMT_SUBNET_START=192.168.43.10
OCTAVIA_MGMT_SUBNET_END=192.168.43.254

openstack network create lb-mgmt-net --provider-network-type vlan --
↪provider-segment 107  --provider-physical-network physnet1
openstack subnet create --subnet-range $OCTAVIA_MGMT_SUBNET --allocation-
↪pool \
  start=$OCTAVIA_MGMT_SUBNET_START,end=$OCTAVIA_MGMT_SUBNET_END \
  --network lb-mgmt-net lb-mgmt-subnet
```

Make a note of the ID of the network.

Security group

Register the security group in Neutron.

```
openstack security group create lb-mgmt-sec-grp
openstack security group rule create --protocol icmp lb-mgmt-sec-grp
openstack security group rule create --protocol tcp --dst-port 22 lb-mgmt-
↪sec-grp
openstack security group rule create --protocol tcp --dst-port 9443 lb-
↪mgmt-sec-grp
```

Make a note of the ID of the security group.

Kolla Ansible configuration

The following options should be added to `globals.yml`.

Set the IDs of the resources registered previously:

```
octavia_amp_boot_network_list: <ID of lb-mgmt-net>
octavia_amp_secgroup_list: <ID of lb-mgmt-sec-grp>
octavia_amp_flavor_id: <ID of amphora flavor>
```

Now deploy Octavia:

```
kolla-ansible -i <inventory> deploy --tags common,horizon,octavia
```

Amphora image

It is necessary to build an Amphora image. On CentOS / RHEL 8:

```
sudo dnf -y install epel-release
sudo dnf install -y debootstrap qemu-img git e2fsprogs policycoreutils-
↪python-utils
```

On Ubuntu:

```
sudo apt -y install debootstrap qemu-utils git kpartx
```

Acquire the Octavia source code:

```
git clone https://opendev.org/openstack/octavia -b <branch>
```

Install `diskimage-builder`, ideally in a virtual environment:

```
python3 -m venv dib-venv
source dib-venv/bin/activate
pip install diskimage-builder
```

Create the Amphora image:

```
cd octavia/diskimage-create
./diskimage-create.sh
```

Source octavia user openrc:

```
. /etc/kolla/octavia-openrc.sh
```

Note: Ensure that you have executed `kolla-ansible post-deploy`

Register the image in Glance:

```
openstack image create amphora-x64-haproxy.qcow2 --container-format bare --
↪disk-format qcow2 --private --tag amphora --file amphora-x64-haproxy.
↪qcow2 --property hw_architecture='x86_64' --property hw_rng_model=virtio
```

Note: the tag should match the `octavia_amp_image_tag` in `/etc/kolla/globals.yml`, by default, the tag is amphora, octavia uses the tag to determine which image to use.

Debug

SSH to an amphora

login into one of octavia-worker nodes, and ssh into amphora.

```
ssh -i /etc/kolla/octavia-worker/octavia_ssh_key ubuntu@<amphora_ip>
```

Note: amphora private key is located at `/etc/kolla/octavia-worker/octavia_ssh_key` on all octavia-worker nodes.

Upgrade

If you upgrade from the Ussuri release, you must disable `octavia_auto_configure` in `globals.yml` and keep your other octavia config as before.

Development or Testing

Kolla Ansible provides a simple way to setup Octavia networking for development or testing, when using the Neutron Open vSwitch ML2 mechanism driver. In this case, Kolla Ansible will create a tenant network and configure Octavia control services to access it. Please do not use this option in production, the network may not be reliable enough for production.

Add `octavia_network_type` to `globals.yml` and set the value to `tenant`

```
octavia_network_type: "tenant"
```

Nextfollow the deployment instructions as normal.

SRIOV

Neutron SRIOV

Preparation and deployment

SRIOV requires specific NIC and BIOS configuration and is not supported on all platforms. Consult NIC and platform specific documentation for instructions on enablement.

Modify the `/etc/kolla/globals.yml` file as the following example shows which automatically appends `sriovnicswitch` to the `mechanism_drivers` inside `ml2_conf.ini`.

```
enable_neutron_sriov: "yes"
```

It is also a requirement to define physnet:interface mappings for all SRIOV devices as shown in the following example where `sriovtenant1` is the physnet mapped to `ens785f0` interface:

```
neutron_sriov_physnet_mappings:
  sriovtenant1: ens785f0
```

However, the provider networks using SRIOV should be configured. Both flat and VLAN are configured with the same physical network name in this example:

```
[ml2_type_vlan]
network_vlan_ranges = sriovtenant1:1000:1009

[ml2_type_flat]
flat_networks = sriovtenant1
```

Modify the `nova.conf` file and add `PciPassthroughFilter` to `enabled_filters`. This filter is required by the Nova Scheduler service on the controller node.

```
[filter_scheduler]
enabled_filters = <existing filters>, PciPassthroughFilter
available_filters = nova.scheduler.filters.all_filters
```

PCI devices listed under `neutron_sriov_physnet_mappings` will be whitelisted on the Compute hosts inside `nova.conf`.

Physical network to interface mappings in `neutron_sriov_physnet_mappings` will be automatically added to `sriov_agent.ini`. Specific VFs can be excluded via `excluded_devices`. However, leaving blank (default) leaves all VFs enabled:

```
[sriov_nic]
exclude_devices =
```

Run deployment.

Verification

Check that VFs were created on the compute node(s). VFs will appear in the output of both `lspci` and `ip link show`. For example:

```
# lspci | grep net
05:10.0 Ethernet controller: Intel Corporation 82599 Ethernet Controller␣
↪Virtual Function (rev 01)


# ip -d link show ens785f0
4: ens785f0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq master␣
↪ovs-system state UP mode DEFAULT qlen 1000
link/ether 90:e2:ba:ba:fb:20 brd ff:ff:ff:ff:ff:ff promiscuity 1
openvswitch_slave addrgenmode eui64
vf 0 MAC 52:54:00:36:57:e0, spoof checking on, link-state auto, trust off
vf 1 MAC 52:54:00:00:62:db, spoof checking on, link-state auto, trust off
vf 2 MAC fa:16:3e:92:cf:12, spoof checking on, link-state auto, trust off
vf 3 MAC fa:16:3e:00:a3:01, vlan 1000, spoof checking on, link-state auto,␣
↪trust off
```

Verify the SRIOV Agent container is running on the compute node(s):

```
# docker ps --filter name=neutron_sriov_agent
CONTAINER ID   IMAGE                                                          ␣
↪          COMMAND         CREATED          STATUS          PORTS   NAMES
b03a8f4c0b80   10.10.10.10:4000/registry/centos-source-neutron-sriov-
↪agent:17.04.0  "kolla_start"  18 minutes ago  Up 18 minutes           ␣
↪neutron_sriov_agent
```

Verify the SRIOV Agent service is present and UP:

```
# openstack network agent list

+--------------------------------------+--------------------+-------------
↪+-------------------+-------+-------+---------------------------+
| ID                                   | Agent Type         | Host        ␣
↪| Availability Zone | Alive | State | Binary                    |
+--------------------------------------+--------------------+-------------
↪+-------------------+-------+-------+---------------------------+
| 7c06bda9-7b87-487e-a645-cc6c289d9082 | NIC Switch agent   | av09-18-wcp␣
↪| None              | :-)   | UP    | neutron-sriov-nic-agent   |
+--------------------------------------+--------------------+-------------
↪+-------------------+-------+-------+---------------------------+
```

Create a new provider network. Set `provider-physical-network` to the physical network name that was configured in `/etc/kolla/config/nova.conf`. Set `provider-network-type` to the desired type. If using VLAN, ensure `provider-segment` is set to the correct VLAN ID. This example uses `VLAN` network type:

```
# openstack network create --project=admin \
  --provider-network-type=vlan \
  --provider-physical-network=sriovtenant1 \
  --provider-segment=1000 \
  sriovnet1
```

Create a subnet with a DHCP range for the provider network:

```
# openstack subnet create --network=sriovnet1 \
  --subnet-range=11.0.0.0/24 \
  --allocation-pool start=11.0.0.5,end=11.0.0.100 \
  sriovnet1_sub1
```

Create a port on the provider network with `vnic_type` set to `direct`:

```
# openstack port create --network sriovnet1 --vnic-type=direct sriovnet1-
↪port1
```

Start a new instance with the SRIOV port assigned:

```
# openstack server create --flavor flavor1 \
  --image fc-26 \
  --nic port-id=`openstack port list | grep sriovnet1-port1 | awk '{print
↪$2}'` \
  vm1
```

Verify the instance boots with the SRIOV port. Verify VF assignment by running dmesg on the compute node where the instance was placed.

```
# dmesg
[ 2896.849970] ixgbe 0000:05:00.0: setting MAC fa:16:3e:00:a3:01 on VF 3
[ 2896.850028] ixgbe 0000:05:00.0: Setting VLAN 1000, QOS 0x0 on VF 3
[ 2897.403367] vfio-pci 0000:05:10.4: enabling device (0000 -> 0002)
```

For more information see OpenStack SRIOV documentation.

Nova SRIOV

Preparation and deployment

Nova provides a separate mechanism to attach PCI devices to instances that is independent from Neutron. Using the PCI alias configuration option in nova.conf, any PCI device (PF or VF) that supports passthrough can be attached to an instance. One major drawback to be aware of when using this method is that the PCI alias option uses a devices product id and vendor id only, so in environments that have NICs with multiple ports configured for SRIOV, it is impossible to specify a specific NIC port to pull VFs from.

Modify the file `/etc/kolla/config/nova.conf`. The Nova Scheduler service on the control node requires the `PciPassthroughFilter` to be added to the list of filters and the Nova Compute service(s) on the compute node(s) need PCI device whitelisting. The Nova API service on the control node and the Nova Compute service on the compute node also require the `alias` option under the `[pci]` section. The alias can be configured as type-VF to pass VFs or type-PF to pass the PF. Type-VF is shown in this example:

```
[filter_scheduler]
enabled_filters = <existing filters>, PciPassthroughFilter
available_filters = nova.scheduler.filters.all_filters

[pci]
passthrough_whitelist = [{"vendor_id": "8086", "product_id": "10fb"}]
alias = [{"vendor_id":"8086", "product_id":"10ed", "device_type":"type-VF",
↪ "name":"vf1"}]
```

(continues on next page)

(continued from previous page)

Run deployment.

Verification

Create (or use an existing) flavor, and then configure it to request one PCI device from the PCI alias:

```
# openstack flavor set sriov-flavor --property "pci_passthrough:alias"=
↪"vf1:1"
```

Start a new instance using the flavor:

```
# openstack server create --flavor sriov-flavor --image fc-26 vm2
```

Verify VF devices were created and the instance starts successfully as in the Neutron SRIOV case.

For more information see OpenStack PCI passthrough documentation.

6.1.5 Shared services

This section describes configuring different shared service options like backends, dashboards and so on.

Glance - Image service

Glance backends

Overview

Glance can be deployed using Kolla and supports the following backends:

- file
- ceph
- vmware
- swift

File backend

When using the `file` backend, images will be stored locally under the value of the `glance_file_datadir_volume` variable, which defaults to a docker volume called `glance`. By default when using `file` backend only one `glance-api` container can be running.

For better reliability and performance, `glance_file_datadir_volume` should be mounted under a shared filesystem such as NFS.

Usage of glance file backend under shared filesystem:

```
glance_backend_file: "yes"
glance_file_datadir_volume: "/path/to/shared/storage/"
```

Ceph backend

To make use of `ceph` backend in glance, simply enable external ceph. By default will enable backend ceph automatically. Please refer to *External Ceph* on how to configure this backend.

To enable the ceph backend manually:

```
glance_backend_ceph: "yes"
```

VMware backend

To make use of VMware datastores as a glance backend, enable *glance_backend_vmware* and refer to *VMware - Nova Virtualisation Driver* for further VMware configuration.

To enable the vmware backend manually:

```
glance_backend_vmware: "yes"
```

Swift backend

To store glance images in a swift cluster, the `swift` backend should be enabled. Refer to *Swift - Object storage service* on how to configure swift in kolla. If ceph is enabled, will have higher precedence over swift as glance backend.

To enable the swift backend manually:

```
glance_backend_swift: "yes"
```

Upgrading glance

Overview

Glance can be upgraded with the following methods:

- Rolling upgrade
- Legacy upgrade

Rolling upgrade

As of the Rocky release, glance can be upgraded in a rolling upgrade mode. This mode will reduce the API downtime during upgrade to a minimum of a container restart, aiming for zero downtime in future releases.

By default it is disabled, so if you want to upgrade using this mode it will need to be enabled.

```
glance_enable_rolling_upgrade: "yes"
```

> **Warning:** When using glance backend `file` without a shared filesystem, this method cannot be used or will end up with a corrupt state of glance services. Reasoning behind is because glance api is only running in one host, blocking the orchestration of a rolling upgrade.

Legacy upgrade

This upgrade method will stop APIs during database schema migrations, and container restarts.

It is the default mode, ensure rolling upgrade method is not enabled.

```
glance_enable_rolling_upgrade: "no"
```

Other configuration

Glance cache

Glance cache is disabled by default, it can be enabled by:

```
enable_glance_image_cache: "yes"
glance_cache_max_size: "10737418240" # 10GB by default
```

> **Warning:** When using the ceph backend, is recommended to not use glance cache, since nova already has a cached version of the image, and the image is directly copied from ceph instead of glance api hosts. Enabling glance cache will lead to unnecessary storage consumption.

Glance caches are not cleaned up automatically, the glance team recommends to use a cron service to regularly clean cached images. In the future kolla will deploy a cron container to manage such clean ups. Please refer to Glance image cache.

Property protection

Property protection is disabled by default, it can be enabled by:

```
glance_enable_property_protection: "yes"
```

and defining `property-protections-rules.conf` under `{{ node_custom_config }}/glance/`. The default `property_protection_rule_format` is `roles` but it can be overwritten.

Interoperable image import

The interoperable image import is disabled by default, it can be enabled by:

```
glance_enable_interoperable_image_import: "yes"
```

and defining `glance-image-import.conf` under `{{ node_custom_config }}/glance/`.

Horizon - OpenStack dashboard

Overview

Kolla can deploy a full working Horizon dashboard setup in either a **all-in-one** or **multinode** setup.

Extending the default local_settings options

It is possible to extend the default configuration options for Horizon by using a custom python settings file that will override the default options set on the local_settings file.

As an example, for setting a different (material) theme as the default one, a file named custom_local_settings should be created under the directory `{{ node_custom_config }}/horizon/` with the following contents:

```
AVAILABLE_THEMES = [
            ('material', 'Material', 'themes/material'),
]
```

Keystone - Identity service

Tokens

The Keystone token provider is configured via `keystone_token_provider`. The default value for this is `fernet`.

Fernet Tokens

Fernet tokens require the use of keys that must be synchronised between Keystone servers. Kolla Ansible deploys two containers to handle this - `keystone_fernet` runs cron jobs to rotate keys via rsync when necessary. `keystone_ssh` is an SSH server that provides the transport for rsync. In a multi-host control plane, these rotations are performed by the hosts in a round-robin manner.

The following variables may be used to configure the token expiry and key rotation.

`fernet_token_expiry` Keystone fernet token expiry in seconds. Default is 86400, which is 1 day.

`fernet_token_allow_expired_window` Keystone window to allow expired fernet tokens. Default is 172800, which is 2 days.

`fernet_key_rotation_interval` Keystone fernet key rotation interval in seconds. Default is sum of token expiry and allow expired window, which is 3 days.

The default rotation interval is set up to ensure that the minimum number of keys may be active at any time. This is one primary key, one secondary key and a buffer key - three in total. If the rotation interval is set lower than the sum of the token expiry and token allow expired window, more active keys will be configured in Keystone as necessary.

Further infomation on Fernet tokens is available in the Keystone documentation.

Federated identity

Keystone allows users to be authenticated via identity federation. This means integrating OpenStack Keystone with an identity provider. The use of identity federation allows users to access OpenStack services without the necessity of an account in the OpenStack environment per se. The authentication is then off-loaded to the identity provider of the federation.

To enable identity federation, you will need to execute a set of configurations in multiple OpenStack systems. Therefore, it is easier to use Kolla Ansible to execute this process for operators.

For upstream documentations, please see Configuring Keystone for Federation

Supported protocols

OpenStack supports both OpenID Connect and SAML protocols for federated identity, but for now, kolla Ansible supports only OpenID Connect. Therefore, if you desire to use SAML in your environment, you will need to set it up manually or extend Kolla Ansible to also support it.

Setting up OpenID Connect via Kolla Ansible

First, you will need to register the OpenStack (Keystone) in your Identity provider as a Service Provider.

After registering Keystone, you will need to add the Identity Provider configurations in your kolla-ansible globals configuration as the example below:

```
keystone_identity_providers:
  - name: "myidp1"
    openstack_domain: "my-domain"
    protocol: "openid"
```

(continues on next page)

(continued from previous page)

```
    identifier: "https://accounts.google.com"
    public_name: "Authenticate via myidp1"
    attribute_mapping: "mappingId1"
    metadata_folder: "path/to/metadata/folder"
    certificate_file: "path/to/certificate/file.pem"

keystone_identity_mappings:
  - name: "mappingId1"
    file: "/full/qualified/path/to/mapping/json/file/to/mappingId1"
```

In some cases its necessary to add JWKS (JSON Web Key Set) uri. It is required for auth-openidc endpoint - which is used by OpenStack command line client. Example config shown below:

```
keystone_federation_oidc_jwks_uri: "https://<AUTH PROVIDER>/<ID>/discovery/
↪v2.0/keys"
```

Identity providers configurations

name

The internal name of the Identity provider in OpenStack.

openstack_domain

The OpenStack domain that the Identity Provider belongs.

protocol

The federated protocol used by the IdP; e.g. openid or saml. We support only OpenID connect right now.

identifier

The Identity provider URL; e.g. https://accounts.google.com .

public_name

The Identity provider public name that will be shown for users in the Horizon login page.

attribute_mapping

The attribute mapping to be used for the Identity Provider. This mapping is expected to already exist in OpenStack or be configured in the *keystone_identity_mappings* property.

metadata_folder

Path to the folder containing all of the identity provider metadata as JSON files.

The metadata folder must have all your Identity Providers configurations, the name of the files will be the name (with path) of the Issuer configuration. Such as:

```
- <IDP metadata directory>
  - keycloak.example.org%2Fauth%2Frealms%2Fidp.client
  |
  - keycloak.example.org%2Fauth%2Frealms%2Fidp.conf
  |
  - keycloak.example.org%2Fauth%2Frealms%2Fidp.provider
```

Note: The name of the file must be URL-encoded if needed. For example, if you have an Issuer with `/` in the URL, then you need to escape it to `%2F` by applying a URL escape in the file name.

The content of these files must be a JSON

`client`:

The `.client` file handles the Service Provider credentials in the Issuer.

During the first step, when you registered the OpenStack as a Service Provider in the Identity Provider, you submitted a *cliend_id* and generated a *client_secret*, so these are the values you must use in this JSON file.

```
{
  "client_id":"<openid_client_id>",
  "client_secret":"<openid_client_secret>"
}
```

`conf`:

This file will be a JSON that overrides some of the OpenID Connect options. The options that can be overridden are listed in the *OpenID Connect Apache2 plugin documentation*. .. OpenID Connect Apache2 plugin documentation: https://github.com/zmartzone/mod_auth_openidc/wiki/Multiple-Providers#opclient-configuration

If you do not want to override the config values, you can leave this file as an empty JSON file such as `{}`.

`provider`:

This file will contain all specifications about the IdentityProvider. To simplify, you can just use the JSON returned in the `.well-known` Identity providers endpoint:

```
{
  "issuer": "https://accounts.google.com",
  "authorization_endpoint": "https://accounts.google.com/o/oauth2/v2/auth",
  "token_endpoint": "https://oauth2.googleapis.com/token",
  "userinfo_endpoint": "https://openidconnect.googleapis.com/v1/userinfo",
  "revocation_endpoint": "https://oauth2.googleapis.com/revoke",
  "jwks_uri": "https://www.googleapis.com/oauth2/v3/certs",
  "response_types_supported": [
   "code",
   "token",
   "id_token",
   "code token",
   "code id_token",
   "token id_token",
   "code token id_token",
   "none"
  ],
  "subject_types_supported": [
   "public"
  ],
  "id_token_signing_alg_values_supported": [
   "RS256"
  ],
  "scopes_supported": [
   "openid",
   "email",
   "profile"
  ],
  "token_endpoint_auth_methods_supported": [
   "client_secret_post",
   "client_secret_basic"
  ],
  "claims_supported": [
   "aud",
   "email",
   "email_verified",
   "exp",
   "family_name",
   "given_name",
   "iat",
   "iss",
   "locale",
   "name",
   "picture",
   "sub"
  ],
  "code_challenge_methods_supported": [
   "plain",
   "S256"
  ]
}
```

certificate_file

Optional path to the Identity Provider certificate file. If included, the file must be named as certificate-key-id.pem. E.g.:

```
- fb8ca5b7d8d9a5c6c6788071e866c6c40f3fc1f9.pem
```

You can find the key-id in the Identity provider *.well-known/openid-configuration jwks_uri* like in *https://www.googleapis.com/oauth2/v3/certs* :

```
{
  "keys": [
    {
      "e": "AQAB",
      "use": "sig",
      "n": "zK8PHf_6V3G5rU-viUOL1HvAYn7q--dxMoU...",
      "kty": "RSA",
      "kid": "fb8ca5b7d8d9a5c6c6788071e866c6c40f3fc1f9",
      "alg": "RS256"
    }
  ]
}
```

Note: The public key is different from the certificate, the file in this configuration must be the Identity providers certificate and not the Identity providers public key.

6.1.6 Orchestration and NFV

This section describes configuration of orchestration and NFV services.

Tacker - NFV orchestration

Tacker is an OpenStack service for NFV Orchestration with a general purpose VNF Manager to deploy and operate Virtual Network Functions (VNFs) and Network Services on an NFV Platform. It is based on ETSI MANO Architectural Framework. For more details about Tacker, see OpenStack Tacker Documentation.

Overview

As of the Pike release, tacker requires the following services to be enabled to operate correctly.

- Core compute stack (nova, neutron, glance, etc)
- Heat
- Mistral + Redis
- Barbican (Required only for multinode)

Optionally tacker supports the following services and features.

- Aodh

- Ceilometer
- Networking-sfc
- Opendaylight

Compatibility

Tacker is supported by the following distros and install_types.

- CentOS and RHEL: Source and binary images.
- Debian and Ubuntu: Only source images.

Preparation and Deployment

By default tacker and required services are disabled in the `group_vars/all.yml` file. In order to enable them, you need to edit the file `/etc/kolla/globals.yml` and set the following variables:

Note: Heat is enabled by default, ensure it is not disabled.

```
enable_tacker: "yes"
enable_barbican: "yes"
enable_mistral: "yes"
enable_redis: "yes"
```

Warning: Barbican is required in multinode deployments to share VIM fernet_keys. If not enabled, only one tacker-server host will have the keys on it and any request made to a different tacker-server will fail with a similar error as `No such file or directory /etc/tacker/vim/fernet_keys`

Warning: In Train, Tacker started using local filesystem to store VNF packages and CSAR files. Kolla Ansible provides no shared filesystem capabilities, hence only one instance of each Tacker service is deployed and all on the same host.

Deploy tacker and related services.

```
$ kolla-ansible deploy
```

Verification

Generate the credentials file.

```
$ kolla-ansible post-deploy
```

Source credentials file.

```
$ . /etc/kolla/admin-openrc.sh
```

In kolla-ansible git repository a tacker demo is present in `kolla-ansible/contrib/demos/tacker/` that will create a very basic VNF from a cirros image in `demo-net` network.

Install python-tackerclient.

Note: Barbican, heat and mistral python clients are in tackers requirements and will be installed as dependency.

```
$ pip install python-tackerclient
```

Warning: You are free to use the following `init-runonce` script for demo purposes but note it does **not** have to be run in order to use your cloud. Depending on your customisations, it may not work, or it may conflict with the resources you want to create. You have been warned.

From kolla-ansible git repository, execute `init-runonce` and `deploy-tacker-demo` scripts to initialize the demo VNF creation.

```
$ ./tools/init-runonce
$ ./contrib/demos/tacker/deploy-tacker-demo
```

Tacker demo script will create sample VNF Descriptor (VNFD) file, then register a default VIM, create a tacker VNFD and finally deploy a VNF from the previously created VNFD.

After a few minutes, the tacker VNF is ACTIVE with a cirros instance running in nova and with its corresponding heat stack CREATION_COMPLETE.

Verify tacker VNF status is ACTIVE.

```
$ openstack vnf list

+--------------------------------------+------------------+-----------------
↪-------+--------+--------------------------------------+------------------
↪--------------------+
| ID                                   | Name             | Mgmt Url
↪       | Status | VIM ID                               | VNFD ID
↪                    |
+--------------------------------------+------------------+-----------------
↪-------+--------+--------------------------------------+------------------
↪--------------------+
| c52fcf99-101d-427b-8a2d-c9ef54af8b1d | kolla-sample-vnf | {"VDU1": "10.0.
↪0.10"} | ACTIVE | eb3aa497-192c-4557-a9d7-1dff6874a8e6 | 27e8ea98-f1ff-
↪4a40-a45c-e829e53b3c41 |
```

(continues on next page)

(continued from previous page)

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

Verify nova instance status is ACTIVE.

```
$ openstack server list

+--------------------------------------+--------------------------------------
↪--------------------+--------+--------------------+--------+-------------
↪-------------------------------------------------------------------------
↪-----------------------------------+
| ID                                   | Name
↪                     | Status | Networks           | Image  | Flavor      
↪                                                                            
↪                                    |
+--------------------------------------+--------------------------------------
↪--------------------+--------+--------------------+--------+-------------
↪-------------------------------------------------------------------------
↪-----------------------------------+
| d2d59eeb-8526-4826-8f1b-c50b571395e2 | ta-cf99-101d-427b-8a2d-
↪c9ef54af8b1d-VDU1-fchiv6saay7p | ACTIVE | demo-net=10.0.0.10 | cirros |
↪tacker.vnfm.infra_drivers.openstack.openstack_OpenStack-c52fcf99-101d-
↪427b-8a2d-c9ef54af8b1d-VDU1_flavor-yl4bzskwxdkn |
+--------------------------------------+--------------------------------------
↪--------------------+--------+--------------------+--------+-------------
↪-------------------------------------------------------------------------
↪-----------------------------------+
```

Verify Heat stack status is CREATE_COMPLETE.

```
$ openstack stack list

+--------------------------------------+--------------------------------------
↪--------------------------------------------------------------+-------------
↪---------+----------------+----------------------+----------
↪----+
| ID                                   | Stack Name
↪                                                              | Project
↪                     | Stack Status    | Creation Time        | Updated
↪Time |
+--------------------------------------+--------------------------------------
↪--------------------------------------------------------------+-------------
↪---------+----------------+----------------------+----------
↪----+
| 289a6686-70f6-4db7-aa10-ed169fe547a6 | tacker.vnfm.infra_drivers.
↪openstack.openstack_OpenStack-c52fcf99-101d-427b-8a2d-c9ef54af8b1d |
↪1243948e59054aab83dbf2803e109b3f | CREATE_COMPLETE | 2017-08-
↪23T09:49:50Z | None         |
+--------------------------------------+--------------------------------------
↪--------------------------------------------------------------+-------------
↪---------+----------------+----------------------+----------
↪----+
```

After the correct functionality of tacker is verified, tacker demo can be cleaned up executing `cleanup-tacker` script.

```
$ ./cleanup-tacker
```

Warning: The above does not clean up resources created by `init-runonce`.

6.1.7 Logging and monitoring

This section describes configuration for the different logging and monitoring services available in kolla.

Central Logging

An OpenStack deployment generates vast amounts of log data. In order to successfully monitor this and use it to diagnose problems, the standard ssh and grep solution quickly becomes unmanageable.

Preparation and deployment

Modify the configuration file `/etc/kolla/globals.yml` and change the following:

```
enable_central_logging: "yes"
```

Elasticsearch

Kolla deploys Elasticsearch as part of the E*K stack to store, organize and make logs easily accessible.

By default Elasticsearch is deployed on port `9200`.

Note: Elasticsearch stores a lot of logs, so if you are running centralized logging, remember to give `/var/lib/docker` adequate space.

Alternatively it is possible to use a local directory instead of the volume `elasticsearch` to store the data of Elasticsearch. The path can be set via the variable `elasticsearch_datadir_volume`.

Curator

To stop your disks filling up, retention policies can be set. These are enforced by Elasticsearch Curator which can be enabled by setting the following in `/etc/kolla/globals.yml`:

```
enable_elasticsearch_curator: "yes"
```

Elasticsearch Curator is configured via an actions file. The format of the actions file is described in the Elasticsearch Curator documentation. A default actions file is provided which closes indices and then deletes them some time later. The periods for these operations, as well as the prefix for determining which indicies should be managed are defined in the Elasticsearch role defaults and can be overridden in `/etc/kolla/globals.yml` if required.

If the default actions file is not malleable enough, a custom actions file can be placed in the Kolla custom config directory, for example: `/etc/kolla/config/elasticsearch/elasticsearch-curator-actions.yml`.

When testing the actions file you may wish to perform a dry run to be certain of what Curator will actually do. A dry run can be enabled by setting the following in `/etc/kolla/globals.yml`:

```
elasticsearch_curator_dry_run: "yes"
```

The actions which *would* be taken if a dry run were to be disabled are then logged in the Elasticsearch Kolla logs folder under `/var/log/kolla/elasticsearch/elasticsearch-curator.log`.

Kibana

Kolla deploys Kibana as part of the E*K stack in order to allow operators to search and visualise logs in a centralised manner.

After successful deployment, Kibana can be accessed using a browser on `<kolla_external_vip_address>:5601`.

The default username is `kibana`, the password can be located under `<kibana_password>` in `/etc/kolla/passwords.yml`.

First Login

When Kibana is opened for the first time, it requires creating a default index pattern. To view, analyse and search logs, at least one index pattern has to be created. To match indices stored in ElasticSearch, we suggest using the following configuration:

1. Index pattern - flog-*
2. Time Filter field name - @timestamp
3. Expand index pattern when searching [DEPRECATED] - not checked
4. Use event times to create index names [DEPRECATED] - not checked

After setting parameters, one can create an index with the *Create* button.

Search logs - Discover tab

Operators can create and store searches based on various fields from logs, for example, show all logs marked with ERROR on nova-compute.

To do this, click the `Discover` tab. Fields from the logs can be filtered by hovering over entries from the left hand side, and clicking `add` or `remove`. Add the following fields:

- Hostname
- Payload
- severity_label
- programname

This yields an easy to read list of all log events from each node in the deployment within the last 15 minutes. A tail like functionality can be achieved by clicking the clock icon in the top right hand corner of the screen, and selecting `Auto-refresh`.

Logs can also be filtered down further. To use the above example, type `programname:nova-compute` in the search bar. Click the drop-down arrow from one of the results, then the small magnifying glass icon from beside the programname field. This should now show a list of all events from nova-compute services across the cluster.

The current search can also be saved by clicking the `Save Search` icon available from the menu on the right hand side.

Example: using Kibana to diagnose a common failure

The following example demonstrates how Kibana can be used to diagnose a common OpenStack problem, where an instance fails to launch with the error No valid host was found.

First, re-run the server creation with `--debug`:

```
openstack --debug server create --image cirros --flavor m1.tiny \
--key-name mykey --nic net-id=00af016f-dffe-4e3c-a9b8-ec52ccd8ea65 \
demo1
```

In this output, look for the key `X-Compute-Request-Id`. This is a unique identifier that can be used to track the request through the system. An example ID looks like this:

```
X-Compute-Request-Id: req-c076b50a-6a22-48bf-8810-b9f41176a6d5
```

Taking the value of `X-Compute-Request-Id`, enter the value into the Kibana search bar, minus the leading `req-`. Assuming some basic filters have been added as shown in the previous section, Kibana should now show the path this request made through the OpenStack deployment, starting at a `nova-api` on a control node, through the `nova-scheduler`, `nova-conductor`, and finally `nova-compute`. Inspecting the `Payload` of the entries marked `ERROR` should quickly lead to the source of the problem.

While some knowledge is still required of how Nova works in this instance, it can still be seen how Kibana helps in tracing this data, particularly in a large scale deployment scenario.

Visualize data - Visualize tab

In the visualization tab a wide range of charts is available. If any visualization has not been saved yet, after choosing this tab *Create a new visualization* panel is opened. If a visualization has already been saved, after choosing this tab, lately modified visualization is opened. In this case, one can create a new visualization by choosing *add visualization* option in the menu on the right. In order to create new visualization, one of the available options has to be chosen (pie chart, area chart). Each visualization can be created from a saved or a new search. After choosing any kind of search, a design panel is opened. In this panel, a chart can be generated and previewed. In the menu on the left, metrics for a chart can be chosen. The chart can be generated by pressing a green arrow on the top of the left-side menu.

Note: After creating a visualization, it can be saved by choosing *save visualization* option in the menu on the right. If it is not saved, it will be lost after leaving a page or creating another visualization.

Organize visualizations and searches - Dashboard tab

In the Dashboard tab all of saved visualizations and searches can be organized in one Dashboard. To add visualization or search, one can choose *add visualization* option in the menu on the right and then choose an item from all saved ones. The order and size of elements can be changed directly in this place by moving them or resizing. The color of charts can also be changed by checking a colorful dots on the legend near each visualization.

Note: After creating a dashboard, it can be saved by choosing *save dashboard* option in the menu on the right. If it is not saved, it will be lost after leaving a page or creating another dashboard.

If a Dashboard has already been saved, it can be opened by choosing *open dashboard* option in the menu on the right.

Exporting and importing created items - Settings tab

Once visualizations, searches or dashboards are created, they can be exported to a JSON format by choosing Settings tab and then Objects tab. Each of the item can be exported separately by selecting it in the menu. All of the items can also be exported at once by choosing *export everything* option. In the same tab (Settings - Objects) one can also import saved items by choosing *import* option.

Custom log rules

Kolla Ansible automatically deploys Fluentd for forwarding OpenStack logs from across the control plane to a central logging repository. The Fluentd configuration is split into four parts: Input, forwarding, filtering and formatting. The following can be customised:

Custom log filtering

In some scenarios it may be useful to apply custom filters to logs before forwarding them. This may be useful to add additional tags to the messages or to modify the tags to conform to a log format that differs from the one defined by kolla-ansible.

Configuration of custom fluentd filters is possible by placing filter configuration files in `/etc/kolla/config/fluentd/filter/*.conf` on the control host.

Custom log formatting

In some scenarios it may be useful to perform custom formatting of logs before forwarding them. For example, the JSON formatter plugin can be used to convert an event to JSON.

Configuration of custom fluentd formatting is possible by placing filter configuration files in `/etc/kolla/config/fluentd/format/*.conf` on the control host.

Custom log forwarding

In some scenarios it may be useful to forward logs to a logging service other than elasticsearch. This can be done by configuring custom fluentd outputs.

Configuration of custom fluentd outputs is possible by placing output configuration files in `/etc/kolla/config/fluentd/output/*.conf` on the control host.

Custom log inputs

In some scenarios it may be useful to input logs from other services, e.g. network equipment. This can be done by configuring custom fluentd inputs.

Configuration of custom fluentd inputs is possible by placing input configuration files in `/etc/kolla/config/fluentd/input/*.conf` on the control host.

Grafana

Overview

Grafana is open and composable observability and data visualization platform. Visualize metrics, logs, and traces from multiple sources like Prometheus, Loki, Elasticsearch, InfluxDB, Postgres and many more..

Preparation and deployment

To enable Grafana, modify the configuration file `/etc/kolla/globals.yml` and change the following:

```
enable_grafana: "yes"
```

If you would like to set up Prometheus as a data source, additionally set:

```
enable_prometheus: "yes"
```

Please follow *Prometheus Guide* for more information.

Custom dashboards provisioning

Kolla Ansible sets custom dashboards provisioning using Dashboard provider.

Dashboards JSON files should be placed into `{{ node_custom_config }}/grafana/dashboards/` folder. Dashboards will be imported to Grafana dashboards General Folder.

Grafana provisioner config can be altered by placing `provisioning.yaml` to `{{ node_custom_config }}/grafana/` folder.

For other settings, follow configuration reference: Dashboard provider configuration.

InfluxDB - Time Series Database

Overview

InfluxDB is a time series database developed by InfluxData. It is possible to deploy a single instance without charge. To use the clustering features you will require a commercial license.

InfluxDB

The recommendation is to use flash storage for InfluxDB. If docker is configured to use spinning disks by default, or you have some higher performance drives available, it may be desirable to control where the docker volume is located. This can be achieved by setting a path for `influxdb_datadir_volume` in `/etc/kolla/globals.yml`:

```
influxdb_datadir_volume: /mnt/ssd/influxdb/
```

The default is to use a named volume, `influxdb`.

Apache Kafka

Overview

Kafka is a distributed stream processing system. It forms the central component of Monasca and in an OpenStack context can also be used as an experimental messaging backend in Oslo messaging.

Kafka

A spinning disk array is normally sufficient for Kafka. The data directory defaults to a docker volume, `kafka`. Since it can use a lot of disk space, you may wish to store the data on a dedicated device. This can be achieved by setting `kafka_datadir_volume` in `/etc/kolla/globals.yml`:

```
kafka_datadir_volume: /mnt/spinning_array/kafka/
```

Monasca - Monitoring service

Overview

Monasca provides monitoring and logging as-a-service for OpenStack. It consists of a large number of micro-services coupled together by Apache Kafka. If it is enabled in Kolla, it is automatically configured to collect logs and metrics from across the control plane. These logs and metrics are accessible from the Monasca APIs to anyone with credentials for the OpenStack project to which they are posted.

Monasca is not just for the control plane. Monitoring data can just as easily be gathered from tenant deployments, by for example baking the Monasca Agent into the tenant image, or installing it post-deployment using an orchestration tool.

Finally, one of the key tenets of Monasca is that it is scalable. In Kolla Ansible, the deployment has been designed from the beginning to work in a highly available configuration across multiple nodes. Traffic is

typically balanced across multiple instances of a service by HAProxy, or in other cases using the native load balancing mechanism provided by the service. For example, topic partitions in Kafka. Of course, if you start out with a single server thats fine too, and if you find that you need to improve capacity later on down the line, adding additional nodes should be a fairly straightforward exercise.

Pre-deployment configuration

Before enabling Monasca, read the *Security impact* section and decide whether you need to configure a firewall, and/or wish to prevent users from accessing Monasca services.

Enable Monasca in `/etc/kolla/globals.yml`:

```
enable_monasca: "yes"
```

If you wish to disable the alerting and notification pipeline to reduce resource usage you can set `/etc/kolla/globals.yml`:

```
monasca_enable_alerting_pipeline: "no"
```

You can optionally bypass Monasca for control plane logs, and instead have them sent directly to Elasticsearch. This should be avoided if you have deployed Monasca as a standalone service for the purpose of storing logs in a protected silo for security purposes. However, if this is not a relevant consideration, for example you have deployed Monasca alongside the existing OpenStack control plane, then you may free up some resources by setting:

```
monasca_ingest_control_plane_logs: "no"
```

You should note that when making this change with the default `kibana_log_prefix` prefix of `flog-`, you will need to create a new index pattern in Kibana accordingly. If you wish to continue to search all logs using the same index pattern in Kibana, then you can override `kibana_log_prefix` to `monasca` or similar in `/etc/kolla/globals.yml`:

```
kibana_log_prefix: "monasca"
```

If you have enabled Elasticsearch Curator, it will be configured to rotate logs with index patterns matching either `^flog-.*` or `^monasca-.*` by default. If this is undesirable, then you can update the `elasticsearch_curator_index_pattern` variable accordingly.

Currently Monasca is only supported using the `source` install type Kolla images. If you are using the `binary` install type you should set the following override in `/etc/kolla/globals.yml`:

```
monasca_install_type: "source"
```

Stand-alone configuration (optional)

Monasca can be deployed via Kolla Ansible in a standalone configuration. The deployment will include all supporting services such as HAProxy, Keepalived, MariaDB and Memcached. It can also include Keystone, but you will likely want to integrate with the Keystone instance provided by your existing OpenStack deployment. Some reasons to perform a standalone deployment are:

- Your OpenStack deployment is *not* managed by Kolla Ansible, but you want to take advantage of Monasca support in Kolla Ansible.

- Your OpenStack deployment *is* managed by Kolla Ansible, but you do not want the Monasca deployment to share services with your OpenStack deployment. For example, in a combined deployment Monasca will share HAProxy and MariaDB with the core OpenStack services.
- Your OpenStack deployment *is* managed by Kolla Ansible, but you want Monasca to be decoupled from the core OpenStack services. For example, you may have a dedicated monitoring and logging team, and wish to prevent that team accidentally breaking, or redeploying core OpenStack services.
- You want to deploy Monasca for testing. In this case you will likely want to deploy Keystone as well.

To configure a standalone installation you will need to add the following to */etc/kolla/globals.yml*‘:

```
enable_openstack_core: "no"
enable_rabbitmq: "no"
enable_keystone: "yes"
```

With the above configuration alone Keystone *will* be deployed. If you want Monasca to be registered with an external instance of Keystone remove *enable_keystone: yes* from */etc/kolla/globals.yml* and add the following, additional configuration:

```
keystone_admin_url: "http://172.28.128.254:35357"
keystone_internal_url: "http://172.28.128.254:5000"
monasca_openstack_auth:
  auth_url: "{{ keystone_admin_url }}"
  username: "admin"
  password: "{{ external_keystone_admin_password }}"
  project_name: "admin"
  domain_name: "default"
  user_domain_name: "default"
```

In this example it is assumed that the external Keystone admin and internal URLs are *http://172.28.128.254:35357* and *http://172.28.128.254:5000* respectively, and that the external Keystone admin password is defined by the variable *external_keystone_admin_password* which you will most likely want to save in */etc/kolla/passwords.yml*. Note that the Keystone URLs can be obtained from the external OpenStack CLI, for example:

```
openstack endpoint list --service identity
+----------------------------------+-----------+--------------+------------
↪--+---------+-----------+-----------------------------+
| ID                               | Region    | Service Name | Service␣
↪Type | Enabled | Interface | URL                         |
+----------------------------------+-----------+--------------+------------
↪--+---------+-----------+-----------------------------+
| 162365440e6c43d092ad6069f0581a57 | RegionOne | keystone     | identity␣
↪  | True    | admin     | http://172.28.128.254:35357 |
| 6d768ee2ce1c4302a49e9b7ac2af472c | RegionOne | keystone     | identity␣
↪  | True    | public    | http://172.28.128.254:5000  |
| e02067a58b1946c7ae53abf0cfd0bf11 | RegionOne | keystone     | identity␣
↪  | True    | internal  | http://172.28.128.254:5000  |
+----------------------------------+-----------+--------------+------------
↪--+---------+-----------+-----------------------------+
```

If you are also using Kolla Ansible to manage the external OpenStack installation, the external Keystone admin password will most likely be defined in the *external /etc/kolla/passwords.yml* file. For other deployment methods you will need to consult the relevant documentation.

Building images

To build any custom images required by Monasca see the instructions in the Kolla repo: *kolla/doc/source/admin/template-override/monasca.rst*. The remaining images may be pulled from Docker Hub, but if you need to build them manually you can use the following commands:

```
$ kolla-build -t source monasca
$ kolla-build kafka zookeeper storm elasticsearch logstash kibana
```

If you are deploying Monasca standalone you will also need the following images:

```
$ kolla-build cron fluentd mariadb kolla-toolbox keystone memcached␣
↪keepalived haproxy
```

Deployment

Run the deploy as usual, following whichever procedure you normally use to decrypt secrets if you have encrypted them with Ansible Vault:

```
$ kolla-genpwd
$ kolla-ansible deploy
```

Quick start

The first thing you will want to do is to create a Monasca user to view metrics harvested by the Monasca Agent. By default these are saved into the *monasca_control_plane* project, which serves as a place to store all control plane logs and metrics:

```
[vagrant@operator kolla]$ openstack project list
+----------------------------------+------------------------+
| ID                               | Name                   |
+----------------------------------+------------------------+
| 03cb4b7daf174febbc4362d5c79c5be8 | service                |
| 2642bcc8604f4491a50cb8d47e0ec55b | monasca_control_plane  |
| 6b75784f6bc942c6969bc618b80f4a8c | admin                  |
+----------------------------------+------------------------+
```

The permissions of Monasca users are governed by the roles which they have assigned to them in a given OpenStack project. This is an important point and forms the basis of how Monasca supports multi-tenancy.

By default the *admin* role and the *monasca-read-only-user* role are configured. The *admin* role grants read/write privileges and the *monasca-read-only-user* role grants read privileges to a user.

```
[vagrant@operator kolla]$ openstack role list
+----------------------------------+--------------------------+
| ID                               | Name                     |
+----------------------------------+--------------------------+
| 0419463fd5a14ace8e5e1a1a70bbbd84 | agent                    |
| 1095e8be44924ae49585adc5d1136f86 | member                   |
| 60f60545e65f41749b3612804a7f6558 | admin                    |
| 7c184ade893442f78cea8e074b098cfd | _member_                 |
```

(continues on next page)

(continued from previous page)

```
| 7e56318e207a4e85b7d7feeebf4ba396 | reader                 |
| fd200a805299455d90444a00db5074b6 | monasca-read-only-user |
+----------------------------------+------------------------+
```

Now lets consider the example of creating a monitoring user who has read/write privileges in the *monasca_control_plane* project. First we create the user:

```
openstack user create --project monasca_control_plane mon_user
User Password:
Repeat User Password:
+---------------------+----------------------------------+
| Field               | Value                            |
+---------------------+----------------------------------+
| default_project_id  | 2642bcc8604f4491a50cb8d47e0ec55b |
| domain_id           | default                          |
| enabled             | True                             |
| id                  | 088a725872c9410d9c806c24952f9ae1 |
| name                | mon_user                         |
| options             | {}                               |
| password_expires_at | None                             |
+---------------------+----------------------------------+
```

Secondly we assign the user the *admin* role in the *monasca_control_plane* project:

```
openstack role add admin --project monasca_control_plane --user mon_user
```

Alternatively we could have assigned the user the read only role:

```
openstack role add monasca_read_only_user --project monasca_control_plane --
↪user mon_user
```

The user is now active and the credentials can be used to generate an OpenStack token which can be added to the Monasca Grafana datasource in Grafana. For example, first set the OpenStack credentials for the project you wish to view metrics in. This is normally easiest to do by logging into Horizon with the user you have configured for monitoring, switching to the OpenStack project you wish to view metrics in, and then downloading the credentials file for that project. The credentials file can then be sourced from the command line. You can then generate a token for the datasource using the following command:

```
openstack token issue
```

You should then log into Grafana. By default Grafana is available on port *3000* on both internal and external VIPs. See the *Grafana guide* for further details. Once in Grafana you can select the Monasca datasource and add your token to it. You are then ready to view metrics from Monasca.

For log analysis Kibana is also available, by default on port *5601* on both internal and external VIPs. Currently the Keystone authentication plugin is not configured and the HAProxy endpoints are protected by a password which is defined in */etc/kolla/passwords.yml* under *kibana_password*.

Migrating state from an existing Monasca deployment

These steps should be considered after Monasca has been deployed by Kolla. The aim here is to provide some general guidelines on how to migrate service databases. Migration of time series or log data is not considered.

Migrating service databases

The first step is to dump copies of the existing Monasca database. For example:

```
mysqldump -h 10.0.0.1 -u monasca_db_user -p monasca_db > monasca_db.sql
```

This can then be used to replace the Kolla managed Monasca database. Note that it is simplest to get the database password, IP and port from the Monasca API Kolla config file in */etc/kolla/monasca-api*. Also note that the commands below drop and recreate the database before loading in the existing database.

```
mysql -h 192.168.0.1 -u monasca -p -e "drop database monasca; create␣
↪database monasca;"
mysql -h 192.198.0.1 -u monasca -p monasca < monasca_db.sql
```

Migrating passwords

The next step is to set the Kolla Ansible service passwords so that they match the legacy services. The alternative of changing the passwords to match the passwords generated by Kolla Ansible is not considered here.

The passwords which you may wish to set to match the original passwords are:

```
monasca_agent_password:
```

These can be found in the Kolla Ansible passwords file.

Stamping the database with an Alembic revision ID (migrations from pre-Rocky)

Kolla Ansible supports deploying Monasca from the Rocky release onwards. If you are migrating from Queens or below, your database will not have been stamped with a revision ID by Alembic, and this will not be automatic. Support for Alembic migrations was added to Monasca in the Rocky release. You will first need to make sure that the database you have loaded in has been manually migrated to the Queens schema. You can then stamp the database from any Monasca API container running the Rocky release onwards. An example of how this can be done is given below:

```
sudo docker exec -it monasca_api monasca_db stamp --from-fingerprint
```

Applying the configuration

Restart Monasca services on all nodes, for example:

```
for service in `docker ps | grep monasca_ | awk '{print $11}'`; do docker␣
↪restart $service; done
```

Apply the password changes by running the following command:

```
kolla-ansible reconfigure -t monasca
```

Cleanup

From time-to-time it may be necessary to manually invoke the Monasca cleanup command. Normally this will be triggered automatically during an upgrade for services which are removed or disabled by default. However, volume cleanup will always need to be addressed manually. It may also be necessary to run the cleanup command when disabling certain parts of the Monasca pipeline. A full list of scenarios in which you must run the cleanup command is given below. Those marked as automatic will be triggered as part of an upgrade.

- Upgrading from Victoria to Wallaby to remove the unused Monasca Log Transformer service (automatic).
- Upgrading from Victoria to Wallaby to remove the Monasca Log Metrics service, unless the option to disable it by default has been overridden in Wallaby (automatic).
- Upgrading from Wallaby to Xena to remove the Monasca Log Metrics service if the option to disable it by default was overridden in Wallaby (automatic).
- If you have disabled the alerting pipeline via the *monasca_enable_alerting_pipeline* flag after you have deployed the alerting services.

The cleanup command can be invoked from the Kolla Ansible CLI, for example:

```
kolla-ansible monasca_cleanup
```

Following cleanup, you may also choose to remove unused container volumes. It is recommended to run this manually on each Monasca service host. Note that *docker prune* will indiscriminately remove all unused volumes, which may not always be what you want. If you wish to keep a subset of unused volumes, you can remove them individually.

To remove all unused volumes on a host:

```
docker prune
```

To remove a single unused volume, run for example:

```
docker volume rm monasca_log_transformer_data
```

System requirements and performance impact

Monasca will deploy the following Docker containers:

- Apache Kafka
- Apache Storm (optional)
- Apache Zookeeper
- Elasticsearch
- Grafana
- InfluxDB
- Kibana
- Monasca Agent Collector
- Monasca Agent Forwarder
- Monasca Agent Statsd
- Monasca API
- Monasca Log API
- Monasca Log Metrics (Logstash, optional, deprecated)
- Monasca Log Persister (Logstash)
- Monasca Notification (optional)
- Monasca Persister
- Monasca Thresh (Apache Storm topology, optional)

In addition to these, Monasca will also utilise Kolla deployed MariaDB, Keystone, Memcached and HAProxy/Keepalived. The Monasca Agent containers will, by default, be deployed on all nodes managed by Kolla Ansible. This includes all nodes in the control plane as well as compute, storage and monitoring nodes.

Whilst these services will run on an all-in-one deployment, in a production environment it is recommended to use at least one dedicated monitoring node to avoid the risk of starving core OpenStack services of resources. As a general rule of thumb, for a standalone monitoring server running Monasca in a production environment, you will need at least 32GB RAM and a recent multi-core CPU. You will also need enough space to store metrics and logs, and to buffer these in Kafka. Whilst Kafka is happy with spinning disks, you will likely want to use SSDs to back InfluxDB and Elasticsearch.

If resources are tight, it is possible to disable the alerting and notification pipeline which removes the need for Apache Storm, Monasca Thresh and Monasca Notification. This can have a significant effect.

Security impact

The Monasca API, Log API, Grafana and Kibana ports will be exposed on public endpoints via HAProxy/Keepalived. If your public endpoints are exposed externally, then you should use a firewall to restrict access. You should also consider whether you wish to allow tenants to access these services on the internal network.

If you are using the multi-tenant capabilities of Monasca there is a risk that tenants could gain access to other tenants logs and metrics. This could include logs and metrics for the control plane which could reveal sensitive information about the size and nature of the deployment.

Another risk is that users may gain access to system logs via Kibana, which is not accessed via the Monasca APIs. Whilst Kolla configures a password out of the box to restrict access to Kibana, the password will not apply if a user has access to the network on which the individual Kibana service(s) bind behind HAProxy. Note that Elasticsearch, which is not protected by a password, will also be directly accessible on this network, and therefore great care should be taken to ensure that untrusted users do not have access to it.

A full evaluation of attack vectors is outside the scope of this document.

Assignee

Monasca support in Kolla was contributed by StackHPC Ltd. and the Kolla community. If you have any issues with the deployment please ask in the Kolla IRC channel.

OSprofiler - Cross-project profiling

Overview

OSProfiler provides a tiny but powerful library that is used by most (soon to be all) OpenStack projects and their corresponding python clients as well as the Openstack client. It provides functionality to generate 1 trace per request, that goes through all involved services. This trace can then be extracted and used to build a tree of calls which can be quite handy for a variety of reasons (for example in isolating cross-project performance issues).

Configuration on Kolla deployment

Enable `OSprofiler` in `/etc/kolla/globals.yml` file:

```
enable_osprofiler: "yes"
enable_elasticsearch: "yes"
```

Verify operation

Retrieve `osprofiler_secret` key present at `/etc/kolla/passwords.yml`.

Profiler UUIDs can be created executing OpenStack clients (Nova, Glance, Cinder, Heat, Keystone) with `--profile` option or using the official Openstack client with `--os-profile`. In example to get the OSprofiler trace UUID for **`openstack server create`** command.

```
$ openstack --os-profile <OSPROFILER_SECRET> server create \
  --image cirros --flavor m1.tiny --key-name mykey \
  --nic net-id=${NETWORK_ID} demo
```

The previous command will output the command to retrieve OSprofiler trace.

```
$ osprofiler trace show --html <TRACE_ID> --connection-string \
  elasticsearch://<api_interface_address>:9200
```

For more information about how OSprofiler works, see OSProfiler Cross-project profiling library.

Prometheus - Monitoring System & Time Series Database

Overview

Kolla can deploy a full working Prometheus setup in either a **all-in-one** or **multinode** setup.

Preparation and deployment

To enable Prometheus, modify the configuration file `/etc/kolla/globals.yml` and change the following:

```
enable_prometheus: "yes"
```

This will, by default, deploy Prometheus version 2.x. Since Prometheus 1.x data is not compatible with Prometheus 2.x and no automatic data migration is provided, any previous Prometheus 1.x deployment will be replaced and all its stored metrics will become inacessible (but still available in the old data volume: `prometheus`; the new data volume defaults to `prometheus_v2`). If you rely on Prometheus only as e.g. a source of alert notifications (in pair with Alertmanager), it might not be worth migrating old metrics and they could be discarded. Otherwise, its either possible to use remote storage or scrape Kollas Prometheus `/federate` endpoint with an external system. However, if you want to stay on 1.x series, set the following variable:

```
prometheus_use_v1: yes
```

> **Warning:** Support for Prometheus 1.x is deprecated and will be removed in next Kolla Ansible release (Xena).

In order to remove leftover volume containing Prometheus 1.x data, execute:

```
docker volume rm prometheus
```

on all hosts wherever Prometheus was previously deployed.

Extending the default command line options

It is possible to extend the default command line options for Prometheus by using a custom variable. As an example, to set query timeout to 1 minute and data retention size to 30 gigabytes:

```
prometheus_cmdline_extras: "--query.timeout=1m --storage.tsdb.retention.
↪size=30GB"
```

Extending prometheus.cfg

If you want to add extra targets to scrape, you can extend the default `prometheus.yml` config file by placing additional configs in `{{ node_custom_config }}/prometheus/prometheus.yml.d`. These should have the same format as `prometheus.yml`. These additional configs are merged so that any list items are extended. For example, if using the default value for `node_custom_config`, you could add additional targets to scape by defining `/etc/kolla/config/prometheus/prometheus.yml.d/10-custom.yml` containing the following:

```
scrape_configs:
  - job_name: custom
    static_configs:
      - targets:
        - '10.0.0.111:1234'
  - job_name: custom-template
    static_configs:
      - targets:
{% for host in groups['prometheus'] %}
        - '{{ hostvars[host]['ansible_' + hostvars[host]['api_interface']][
↪'ipv4']['address'] }}:{{ 3456 }}'
{% endfor %}
```

The jobs, `custom`, and `custom_template` would be appended to the default list of `scrape_configs` in the final `prometheus.yml`. To customize on a per host basis, files can also be placed in `{{ node_custom_config }}/prometheus/<inventory_hostname>/prometheus.yml.d` where, `inventory_hostname` is one of the hosts in your inventory. These will be merged with any files in `{{ node_custom_config }}/prometheus/prometheus.yml.d`, so in order to override a list value instead of extending it, you will need to make sure that no files in `{{ node_custom_config }}/prometheus/prometheus.yml.d` set a key with an equivalent hierarchical path.

Extra files

Sometimes it is necessary to reference additional files from within `prometheus.yml`, for example, when defining file service discovery configuration. To enable you to do this, kolla-ansible will resursively discover any files in `{{ node_custom_config }}/prometheus/extras` and template them. The templated output is then copied to `/etc/prometheus/extras` within the container on startup. For example to configure ipmi_exporter, using the default value for `node_custom_config`, you could create the following files:

- `/etc/kolla/config/prometheus/prometheus.yml.d/ipmi-exporter.yml`:

```
---
scrape_configs:
- job_name: ipmi
  params:
    module: ["default"]
    scrape_interval: 1m
    scrape_timeout: 30s
    metrics_path: /ipmi
    scheme: http
    file_sd_configs:
      - files:
          - /etc/prometheus/extras/file_sd/ipmi-exporter-
↪targets.yml
    refresh_interval: 5m
    relabel_configs:
      - source_labels: [__address__]
        separator: ;
        regex: (.*)
        target_label: __param_target
        replacement: ${1}
        action: replace
      - source_labels: [__param_target]
        separator: ;
        regex: (.*)
        target_label: instance
        replacement: ${1}
        action: replace
      - separator: ;
        regex: .*
        target_label: __address__
        replacement: "{{ ipmi_exporter_listen_address }}:9290"
        action: replace
```

where `ipmi_exporter_listen_address` is a variable containing the IP address of the node where the exporter is running.

- **`/etc/kolla/config/prometheus/extras/file_sd/ipmi-exporter-targets.yml`:**

```
---
- targets:
  - 192.168.1.1
labels:
    job: ipmi_exporter
```

Skydive - Real time network analyzer

Overview

Skydive is an open source real-time network topology and protocols analyzer. It aims to provide a comprehensive way of understanding what is happening in the network infrastructure. Skydive agents collect topology information and flows and forward them to a central agent for further analysis. All the information is stored in an Elasticsearch database.

Configuration on Kolla deployment

Enable Skydive in `/etc/kolla/globals.yml` file:

```
enable_skydive: "yes"
enable_elasticsearch: "yes"
```

Verify operation

After successful deployment, Skydive can be accessed using a browser on `<kolla_external_vip_address>:8085`.

The default username is `admin`, the password can be located under `<keystone_admin_password>` in `/etc/kolla/passwords.yml`.

For more information about how Skydive works, see Skydive An open source real-time network topology and protocols analyzer.

6.1.8 Containers

This section describes configuring and running container based services including kuryr.

Kuryr - Container networking

Kuryr is a Docker network plugin that uses Neutron to provide networking services to Docker containers. It provides containerized images for the common Neutron plugins. Kuryr requires at least Keystone and neutron. Kolla makes kuryr deployment faster and accessible.

Requirements

- A minimum of 3 hosts for a vanilla deploy

Preparation and Deployment

To allow Docker daemon connect to the etcd, add the following in the `docker.service` file.

```
ExecStart= -H tcp://172.16.1.13:2375 -H unix:///var/run/docker.sock --
↪cluster-store=etcd://172.16.1.13:2379 --cluster-advertise=172.16.1.
↪13:2375
```

The IP address is host running the etcd service. `2375` is port that allows Docker daemon to be accessed remotely. `2379` is the etcd listening port.

By default etcd and kuryr are disabled in the `group_vars/all.yml`. In order to enable them, you need to edit the file globals.yml and set the following variables

```
enable_etcd: "yes"
enable_kuryr: "yes"
```

Deploy the OpenStack cloud and kuryr network plugin

```
kolla-ansible deploy
```

Create a Virtual Network

```
docker network create -d kuryr --ipam-driver=kuryr --subnet=10.1.0.0/24 --
↪gateway=10.1.0.1 docker-net1
```

To list the created network:

```
docker network ls
```

The created network is also available from OpenStack CLI:

```
openstack network list
```

For more information about how kuryr works, see kuryr (OpenStack Containers Networking).

Magnum - Container cluster service

Magnum is an OpenStack service that provides support for deployment and management of container clusters such as Kubernetes. See the Magnum documentation for information on using Magnum.

Configuration

Enable Magnum, in `globals.yml`:

```
enable_magnum: true
```

Optional: enable cluster user trust

This allows the cluster to communicate with OpenStack on behalf of the user that created it, and is necessary for the auto-scaler and auto-healer to work. Note that this is disabled by default since it exposes the cluster to CVE-2016-7404. Ensure that you understand the consequences before enabling this option. In `globals.yml`:

```
enable_cluster_user_trust: true
```

Optional: private CA

If using TLS with a private CA for OpenStack public APIs, the cluster will need to add the CA certificate to its trust store in order to communicate with OpenStack. The certificate must be available in the magnum conductor container. It is copied to the cluster via user-data, so it is better to include only the necessary certificates to avoid exceeding the max Nova API request body size (this may be set via `[oslo_middleware] max_request_body_size` in `nova.conf` if necessary). In `/etc/kolla/config/magnum.conf`:

```
[drivers]
openstack_ca_file = <path to CA file>
```

If using Kolla Ansible to *copy CA certificates into containers*, the certificates are located at `/etc/pki/ca-trust/source/anchors/kolla-customca-*.crt`.

Deployment

To deploy magnum and its dashboard in an existing OpenStack cluster:

```
kolla-ansible -i <inventory> deploy --tags common,horizon,magnum
```

6.1.9 Databases

This section describes configuration of database services.

External MariaDB

Sometimes, for various reasons (Redundancy, organisational policies, etc.), it might be necessary to use an externally managed database. This use case can be achieved by simply taking some extra steps:

Requirements

- An existing MariaDB cluster / server, reachable from all of your nodes.
- If you choose to use preconfigured databases and users (**use_preconfigured_databases** is set to yes), databases and user accounts for all enabled services should exist on the database.
- If you choose not to use preconfigured databases and users (**use_preconfigured_databases** is set to no), root access to the database must be available in order to configure databases and user accounts for all enabled services.

Enabling External MariaDB support

In order to enable external mariadb support, you will first need to disable mariadb deployment, by ensuring the following line exists within `/etc/kolla/globals.yml` :

```
enable_mariadb: "no"
```

There are two ways in which you can use external MariaDB: * Using an already load-balanced MariaDB address * Using an external MariaDB cluster

Using an already load-balanced MariaDB address (recommended)

If your external database already has a load balancer, you will need to do the following:

1. Edit the inventory file, change `control` to the hostname of the load balancer within the `mariadb` group as below:

   ```
   [mariadb]
   myexternalmariadbloadbalancer.com
   ```

2. Define `database_address` in `/etc/kolla/globals.yml` file:

   ```
   database_address: myexternalmariadbloadbalancer.com
   ```

Note: If `enable_external_mariadb_load_balancer` is set to `no` (default), the external DB load balancer should be accessible from all nodes during your deployment.

Using an external MariaDB cluster

Using this way, you need to adjust the inventory file:

```
[mariadb:children]
myexternaldbserver1.com
myexternaldbserver2.com
myexternaldbserver3.com
```

If you choose to use haproxy for load balancing between the members of the cluster, every node within this group needs to be resolvable and reachable from all the hosts within the `[haproxy:children]` group of your inventory (defaults to `[network]`).

In addition, configure the `/etc/kolla/globals.yml` file according to the following configuration:

```
enable_external_mariadb_load_balancer: yes
```

Using External MariaDB with a privileged user

In case your MariaDB user is root, just leave everything as it is within globals.yml (Except the internal mariadb deployment, which should be disabled), and set the `database_password` in `/etc/kolla/passwords.yml` file:

```
database_password: mySuperSecurePassword
```

If the MariaDB `username` is not `root`, set `database_user` in `/etc/kolla/globals.yml` file:

```
database_user: "privillegeduser"
```

Using preconfigured databases / users:

The first step you need to take is to set `use_preconfigured_databases` to `yes` in the `/etc/kolla/globals.yml` file:

```
use_preconfigured_databases: "yes"
```

Note: when the `use_preconfigured_databases` flag is set to `"yes"`, you need to make sure the mysql variable `log_bin_trust_function_creators` set to `1` by the database administrator before running the **`upgrade`** command.

Using External MariaDB with separated, preconfigured users and databases

In order to achieve this, you will need to define the user names in the `/etc/kolla/globals.yml` file, as illustrated by the example below:

```
keystone_database_user: preconfigureduser1
nova_database_user: preconfigureduser2
```

Also, you will need to set the passwords for all databases in the `/etc/kolla/passwords.yml` file

However, fortunately, using a common user across all databases is possible.

Using External MariaDB with a common user across databases

In order to use a common, preconfigured user across all databases, all you need to do is the following steps:

1. Edit the `/etc/kolla/globals.yml` file, add the following:

   ```
   use_common_mariadb_user: "yes"
   ```

2. Set the database_user within `/etc/kolla/globals.yml` to the one provided to you:

   ```
   database_user: mycommondatabaseuser
   ```

3. Set the common password for all components within `/etc/kolla/passwords.yml`. In order to achieve that you could use the following command:

```
sed -i -r -e 's/([a-z_]{0,}database_password:+)(.*)$/\1 mycommonpass/
↪gi' /etc/kolla/passwords.yml
```

MariaDB Guide

Kolla Ansible supports deployment of a MariaDB/Galera cluster for use by OpenStack and other services.

MariaDB Shards

A database shard, or simply a shard, is a horizontal partition of data in a database or search engine. Each shard is held on a separate database server/cluster, to spread load. Some data within a database remains present in all shards, but some appears only in a single shard. Each shard acts as the single source for this subset of data.

Kolla supports sharding on services database level, so every database can be hosted on different shard. Each shard is implemented as an independent Galera cluster.

This section explains how to configure multiple database shards. Currently, only one shard is accessible via the HAProxy load balancer and supported by the `kolla-ansible mariadb_backup` command. This will be improved in future by using ProxySQL, allowing load balanced access to all shards.

Deployment

Each shard is identified by an integer ID, defined by `mariadb_shard_id`. The default shard, defined by `mariadb_default_database_shard_id` (default 0), identifies the shard that will be accessible via HAProxy and available for backing up.

In order to deploy several MariaDB cluster, you will need to edit inventory file in the way described below:

```
[mariadb]
server1ofcluster0
server2ofcluster0
server3ofcluster0
server1ofcluster1 mariadb_shard_id=1
server2ofcluster1 mariadb_shard_id=1
server3ofcluster1 mariadb_shard_id=1
server1ofcluster2 mariadb_shard_id=2
server2ofcluster2 mariadb_shard_id=2
server3ofcluster2 mariadb_shard_id=2
```

Note: If `mariadb_shard_id` is not defined for host in inventory file it will be set automatically to `mariadb_default_database_shard_id` (default 0) from `group_vars/all.yml` and can be overwritten in `/etc/kolla/globals.yml`. Shard which is marked as default is special in case of backup or loadbalance, as it is described below.

Loadbalancer

Kolla currently supports balancing only for default shard. This will be changed in future by replacement of HAProxy with ProxySQL. This results in certain limitations as described below.

Backup and restore

Backup and restore is working only for default shard as kolla currently using HAProxy solution for MariaDB loadbalancer which is simple TCP and has configured only default shard hosts as backends, therefore backup script will reach only default shard on `kolla_internal_vip_address`.

6.1.10 Message queues

This section describes configuration of message queue services.

RabbitMQ

RabbitMQ is a message broker written in Erlang. It is currently the default provider of message queues in Kolla Ansible deployments.

TLS encryption

There are a number of channels to consider when securing RabbitMQ communication. Kolla Ansible currently supports TLS encryption of the following:

- client-server traffic, typically between OpenStack services using the oslo.messaging library and RabbitMQ
- RabbitMQ Management API and UI (frontend connection to HAProxy only)

Encryption of the following channels is not currently supported:

- RabbitMQ cluster traffic between RabbitMQ server nodes
- RabbitMQ CLI communication with RabbitMQ server nodes
- RabbitMQ Management API and UI (backend connection from HAProxy to RabbitMQ)

Client-server

Encryption of client-server traffic is enabled by setting `rabbitmq_enable_tls` to `true`. Additionally, certificates and keys must be available in the following paths (in priority order):

Certificates:

- `"{{ kolla_certificates_dir }}/{{ inventory_hostname }}/rabbitmq-cert.pem"`
- `"{{ kolla_certificates_dir }}/{{ inventory_hostname }}-cert.pem"`
- `"{{ kolla_certificates_dir }}/rabbitmq-cert.pem"`

Keys:

- "{{ kolla_certificates_dir }}/{{ inventory_hostname }}/rabbitmq-key.pem"
- "{{ kolla_certificates_dir }}/{{ inventory_hostname }}-key.pem"
- "{{ kolla_certificates_dir }}/rabbitmq-key.pem"

The default for `kolla_certificates_dir` is `/etc/kolla/certificates`.

The certificates must be valid for the IP address of the host running RabbitMQ on the API network.

Additional TLS configuration options may be passed to RabbitMQ via `rabbitmq_tls_options`. This should be a dict, and the keys will be prefixed with `ssl_options.`. For example:

```
rabbitmq_tls_options:
  ciphers.1: ECDHE-ECDSA-AES256-GCM-SHA384
  ciphers.2: ECDHE-RSA-AES256-GCM-SHA384
  ciphers.3: ECDHE-ECDSA-AES256-SHA384
  honor_cipher_order: true
  honor_ecc_order: true
```

Details on configuration of RabbitMQ for TLS can be found in the RabbitMQ documentation.

When `om_rabbitmq_enable_tls` is `true` (it defaults to the value of `rabbitmq_enable_tls`), applicable OpenStack services will be configured to use oslo.messaging with TLS enabled. The CA certificate is configured via `om_rabbitmq_cacert` (it defaults to `rabbitmq_cacert`, which points to the systems trusted CA certificate bundle for TLS). Note that there is currently no support for using client certificates.

For testing purposes, Kolla Ansible provides the `kolla-ansible certificates` command, which will generate self-signed certificates for RabbitMQ if `rabbitmq_enable_tls` is `true`.

Management API and UI

The management API and UI are accessed via HAProxy, exposed only on the internal VIP. As such, traffic to this endpoint is encrypted when `kolla_enable_tls_internal` is `true`. See *TLS Configuration*.

Passing arguments to RabbitMQ servers Erlang VM

Erlang programs run in an Erlang VM (virtual machine) and use the Erlang runtime. The Erlang VM can be configured.

Kolla Ansible makes it possible to pass arguments to the Erlang VM via the usage of the `rabbitmq_server_additional_erl_args` variable. The contents of it are appended to the `RABBITMQ_SERVER_ADDITIONAL_ERL_ARGS` environment variable which is passed to the RabbitMQ server startup script. Kolla Ansible already configures RabbitMQ server for IPv6 (if necessary). Any argument can be passed there as documented in https://www.rabbitmq.com/runtime.html

The default value for `rabbitmq_server_additional_erl_args` is `+S 2:2 +sbwt none +sbwtdcpu none +sbwtdio none`.

By default RabbitMQ starts N schedulers where N is the number of CPU cores, including hyper-threaded cores. This is fine when you assume all CPUs are dedicated to RabbitMQ. Its not a good idea in a

typical Kolla Ansible setup. Here we go for two scheduler threads (`+S 2:2`). More details can be found here: https://www.rabbitmq.com/runtime.html#scheduling and here: https://erlang.org/doc/man/erl.html#emulator-flags

The `+sbwt none +sbwtdcpu none +sbwtdio none` arguments prevent busy waiting of the scheduler, for more details see: https://www.rabbitmq.com/runtime.html#busy-waiting.

High Availability

RabbitMQ offers two features that, when used together, allow for high availability. These are durable queues and classic queue mirroring. Setting the flag `om_enable_rabbitmq_high_availability` to `true` will enable both of these features. There are some queue types which are intentionally not mirrored using the exclusionary pattern `^(?!(amq\\.)|(.*_fanout_)|(reply_)).*`.

External RabbitMQ

Sometimes, for various reasons (Redundancy, organisational policies, etc.), it might be necessary to use an external RabbitMQ cluster. This use case can be achieved with the following steps:

Requirements

- An existing RabbitMQ cluster, reachable from all of your nodes.

Enabling External RabbitMQ support

In order to enable external RabbitMQ support, you will first need to disable RabbitMQ deployment, by ensuring the following line exists within `/etc/kolla/globals.yml` :

```
enable_rabbitmq: "no"
```

Overwriting transport_url within `globals.yml`

When you use an external RabbitMQ cluster, you must overwrite `*_transport_url` within `/etc/kolla/globals.yml`

```
rpc_transport_url:
notify_transport_url:
nova_cell_rpc_transport_url:
nova_cell_notify_transport_url:
```

For example:

```
rpc_transport_url: rabbit://
 ↪openstack:6Y6Eh3blPXB1Qn4190JKxRoyVhTaFsY2k2V0DuIc@10.0.0.1:5672,
 ↪openstack:6Y6Eh3blPXB1Qn4190JKxRoyVhTaFsY2k2V0DuIc@10.0.0.2:5672,
 ↪openstack:6Y6Eh3blPXB1Qn4190JKxRoyVhTaFsY2k2V0DuIc@10.0.0.3:5672//
notify_transport_url: "{{ rpc_transport_url }}"
```

(continues on next page)

(continued from previous page)

```
nova_cell_rpc_transport_url: rabbit://
↪openstack:6Y6Eh3blPXB1Qn4190JKxRoyVhTaFsY2k2V0DuIc@10.0.0.1:5672//
nova_cell_notify_transport_url: "{{ nova_cell_rpc_transport_url }}"
```

Note: Ensure the rabbitmq user used in `*_transport_url` exists.

6.1.11 Deployment configuration

This section describes configuration of kolla containers, including limiting their resources.

Resource Constraints

Overview

Since the Rocky release it is possible to restrict the resource usage of deployed containers. In Kolla Ansible, container resources to be constrained are referred to as dimensions.

The Docker documentation provides information on container resource constraints. The resources currently supported by Kolla Ansible are:

```
cpu_period
cpu_quota
cpu_shares
cpuset_cpus
cpuset_mems
mem_limit
mem_reservation
memswap_limit
kernel_memory
blkio_weight
ulimits
```

Pre-deployment Configuration

Dimensions are defined as a mapping from a Docker resource name

Table 1: Resource Constraints

Resource	Data Type	Default Value
cpu_period	Integer	0
blkio_weight	Integer	0
cpu_quota	Integer	0
cpu_shares	Integer	0
mem_limit	Integer	0
memswap_limit	Integer	0
mem_reservation	Integer	0
cpuset_cpus	String	(Empty String)
cpuset_mems	String	(Empty String)
ulimits	Dict	{}

The variable `default_container_dimensions` sets the default dimensions for all supported containers, and by default these are unconstrained.

Each supported container has an associated variable, `<container name>_dimensions`, that can be used to set the resources for the container. For example, dimensions for the `nova_libvirt` container are set via the variable `nova_libvirt_dimensions`.

For example, to constrain the number of CPUs that may be used by all supported containers, add the following to the dimensions options section in `/etc/kolla/globals.yml`:

```
default_container_dimensions:
  cpuset_cpus: "1"
```

For example, to constrain the number of CPUs that may be used by the `nova_libvirt` container, add the following to the dimensions options section in `/etc/kolla/globals.yml`:

```
nova_libvirt_dimensions:
  cpuset_cpus: "2"
```

How to config ulimits in kolla

```
<container_name>_dimensions:
  ulimits:
    nofile:
      soft: 131072
      hard: 131072
    fsize:
      soft: 131072
      hard: 131072
```

A list of valid names can be found [here] (https://github.com/docker/go-units/blob/d4a9b9617350c034730bc5051c605919943080bf/ulimit.go#L46-L63)

Deployment

To deploy resource constrained containers, run the deployment as usual:

```
$ kolla-ansible deploy -i /path/to/inventory
```

6.1.12 Deployment and bootstrapping

This section describes deployment and provisioning of baremetal control plane hosts.

Bifrost - Standalone Ironic

From the `Bifrost` developer documentation: Bifrost (pronounced bye-frost) is a set of Ansible playbooks that automates the task of deploying a base image onto a set of known hardware using Ironic. It provides modular utility for one-off operating system deployment with as few operational requirements as reasonably possible.

Kolla uses bifrost as a mechanism for bootstrapping an OpenStack control plane on a set of baremetal servers. Kolla provides a container image for bifrost. Kolla-ansible provides a playbook to configure and deploy the bifrost container, as well as building a base OS image and provisioning it onto the baremetal nodes.

Hosts in the System

In a system deployed by bifrost we define a number of classes of hosts.

Control host The control host is the host on which kolla and kolla-ansible will be installed, and is typically where the cloud will be managed from.

Deployment host The deployment host runs the bifrost deploy container and is used to provision the cloud hosts.

Cloud hosts The cloud hosts run the OpenStack control plane, compute and storage services.

Bare metal compute hosts: In a cloud providing bare metal compute services to tenants via Ironic, these hosts will run the bare metal tenant workloads. In a cloud with only virtualised compute this category of hosts does not exist.

Note: In many cases the control and deployment host will be the same, although this is not mandatory.

Note: Bifrost supports provisioning of bare metal nodes. While kolla-ansible is agnostic to whether the host OS runs on bare metal or is virtualised, in a virtual environment the provisioning of VMs for cloud hosts and their base OS images is currently out of scope.

Cloud Deployment Procedure

Cloud deployment using kolla and bifrost follows the following high level steps:

1. Install and configure kolla and kolla-ansible on the control host.
2. Deploy bifrost on the deployment host.
3. Use bifrost to build a base OS image and provision cloud hosts with this image.
4. Deploy OpenStack services on the cloud hosts provisioned by bifrost.

Preparation

Prepare the Control Host

Follow the **Install dependencies** section of the *Quick Start* guide instructions to set up kolla and kolla-ansible dependencies. Follow the instructions in either the **Install kolla for development** section or the **Install kolla for deployment or evaluation** section to install kolla and kolla-ansible.

Prepare the Deployment Host

RabbitMQ requires that the systems hostname resolves to the IP address that it has been configured to use, which with bifrost will be `127.0.0.1`. Bifrost will attempt to modify `/etc/hosts` on the deployment host to ensure that this is the case. Docker bind mounts `/etc/hosts` into the container from a volume. This prevents atomic renames which will prevent Ansible from fixing the `/etc/hosts` file automatically.

To enable bifrost to be bootstrapped correctly, add an entry to `/etc/hosts` resolving the deployment hosts hostname to `127.0.0.1`, for example:

```
cat /etc/hosts
127.0.0.1 bifrost localhost
```

The following lines are desirable for IPv6 capable hosts:

```
::1 ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
ff02::3 ip6-allhosts
192.168.100.15 bifrost
```

Build a Bifrost Container Image

This section provides instructions on how to build a container image for bifrost using kolla.

Currently kolla only supports the `source` install type for the bifrost image.

1. To generate kolla-build.conf configuration File

   - If required, generate a default configuration file for **`kolla-build`**:

     ```
     cd kolla
     tox -e genconfig
     ```

   - Modify `kolla-build.conf`, setting `install_type` to `source`:

     ```
     install_type = source
     ```

Alternatively, instead of using `kolla-build.conf`, a `source` build can be enabled by appending `--type source` to the **`kolla-build`** or `tools/build.py` command.

1. To build images, for Development:

   ```
   cd kolla
   tools/build.py bifrost-deploy
   ```

   For Production:

   ```
   kolla-build bifrost-deploy
   ```

   Note: By default **`kolla-build`** will build all containers using CentOS as the base image. To change this behavior, use the following parameter with **`kolla-build`** or `tools/build.py` command:

   ```
   --base [centos|debian|rhel|ubuntu]
   ```

Configure and Deploy a Bifrost Container

This section provides instructions for how to configure and deploy a container running bifrost services.

Prepare Kolla Ansible Inventory

Kolla-ansible will deploy bifrost on the hosts in the `bifrost` Ansible group. In the `all-in-one` and `multinode` inventory files, a `bifrost` group is defined which contains all hosts in the `deployment` group. This top level `deployment` group is intended to represent the host running the `bifrost_deploy` container. By default, this group contains `localhost`. See *Multinode Deployment of Kolla* for details on how to modify the Ansible inventory in a multinode deployment.

Bifrost does not currently support running on multiple hosts so the `bifrost` group should contain only a single host, however this is not enforced by kolla-ansible. Bifrost manages a number of services that conflict with services deployed by kolla including OpenStack Ironic, MariaDB, RabbitMQ and

(optionally) OpenStack Keystone. These services should not be deployed on the host on which bifrost is deployed.

Prepare Kolla Ansible Configuration

Follow the instructions in *Quick Start* to prepare kolla-ansibles global configuration file `globals.yml`. For bifrost, the `bifrost_network_interface` variable should be set to the name of the interface that will be used to provision bare metal cloud hosts if this is different than `network_interface`. For example to use `eth1`:

```
bifrost_network_interface: eth1
```

Note that this interface should typically have L2 network connectivity with the bare metal cloud hosts in order to provide DHCP leases with PXE boot options.

Since bifrost only supports the source image type, ensure that this is reflected in `globals.yml`

```
kolla_install_type: source
```

Prepare Bifrost Configuration

Kolla ansible custom configuration files can be placed in a directory given by the `node_custom_config` variable, which defaults do `/etc/kolla/config`. Bifrost configuration files should be placed in this directory or in a `bifrost` subdirectory of it (e.g. `/etc/kolla/config/bifrost`). Within these directories the files `bifrost.yml`, `servers.yml` and `dib.yml` can be used to configure Bifrost.

Create a Bifrost Inventory

The file `servers.yml` defines the bifrost hardware inventory that will be used to populate Ironic. See the bifrost dynamic inventory examples for further details.

For example, the following inventory defines a single node managed via the Ironic `ipmi` driver. The inventory contains credentials required to access the nodes BMC via IPMI, the MAC addresses of the nodes NICs, an IP address to configure the nodes configdrive with, a set of scheduling properties and a logical name.

```
---
cloud1:
  uuid: "31303735-3934-4247-3830-333132535336"
  driver_info:
    power:
      ipmi_username: "admin"
      ipmi_address: "192.168.1.30"
      ipmi_password: "root"
  nics:
    -
      mac: "1c:c1:de:1c:aa:53"
    -
      mac: "1c:c1:de:1c:aa:52"
  driver: "ipmi"
```

(continues on next page)

(continued from previous page)

```
  ipv4_address: "192.168.1.10"
  properties:
    cpu_arch: "x86_64"
    ram: "24576"
    disk_size: "120"
    cpus: "16"
  name: "cloud1"
```

The required inventory will be specific to the hardware and environment in use.

Create Bifrost Configuration

The file `bifrost.yml` provides global configuration for the bifrost playbooks. By default kolla mostly uses bifrosts default variable values. For details on bifrosts variables see the bifrost documentation. For example:

```
mysql_service_name: mysql
ansible_python_interpreter: /var/lib/kolla/venv/bin/python
enabled_hardware_types: ipmi
# uncomment below if needed
# dhcp_pool_start: 192.168.2.200
# dhcp_pool_end: 192.168.2.250
# dhcp_lease_time: 12h
# dhcp_static_mask: 255.255.255.0
```

Create Disk Image Builder Configuration

The file `dib.yml` provides configuration for bifrosts image build playbooks. By default kolla mostly uses bifrosts default variable values when building the baremetal OS and deployment images, and will build an **Ubuntu-based** image for deployment to nodes. For details on bifrosts variables see the bifrost documentation.

For example, to use the `debian` Disk Image Builder OS element:

```
dib_os_element: debian
```

See the diskimage-builder documentation for more details.

Deploy Bifrost

The bifrost container can be deployed either using kolla-ansible or manually.

Deploy Bifrost using Kolla Ansible

For development:

```
cd kolla-ansible
tools/kolla-ansible deploy-bifrost
```

For Production:

```
kolla-ansible deploy-bifrost
```

Deploy Bifrost manually

1. Start Bifrost Container

   ```
   docker run -it --net=host -v /dev:/dev -d \
   --privileged --name bifrost_deploy \
   kolla/ubuntu-source-bifrost-deploy:3.0.1
   ```

2. Copy Configuration Files

   ```
   docker exec -it bifrost_deploy mkdir /etc/bifrost
   docker cp /etc/kolla/config/bifrost/servers.yml bifrost_deploy:/etc/
   ↪bifrost/servers.yml
   docker cp /etc/kolla/config/bifrost/bifrost.yml bifrost_deploy:/etc/
   ↪bifrost/bifrost.yml
   docker cp /etc/kolla/config/bifrost/dib.yml bifrost_deploy:/etc/
   ↪bifrost/dib.yml
   ```

3. Bootstrap Bifrost

   ```
   docker exec -it bifrost_deploy bash
   ```

4. Generate an SSH Key

   ```
   ssh-keygen
   ```

5. Bootstrap and Start Services

   ```
   cd /bifrost
   ./scripts/env-setup.sh
   export OS_CLOUD=bifrost
   cat > /etc/rabbitmq/rabbitmq-env.conf << EOF
   HOME=/var/lib/rabbitmq
   EOF
   ansible-playbook -vvvv \
   -i /bifrost/playbooks/inventory/target \
   /bifrost/playbooks/install.yaml \
   -e @/etc/bifrost/bifrost.yml \
   -e @/etc/bifrost/dib.yml \
   -e skip_package_install=true
   ```

Validate the Deployed Container

```
docker exec -it bifrost_deploy bash
cd /bifrost
export OS_CLOUD=bifrost
```

Running ironic node-list should return with no nodes, for example

```
(bifrost-deploy)[root@bifrost bifrost]# ironic node-list
+------+------+---------------+-------------+--------------------+---------
↪----+
| UUID | Name | Instance UUID | Power State | Provisioning State |␣
↪Maintenance |
+------+------+---------------+-------------+--------------------+---------
↪----+
+------+------+---------------+-------------+--------------------+---------
↪----+
```

Enroll and Deploy Physical Nodes

Once we have deployed a bifrost container we can use it to provision the bare metal cloud hosts specified in the inventory file. Again, this can be done either using kolla-ansible or manually.

By Kolla Ansible

For Development:

```
tools/kolla-ansible deploy-servers
```

For Production:

```
kolla-ansible deploy-servers
```

Manually

```
docker exec -it bifrost_deploy bash
cd /bifrost
export OS_CLOUD=bifrost
export BIFROST_INVENTORY_SOURCE=/etc/bifrost/servers.yml
ansible-playbook -vvvv \
-i /bifrost/playbooks/inventory/bifrost_inventory.py \
/bifrost/playbooks/enroll-dynamic.yaml \
-e "ansible_python_interpreter=/var/lib/kolla/venv/bin/python" \
-e @/etc/bifrost/bifrost.yml

docker exec -it bifrost_deploy bash
cd /bifrost
export OS_CLOUD=bifrost
export BIFROST_INVENTORY_SOURCE=/etc/bifrost/servers.yml
ansible-playbook -vvvv \
```

(continues on next page)

(continued from previous page)

```
-i /bifrost/playbooks/inventory/bifrost_inventory.py \
/bifrost/playbooks/deploy-dynamic.yaml \
-e "ansible_python_interpreter=/var/lib/kolla/venv/bin/python" \
-e @/etc/bifrost/bifrost.yml
```

At this point Ironic should clean down the nodes and install the default OS image.

Advanced Configuration

Bring Your Own Image

TODO

Bring Your Own SSH Key

To use your own SSH key after you have generated the `passwords.yml` file update the private and public keys under `bifrost_ssh_key`.

Known issues

SSH daemon not running

By default `sshd` is installed in the image but may not be enabled. If you encounter this issue you will have to access the server physically in recovery mode to enable the `sshd` service. If your hardware supports it, this can be done remotely with **`ipmitool`** and Serial Over LAN. For example

```
ipmitool -I lanplus -H 192.168.1.30 -U admin -P root sol activate
```

References

- Bifrost documentation
- Bifrost troubleshooting guide
- Bifrost code repository

Bootstrapping Servers

Kolla-ansible provides support for bootstrapping host configuration prior to deploying containers via the `bootstrap-servers` subcommand. This includes support for the following:

- Customisation of `/etc/hosts`
- Creation of user and group
- Kolla configuration directory
- Package installation and removal

- Docker engine installation and configuration
- Disabling firewalls
- Creation of Python virtual environment
- Configuration of Apparmor
- Configuration of SELinux
- Configuration of NTP daemon

All bootstrapping support is provided by the `baremetal` Ansible role.

Running the command

The base command to perform a bootstrap is:

```
kolla-ansible bootstrap-servers -i INVENTORY
```

Further options may be necessary, as described in the following sections.

Initial bootstrap considerations

The nature of bootstrapping means that the environment that Ansible executes in during the initial bootstrap may look different to that seen after bootstrapping is complete. For example:

- The `kolla_user` user account may not yet have been created. If this is normally used as the `ansible_user` when executing Kolla Ansible, a different user account must be used for bootstrapping.
- The Python virtual environment may not exist. If a virtualenv is normally used as the `ansible_python_interpreter` when executing Kolla Ansible, the system python interpreter must be used for bootstrapping.

Each of these variables may be passed via the `-e` argument to Kolla Ansible to override the inventory defaults:

```
kolla-ansible bootstrap-servers -i INVENTORY -e ansible_user=<bootstrap␣
↪user> -e ansible_python_interpreter=/usr/bin/python
```

Subsequent bootstrap considerations

It is possible to run the bootstrapping process against a cloud that has already been bootstrapped, for example to apply configuration from a newer release of Kolla Ansible. In this case, further considerations should be made.

It is possible that the Docker engine package will be updated. This will cause the Docker engine to restart, in addition to all running containers. There are three main approaches to avoiding all control plane services restarting simultaneously.

The first option is to use the `--limit` command line argument to apply the command to hosts in batches, ensuring there is always a quorum for clustered services (e.g. MariaDB):

```
kolla-ansible bootstrap-servers -i INVENTORY --limit controller0,compute[0-
↪1]
kolla-ansible bootstrap-servers -i INVENTORY --limit controller1,compute[2-
↪3]
kolla-ansible bootstrap-servers -i INVENTORY --limit controller2,compute[4-
↪5]
```

The second option is to execute individual plays on hosts in batches:

```
kolla-ansible bootstrap-servers -i INVENTORY -e kolla_serial=30%
```

The last option is to use the Docker `live-restore` configuration option to avoid restarting containers when the Docker engine is restarted. There have been issues reported with using this option however, so use it at your own risk.

Ensure that any operation that causes the Docker engine to be updated has been tested, particularly when moving from legacy Docker packages to Docker Community Edition. See *Package repositories* for details.

Customisation of `/etc/hosts`

This is optional, and enabled by `customize_etc_hosts`, which is `true` by default.

- Ensures that `localhost` is in `/etc/hosts`
- Adds an entry for the IP of the API interface of each host to `/etc/hosts`.

Creation of user and group

This is optional, and enabled by `create_kolla_user`, which is `true` by default.

- Ensures that a group exists with the name defined by the variable `kolla_group` with default `kolla`.
- Ensures that a user exists with the name defined by the variable `kolla_user` with default `kolla`. The users primary group is defined by `kolla_group`. The user is added to the `sudo` group.
- An SSH public key is authorised for `kolla_user`. The key is defined by the `public_key` value of the `kolla_ssh_key` mapping variable, typically defined in `passwords.yml`.
- If the `create_kolla_user_sudoers` variable is set, a sudoers profile will be configured for `kolla_user`, which grants passwordless sudo.

Kolla configuration directory

Kolla ansible service configuration is written to hosts in a directory defined by `node_config_directory`, which by default is `/etc/kolla/`. This directory will be created. If `create_kolla_user` is set, the owner and group of the directory will be set to `kolla_user` and `kolla_group` respectively.

Package installation and removal

Lists of packages are defined for installation and removal. On Debian family systems, these are defined by `debian_pkg_install` and `ubuntu_pkg_removals` respectively. On Red Hat family systems, these are defined by `redhat_pkg_install` and `redhat_pkg_removals` respectively.

Docker engine installation and configuration

Docker engine is a key dependency of Kolla Ansible, and various configuration options are provided.

Package repositories

If the `enable_docker_repo` flag is set, then a package repository for Docker packages will be configured. Kolla Ansible uses the Community Edition packages from https://download.docker.com.

Various other configuration options are available beginning `docker_(apt|yum)_`. Typically these do not need to be changed.

Configuration

The `docker_storage_driver` variable is optional. If set, it defines the storage driver to use for Docker.

The `docker_runtime_directory` variable is optional. If set, it defines the runtime (`data-root`) directory for Docker.

The `docker_registry` variable, which is not set by default, defines the address of the Docker registry. If the variable is not set, Dockerhub will be used.

The `docker_registry_insecure` variable, which defaults to `true` if `docker_registry` is set, or `false` otherwise, defines whether to configure `docker_registry` as an insecure registry. Insecure registries use HTTP rather than HTTPS.

The `docker_log_max_file` variable, which defaults to `5`, defines the maximum number of log files to retain per container. The `docker_log_max_size` variable, which defaults to `50m`, defines the maximum size of each rotated log file per container.

The `docker_http_proxy`, `docker_https_proxy` and `docker_no_proxy` variables can be used to configure Docker Engine to connect to the internet using http/https proxies.

Additional options for the Docker engine can be passed in `docker_custom_config` variable. It will be stored in `daemon.json` config file. Example:

```
{
    "experimental": false
}
```

Disabling firewalls

Kolla Ansible does not support configuration of host firewalls, and instead attempts to disable them.

On Debian family systems where the UFW firewall is enabled, a default policy will be added to allow all traffic.

On Red Hat family systems where firewalld is installed, it will be disabled.

This behaviour can be avoided by setting `disable_firewall` to `false`.

Creation of Python virtual environment

This is optional, and enabled by setting `virtualenv` to a path to a Python virtual environment to create. By default, a virtual environment is not used. If `virtualenv_site_packages` is set, (default is `true`) the virtual environment will inherit packages from the global site-packages directory. This is typically required for modules such as yum and apt which are not available on PyPI. See *Target Hosts* for further information.

Configuration of Apparmor

On Ubuntu systems, the `libvirtd` Apparmor profile will be removed.

Configuration of SELinux

On Red Hat family systems, if `change_selinux` is set (default is `true`), then the SELinux state will be set to `selinux_state` (default `permissive`). See *Kolla Security* for further information.

Configuration of NTP daemon

> **Warning:** Support for configuration of NTP daemon is deprecated and will be removed in the next Kolla Ansible release (Xena). Please use other means of configuring NTP.

This is optional, and enabled by `enable_host_ntp`, which is `false` by default.

6.1.13 High-availability

This section describes high-availability configuration of services.

HAProxy Guide

Kolla Ansible supports a Highly Available (HA) deployment of Openstack and other services. High-availability in Kolla is implented as via Keepalived and HAProxy. Keepalived manages virtual IP addresses, while HAProxy load-balances traffic to service backends. These two components must be installed on the same hosts and they are deployed to hosts in the `haproxy` group.

Preparation and deployment

HAProxy and Keepalived are enabled by default. They may be disabled by setting the following in `/etc/kolla/globals.yml`:

```
enable_haproxy: "no"
enable_keepalived: "no"
```

Configuration

Failover tuning

When a VIP fails over from one host to another, hosts may take some time to detect that the connection has been dropped. This can lead to service downtime.

To reduce the time by the kernel to close dead connections to VIP address, modify the `net.ipv4.tcp_retries2` kernel option by setting the following in `/etc/kolla/globals.yml`:

```
haproxy_host_ipv4_tcp_retries2: 6
```

This is especially helpful for connections to MariaDB. See here, here and here for further information about this kernel option.

CHAPTER

SEVEN

CONTRIBUTOR GUIDE

7.1 Contributor Guide

This guide is for contributors of the Kolla Ansible project. It includes information on proposing your first patch and how to participate in the community. It also covers responsibilities of core reviewers and the Project Team Lead (PTL), and information about development processes.

We welcome everyone to join our project!

7.1.1 So You Want to Contribute

For general information on contributing to OpenStack, please check out the contributor guide to get started. It covers all the basics that are common to all OpenStack projects: the accounts you need, the basics of interacting with our Gerrit review system, how we communicate as a community, etc.

Below will cover the more project specific information you need to get started with Kolla Ansible.

Basics

The source repository for this project can be found at:

https://opendev.org/openstack/kolla-ansible

Communication

Kolla Ansible shares communication channels with Kolla.

IRC Channel `#openstack-kolla` (channel logs) on OFTC

Weekly Meetings On Wednesdays at 15:00 UTC in the IRC channel (meetings logs)

Mailing list (prefix subjects with `[kolla]`) http://lists.openstack.org/pipermail/openstack-discuss/

Meeting Agenda https://wiki.openstack.org/wiki/Meetings/Kolla

Whiteboard (etherpad) Keeping track of CI gate status, release status, stable backports, planning and feature development status. https://etherpad.openstack.org/p/KollaWhiteBoard

Contacting the Core Team

The list in alphabetical order (on first name):

Name	IRC nick	Email address
Chason Chan	chason	chason.chan@foxmail.com
Christian Berendt	berendt	berendt@betacloud-solutions.de
Dincer Celik	osmanlicilegi	hello@dincercelik.com
Eduardo Gonzalez	egonzalez	dabarren@gmail.com
Jeffrey Zhang	Jeffrey4l	jeffrey.zhang@99cloud.net
Marcin Juszkiewicz	hrw	marcin.juszkiewicz@linaro.org
Mark Goddard	mgoddard	mark@stackhpc.com
Micha Nasiadka	mnasiadka	mnasiadka@gmail.com
Radosaw Piliszek	yoctozepto	radoslaw.piliszek@gmail.com
Surya Prakash	spsurya	singh.surya64mnnit@gmail.com
Cao Yuan	caoyuan	cao.yuan@99cloud.net

The current effective list is also available from Gerrit: https://review.opendev.org/#/admin/groups/1637,members

New Feature Planning

New features are discussed via IRC or mailing list (with [kolla] prefix). Kolla project keeps blueprints in Launchpad. Specs are welcome but not strictly required.

Task Tracking

Kolla project tracks tasks in Launchpad. Note this is the same place as for bugs.

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

A more lightweight task tracking is done via etherpad - Whiteboard.

Reporting a Bug

You found an issue and want to make sure we are aware of it? You can do so on Launchpad. Note this is the same place as for tasks.

Getting Your Patch Merged

Most changes proposed to Kolla Ansible require two +2 votes from core reviewers before +W. A release note is required on most changes as well. Release notes policy is described in *its own section*.

Significant changes should have documentation and testing provided with them.

Project Team Lead Duties

All common PTL duties are enumerated in the PTL guide. Kolla Ansible-specific PTL duties are listed in Kolla Ansible PTL guide.

7.1.2 Adding a new service

When adding a role for a new service in Ansible, there are couple of patterns which Kolla uses throughout and which should be followed.

- The sample inventories

  Entries should be added for the service in each of `ansible/inventory/multinode` and `ansible/inventory/all-in-one`.

- The playbook

  The main playbook that ties all roles together is in `ansible/site.yml`, this should be updated with appropriate roles, tags, and conditions. Ensure also that supporting hosts such as haproxy are updated when necessary.

- The common role

  A `common` role exists which sets up logging, `kolla-toolbox` and other supporting components. This should be included in all services within `meta/main.yml` of your role.

- Common tasks

  All services should include the following tasks:

  - `deploy.yml` : Used to bootstrap, configure and deploy containers for the service.
  - `reconfigure.yml` : Used to push new configuration files to the host and restart the service.
  - `pull.yml` : Used to pre fetch the image into the Docker image cache on hosts, to speed up initial deploys.
  - `upgrade.yml` : Used for upgrading the service in a rolling fashion. May include service specific setup and steps as not all services can be upgraded in the same way.

- Log rotation

  - For OpenStack services there should be a `cron-logrotate-PROJECT.conf.j2` template file in `ansible/roles/common/templates` with the following content:

    ```
    "/var/log/kolla/PROJECT/*.log"
    {
    }
    ```

  - For OpenStack services there should be an entry in the `services` list in the `cron.json.j2` template file in `ansible/roles/common/templates`.

- Log delivery

  - For OpenStack services the service should add a new `rewriterule` in the `match` element in the `01-rewrite.conf.j2` template file in `ansible/roles/common/templates/conf/filter` to deliver log messages to Elasticsearch.

- Documentation

  - For OpenStack services there should be an entry in the list `OpenStack services` in the `README.rst` file.
  - For infrastructure services there should be an entry in the list `Infrastructure components` in the `README.rst` file.

- Syntax
  - All YAML data files should start with three dashes (`---`).

Other than the above, most service roles abide by the following pattern:

- `Register`: Involves registering the service with Keystone, creating endpoints, roles, users, etc.
- `Config`: Distributes the config files to the nodes to be pulled into the container on startup.
- `Bootstrap`: Creating the database (but not tables), database user for the service, permissions, etc.
- `Bootstrap Service`: Starts a one shot container on the host to create the database tables, and other initial run time config.

Ansible handlers are used to create or restart containers when necessary.

7.1.3 Release notes

Introduction

Kolla Ansible (just like Kolla) uses the following release notes sections:

- `features` for new features or functionality; these should ideally refer to the blueprint being implemented;
- `fixes` for fixes closing bugs; these must refer to the bug being closed;
- `upgrade` for notes relevant when upgrading from previous version; these should ideally be added only between major versions; required when the proposed change affects behaviour in a non-backwards compatible way or generally changes something impactful;
- `deprecations` to track deprecated features; relevant changes may consist of only the commit message and the release note;
- `prelude` filled in by the PTL before each release or RC.

Other release note types may be applied per common sense. Each change should include a release note unless being a `TrivialFix` change or affecting only docs or CI. Such changes should *not* include a release note to avoid confusion. Remember release notes are mostly for end users which, in case of Kolla, are OpenStack administrators/operators. In case of doubt, the core team will let you know what is required.

To add a release note, run the following command:

```
tox -e venv -- reno new <summary-line-with-dashes>
```

All release notes can be inspected by browsing `releasenotes/notes` directory. Further on this page we show reno templates, examples and how to make use of them.

Note: The term *release note* is often abbreviated to *reno* as it is the name of the tool that is used to manage the release notes.

To generate renos in HTML format in `releasenotes/build`, run:

```
tox -e releasenotes
```

Note this requires the release note to be tracked by `git` so you have to at least add it to the `git`s staging area.

The release notes are linted in the CI system. To lint locally, run:

```
tox -e doc8
```

The above lints all of documentation at once.

Templates and examples

All approved release notes end up being published on a dedicated site:

https://docs.openstack.org/releasenotes/kolla-ansible/

When looking for examples, it is advised to consider browsing the page above for a similar type of change and then comparing with their source representation in `releasenotes/notes`.

The sections below give further guidelines. Please try to follow them but note they are not set in stone and sometimes a different wording might be more appropriate. In case of doubt, the core team will be happy to help.

Features

Template

```
---
features:
  - |
    Implements [some feature].
    [Can be described using multiple sentences if necessary.]
    [Limitations worth mentioning can be included as well.]
    `Blueprint [blueprint id] <https://blueprints.launchpad.net/kolla-
↪ansible/+spec/[blueprint id]>`__
```

Note: The blueprint can be mentioned even if the change implements it only partially. This can be emphasised by preceding the `Blueprint` word by `Partial`. See the example below.

Example

Implementing blueprint with id *letsencrypt-https*, we use `reno` to generate the scaffolded file:

```
tox -e venv -- reno new --from-template releasenotes/templates/feature.yml␣
↪blueprint-letsencrypt-https
```

Note: Since we dont require blueprints for simple features, it is allowed to make up a blueprint-id-friendly string (like in the example here) ad-hoc for the proposed feature. Please then skip the `blueprint-` prefix to avoid confusion.

And then fill it out with the following content:

```
---
features:
  - |
    Implements support for hassle-free integration with Let's Encrypt.
    The support is limited to operators in the underworld.
    For more details check the TLS docs of Kolla Ansible.
    `Partial Blueprint letsencrypt-https <https://blueprints.launchpad.net/
↪kolla-ansible/+spec/letsencrypt-https>`__
```

Note: The example above shows how to introduce a limitation. The limitation may be lifted in the same release cycle and it is OK to mention it nonetheless. Release notes can be edited later as long as they have not been shipped in an existing release or release candidate.

Fixes

Template

```
---
fixes:
  - |
    Fixes [some bug].
    [Can be described using multiple sentences if necessary.]
    [Possibly also giving the previous behaviour description.]
    `LP#[bug number] <https://launchpad.net/bugs/[bug number]>`__
```

Example

Fixing bug number *1889611*, we use `reno` to generate the scaffolded file:

```
tox -e venv -- reno new --from-template releasenotes/templates/fix.yml bug-
↪1889611
```

And then fill it out with the following content:

```
---
fixes:
  - |
    Fixes ``deploy-containers`` action missing for the Masakari role.
    `LP#1889611 <https://launchpad.net/bugs/1889611>`__
```

7.1.4 Development Environment with Vagrant

This guide describes how to use Vagrant to assist in developing for Kolla.

Vagrant is a tool for building and managing virtual machine environments in a single workflow. Vagrant takes care of setting up CentOS-based VMs for Kolla development, each with proper hardware like memory amount and number of network interfaces.

Getting Started

The Vagrant script implements **all-in-one** or **multi-node** deployments. **all-in-one** is the default.

In the case of **multi-node** deployment, the Vagrant setup builds a cluster with the following nodes by default:

- 3 control nodes
- 1 compute node
- 1 storage node (Note: ceph requires at least 3 storage nodes)
- 1 network node
- 1 operator node

The cluster node count can be changed by editing the Vagrantfile.

Kolla runs from the operator node to deploy OpenStack.

All nodes are connected with each other on the secondary NIC. The primary NIC is behind a NAT interface for connecting with the Internet. The third NIC is connected without IP configuration to a public bridge interface. This may be used for Neutron/Nova to connect to instances.

Start by downloading and installing the Vagrant package for the distro of choice. Various downloads can be found at the Vagrant downloads.

Install required dependencies as follows:

For CentOS or RHEL 8:

```
sudo dnf install ruby-devel libvirt-devel zlib-devel libpng-devel gcc \
qemu-kvm qemu-img libvirt python3-libvirt libvirt-client virt-install git
```

For Ubuntu 16.04 or later:

```
sudo apt install vagrant ruby-dev ruby-libvirt python-libvirt \
qemu-utils qemu-kvm libvirt-dev nfs-kernel-server zlib1g-dev libpng12-dev \
gcc git
```

Note: Many distros ship outdated versions of Vagrant by default. When in doubt, always install the latest from the downloads page above.

Next install the hostmanager plugin so all hosts are recorded in `/etc/hosts` (inside each vm):

```
vagrant plugin install vagrant-hostmanager
```

Vagrant supports a wide range of virtualization technologies. If VirtualBox is used, the vbguest plugin will be required to install the VirtualBox Guest Additions in the virtual machine:

```
vagrant plugin install vagrant-vbguest
```

This documentation focuses on libvirt specifics. To install vagrant-libvirt plugin:

```
vagrant plugin install --plugin-version ">= 0.0.31" vagrant-libvirt
```

Some Linux distributions offer vagrant-libvirt packages, but the version they provide tends to be too old to run Kolla. A version of >= 0.0.31 is required.

To use libvirt from Vagrant with a low privileges user without being asked for a password, add the user to the libvirt group:

```
sudo gpasswd -a ${USER} libvirt
newgrp libvirt
```

Note: In Ubuntu 16.04 and later, libvirtd group is used.

Setup NFS to permit file sharing between host and VMs. Contrary to the rsync method, NFS allows both way synchronization and offers much better performance than VirtualBox shared folders. For CentOS:

1. Add the virtual interfaces to the internal zone:

```
sudo firewall-cmd --zone=internal --add-interface=virbr0
sudo firewall-cmd --zone=internal --add-interface=virbr1
```

1. Enable nfs, rpc-bind and mountd services for firewalld:

```
sudo firewall-cmd --permanent --zone=internal --add-service=nfs
sudo firewall-cmd --permanent --zone=internal --add-service=rpc-bind
sudo firewall-cmd --permanent --zone=internal --add-service=mountd
sudo firewall-cmd --permanent --zone=internal --add-port=2049/udp
sudo firewall-cmd --permanent --add-port=2049/tcp
sudo firewall-cmd --permanent --add-port=111/udp
sudo firewall-cmd --permanent --add-port=111/tcp
sudo firewall-cmd --reload
```

Note: You may not have to do this because Ubuntu uses Uncomplicated Firewall (ufw) and ufw is disabled by default.

1. Start required services for NFS:

```
sudo systemctl restart firewalld
sudo systemctl start nfs-server
sudo systemctl start rpcbind.service
```

Ensure your system has libvirt and associated software installed and setup correctly. For CentOS:

```
sudo systemctl start libvirtd
sudo systemctl enable libvirtd
```

Find a location in the systems home directory and checkout Kolla repos:

```
git clone https://opendev.org/openstack/kolla-cli
git clone https://opendev.org/openstack/kolla-ansible
git clone https://opendev.org/openstack/kolla
```

All repos must share the same parent directory so the bootstrap code can locate them.

Developers can now tweak the Vagrantfile or bring up the default **all-in-one** CentOS 7-based environment:

```
cd kolla-ansible/contrib/dev/vagrant && vagrant up
```

The command `vagrant status` provides a quick overview of the VMs composing the environment.

Vagrant Up

Once Vagrant has completed deploying all nodes, the next step is to launch Kolla. First, connect with the **operator** node:

```
vagrant ssh operator
```

To speed things up, there is a local registry running on the operator. All nodes are configured so they can use this insecure repo to pull from, and use it as a mirror. Ansible may use this registry to pull images from.

All nodes have a local folder shared between the group and the hypervisor, and a folder shared between **all** nodes and the hypervisor. This mapping is lost after reboots, so make sure to use the command `vagrant reload <node>` when reboots are required. Having this shared folder provides a method to supply a different Docker binary to the cluster. The shared folder is also used to store the docker-registry files, so they are save from destructive operations like `vagrant destroy`.

Building images

Once logged on the **operator** VM call the `kolla-build` utility:

```
kolla-build
```

`kolla-build` accept arguments as documented in Building Container Images. It builds Docker images and pushes them to the local registry if the **push** option is enabled (in Vagrant this is the default behaviour).

Generating passwords

Before proceeding with the deployment you must generate the service passwords:

```
kolla-genpwd
```

Deploying OpenStack with Kolla

To deploy **all-in-one**:

```
sudo kolla-ansible deploy
```

To deploy **multinode**:

Ensure that the nodes deployed by Vagrant match those specified in the inventory file: `/usr/share/kolla-ansible/ansible/inventory/multinode`.

For Centos 7:

```
sudo kolla-ansible deploy -i /usr/share/kolla-ansible/ansible/inventory/
↪multinode
```

For Ubuntu 16.04 or later:

```
sudo kolla-ansible deploy -i /usr/local/share/kolla-ansible/ansible/
↪inventory/multinode
```

Validate OpenStack is operational:

```
kolla-ansible post-deploy
. /etc/kolla/admin-openrc.sh
openstack user list
```

Or navigate to `http://172.28.128.254/` with a web browser.

Further Reading

All Vagrant documentation can be found at Vagrant documentation.

7.1.5 Running tests

Kolla-ansible contains a suit of tests in the `tests` directory.

Any proposed code change in gerrit is automatically rejected by the Zuul CI system if the change causes test failures.

It is recommended for developers to run the test suite before submitting patch for review. This allows to catch errors as early as possible.

Preferred way to run the tests

The preferred way to run the unit tests is using `tox`. It executes tests in isolated environment, by creating separate virtualenv and installing dependencies from the `requirements.txt`, `test-requirements.txt` and `doc/requirements.txt` files, so the only package you install is `tox` itself:

```
pip install tox
```

For more information, see the unit testing section of the Testing wiki page. For example:

To run the default set of tests:

```
tox
```

To run the Python 3.8 tests:

```
tox -e py38
```

To run the style tests:

```
tox -e linters
```

To run multiple tests separate items by commas:

```
tox -e py38,linters
```

Running a subset of tests

Instead of running all tests, you can specify an individual directory, file, class or method that contains test code, i.e. filter full names of tests by a string.

To run the tests located only in the `kolla-ansible/tests` directory use:

```
tox -e py38 kolla-ansible.tests
```

To run the tests of a specific file `kolla-ansible/tests/test_kolla_docker.py`:

```
tox -e py38 test_kolla_docker
```

To run the tests in the `ModuleArgsTest` class in the `kolla-ansible/tests/test_kolla_docker.py` file:

```
tox -e py38 test_kolla_docker.ModuleArgsTest
```

To run the `ModuleArgsTest.test_module_args` test method in the `kolla-ansible/tests/test_kolla_docker.py` file:

```
tox -e py38 test_kolla_docker.ModuleArgsTest.test_module_args
```

Debugging unit tests

In order to break into the debugger from a unit test we need to insert a breaking point to the code:

```
import pdb; pdb.set_trace()
```

Then run `tox` with the debug environment as one of the following:

```
tox -e debug
tox -e debug test_file_name.TestClass.test_name
```

For more information, see the oslotest documentation.

7.1.6 Using Kolla For OpenStack Development

Kolla-ansible can be used to deploy containers in a way suitable for doing development on OpenStack services.

Note: This functionality is new in the Pike release.

Heat was the first service to be supported, and so the following will use submitting a patch to Heat using Kolla as an example.

Only source containers are supported.

Warning: Kolla dev mode is intended for OpenStack hacking or development only. Do not use this in production!

Enabling Kolla dev mode

To enable dev mode for all supported services, set in `/etc/kolla/globals.yml`:

```
kolla_dev_mode: true
```

To enable it just for heat, set:

```
heat_dev_mode: true
```

Usage

When enabled, the source repo for the service in question will be cloned under `/opt/stack/` on the target node(s). This will be bind mounted into the containers virtualenv under the location expected by the service on startup.

After making code changes, simply restart the container to pick them up:

```
docker restart heat_api
```

Debugging

`remote_pdb` can be used to perform debugging with Kolla containers. First, make sure it is installed in the container in question:

```
docker exec -it -u root heat_api pip install remote_pdb
```

Then, set your breakpoint as follows:

```
from remote_pdb import RemotePdb
RemotePdb('127.0.0.1', 4444).set_trace()
```

Once you run the code(restart the container), pdb can be accessed using `socat`:

```
socat readline tcp:127.0.0.1:4444
```

Learn more information about remote_pdb.

7.1.7 Bug triage

The triage of Kolla bugs follows the OpenStack-wide process documented on BugTriage in the wiki. Please reference Bugs for further details.

7.1.8 PTL Guide

The Kolla PTL is also PTL for Kolla Ansible. See the Kolla PTL guide.

7.1.9 Release Management

Release management for Kolla Ansible is very much linked to that of Kolla. See Kolla release management.

7.1.10 Test Identity Provider setup

This guide shows how to create an Identity Provider that handles the OpenID Connect protocol to authenticate users when using Federation with OpenStack (these configurations must not be used in a production environment).

Keycloak

Keycloak is a Java application that implements an Identity Provider handling both OpenID Connect and SAML protocols.

To setup a Keycloak instance for testing is pretty simple with Docker.

Creating the Docker Keycloak instance

Run the docker command:

```
docker run -p 8080:8080 -p 8443:8443 -e KEYCLOAK_USER=admin -e KEYCLOAK_
↪PASSWORD=admin quay.io/keycloak/keycloak:latest
```

This will create a Keycloak instance that has the admin credentials as admin/admin and is listening on port 8080.

After creating the instance, you will need to log in to the Keycloak as administrator and setup the first Identity Provider.

Creating an Identity Provider with Keycloak

The following guide assumes that the steps are executed from the same machine (localhost), but you can change the hostname if you want to run it from elsewhere.

In this guide, we will use the new_realm as the realm name in Keycloak, so, if you want to use any other realm name, you must to change new_realm in the URIs used in the guide and replace the new_realm with the realm name that you are using.

- Access the admin console on http://localhost:8080/auth/ in the Administration Console option.
- Authenticate using the credentials defined in the creation step.
- Create a new realm in the http://localhost:8080/auth/admin/master/console/#/create/realm page.
- After creating a realm, you will need to create a client to be used by Keystone; to do it, just access http://localhost:8080/auth/admin/master/console/#/create/client/new_realm.
- To create a client, you will need to set the client_id (just choose anyone), the protocol (must be openid-connect) and the Root Url (you can leave it blank)
- After creating the client, you will need to update some clients attributes like:
  - Enable the Implicit flow (this one allows you to use the OpenStack CLI with oidcv3 plugin)
  - Set Access Type to confidential
  - Add the Horizon and Keystone URIs to the Valid Redirect URIs. Keystone should be within the /redirect_uri path, for example: https://horizon.com/ and https://keystone.com/redirect_uri
  - Save the changes
  - Access the clients Mappers tab to add the users attributes that will be shared with the client (Keystone):
    - In this guide, we will need the following attribute mappers in Keycloak:

name/user attribute/token claim name	mapper type
openstack-user-domain	user attribute
openstack-default-project	user attribute

- After creating the client, you will need to create a user in that realm to log in OpenStack via identity federation

- To create a user, access http://localhost:8080/auth/admin/master/console/#/create/user/new_realm and fill the form with the users data
- After creating the user, you can access the tab Credentials to set the users password
- Then, in the tab Attributes, you must set the authorization attributes to be used by Keystone, these attributes are defined in the *attribute mapping* in Keystone

After you create the Identity provider, you will need to get some data from the Identity Provider to configure in Kolla-Ansible

Configuring Kolla Ansible to use the Identity Provider

This section is about how one can get the data needed in *Setup OIDC via Kolla Ansible*.

- name: The realm name, in this case it will be new_realm
- identifier: http://localhost:8080/auth/realms/new_realm/ (again, the new_realm is the name of the realm)
- certificate_file: This one can be downloaded from http://localhost:8080/auth/admin/master/console/#/realms/new_realm/keys
- metadata_folder:
  - localhost%3A8080%2Fauth%2Frealms%2Fnew_realm.client:
    - client_id: Access http://localhost:8080/auth/admin/master/console/#/realms/new_realm/clients , and access the client you created for Keystone, copy the Client ID displayed in the page
    - client_secret: In the same page you got the client_id, access the tab Credentials and copy the secret value
  - localhost%3A8080%2Fauth%2Frealms%2Fnew_realm.provider: Copy the json from http://localhost:8080/auth/realms/new_realm/.well-known/openid-configuration (the new_realm is the realm name)
  - localhost%3A8080%2Fauth%2Frealms%2Fnew_realm.conf: You can leave this file as an empty json {}

After you finished the configuration of the Identity Provider, your main configuration should look something like the following:

```
keystone_identity_providers:
  - name: "new_realm"
    openstack_domain: "new_domain"
    protocol: "openid"
    identifier: "http://localhost:8080/auth/realms/new_realm"
    public_name: "Authenticate via new_realm"
    attribute_mapping: "attribute_mapping_keycloak_new_realm"
    metadata_folder: "/root/inDev/meta-idp"
    certificate_file: "/root/inDev/certs/
↪LRVweuT51StjMdsna59jKfB3xw0r8Iz1d1J1HeAbmlw.pem"
keystone_identity_mappings:
  - name: "attribute_mapping_keycloak_new_realm"
    file: "/root/inDev/attr_map/attribute_mapping.json"
```

Then, after deploying OpenStack, you should be able to log in Horizon using the Authenticate using -> Authenticate via new_realm, and writing new_realm.com in the E-mail or domain name field. After that, you will be redirected to a new page to choose the Identity Provider in Keystone. Just click in the link localhost:8080/auth/realms/new_realm; this will redirect you to Keycloak (idP) where you will need to log in with the user that you created. If the users attributes in Keycloak are ok, the user will be created in OpenStack and you will be able to log in Horizon.

Attribute mapping

This section shows how to create the attribute mapping to map an Identity Provider user to a Keystone user (ephemeral).

The OIDC- prefix in the remote types is defined in the OIDCClaimPrefix configuration in the wsgi-keystone.conf file; this prefix must be in the attribute mapping as the mod-oidc-wsgi is adding the prefix in the users attributes before sending it to Keystone. The attribute openstack-user-domain will define the users domain in OpenStack and the attribute openstack-default-project will define the users project in the OpenStack (the user will be assigned with the role member in the project)

```
[
    {
        "local": [
            {
                "user": {
                    "name": "{0}",
                    "email": "{1}",
                    "domain": {
                        "name": "{2}"
                    }
                },
                "domain": {
                        "name": "{2}"
                    },
                "projects": [
                    {
                        "name": "{3}",
                        "roles": [
                            {
                                "name": "member"
                            }
                        ]
                    }
                ]
            }
        ],
        "remote": [
            {
                "type": "OIDC-preferred_username"
            },
            {
                "type": "OIDC-email"
            },
            {
                "type": "OIDC-openstack-user-domain"
            },
            {
```

(continues on next page)

(continued from previous page)

```
                "type": "OIDC-openstack-default-project"
            }
        ]
    }
]
```