magnum Documentation

Release 14.1.2.dev1

OpenStack Foundation

Feb 02, 2024

CONTENTS

Magnum is an OpenStack project which offers container orchestration engines for deploying and managing containers as first class resources in OpenStack.

- **Free software:** under the Apache license
- **Source:** https://opendev.org/openstack/magnum
- **Blueprints:** https://blueprints.launchpad.net/magnum
- **Bugs:** https://bugs.launchpad.net/magnum
- **REST Client:** https://opendev.org/openstack/python-magnumclient

CHAPTER

ONE

ARCHITECTURE

There are several different types of objects in the magnum system:

- **Cluster:** A collection of node objects where work is scheduled
- **ClusterTemplate:** An object stores template information about the cluster which is used to create new clusters consistently

Two binaries work together to compose the magnum system. The first binary (accessed by the python-magnumclient code) is the magnum-api REST server. The REST server may run as one process or multiple processes. When a REST request is sent to the client API, the request is sent via AMQP to the magnum-conductor process. The REST server is horizontally scalable. At this time, the conductor is limited to one process, but we intend to add horizontal scalability to the conductor as well.

CHAPTER

TWO

FEATURES

- Abstractions for Clusters
- Integration with Kubernetes, Swarm, Mesos for backend container technology
- Integration with Keystone for multi-tenant security
- Integration with Neutron for Kubernetes multi-tenancy network security
- Integration with Cinder to provide volume service for containers

CHAPTER

THREE

INSTALLATION GUIDE

3.1 Magnum Installation Guide

3.1.1 Container Infrastructure Management service overview

The Container Infrastructure Management service consists of the following components:

`magnum` command-line client A CLI that communicates with the `magnum-api` to create and manage container clusters. End developers can directly use the magnum REST API.

`magnum-api` service An OpenStack-native REST API that processes API requests by sending them to the `magnum-conductor` via AMQP.

`magnum-conductor` service Runs on a controller machine and connects to heat to orchestrate a cluster. Additionally, it connects to a Docker Swarm, Kubernetes or Mesos REST API endpoint.

3.1.2 Install and configure

This section describes how to install and configure the Container Infrastructure Management service, code-named magnum, on the controller node.

This section assumes that you already have a working OpenStack environment with at least the following components installed: Identity service, Image service, Compute service, Networking service, Block Storage service and Orchestration service. See OpenStack Install Guides.

To provide access to Docker Swarm or Kubernetes using the native clients (docker or kubectl, respectively) magnum uses TLS certificates. To store the certificates, it is recommended to use the Key Manager service, code-named barbican, or you can save them in magnums database.

Optionally, you can install the following components:

- Load Balancer as a Service (LBaaS v2) to create clusters with multiple masters
- Bare Metal service to create baremetal clusters
- Object Storage service to make private Docker registries available to users
- Telemetry Data Collection service to periodically send magnum-related metrics

Note: Installation and configuration vary by distribution.

Important: Magnum creates clusters of compute instances on the Compute service (nova). These instances must have basic Internet connectivity and must be able to reach magnums API server. Make sure that the Compute and Network services are configured accordingly.

Install and configure for Debian

This section describes how to install and configure the Container Infrastructure Management service for Debian.

Prerequisites

Before you install and configure the Container Infrastructure Management service, you must create a database, service credentials, and API endpoints.

1. To create the database, complete these steps:
   - Use the database access client to connect to the database server as the `root` user:

     ```
     # mysql
     ```

   - Create the `magnum` database:

     ```
     CREATE DATABASE magnum;
     ```

   - Grant proper access to the `magnum` database:

     ```
     GRANT ALL PRIVILEGES ON magnum.* TO 'magnum'@'localhost' \
       IDENTIFIED BY 'MAGNUM_DBPASS';
     GRANT ALL PRIVILEGES ON magnum.* TO 'magnum'@'%' \
       IDENTIFIED BY 'MAGNUM_DBPASS';
     ```

     Replace `MAGNUM_DBPASS` with a suitable password.

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

   ```
   $ . admin-openrc
   ```

3. To create the service credentials, complete these steps:
   - Create the `magnum` user:

     ```
     $ openstack user create --domain default \
       --password-prompt magnum
     User Password:
     Repeat User Password:
     +-----------+----------------------------------+
     | Field     | Value                            |
     +-----------+----------------------------------+
     | domain_id | default                          |
     ```

(continues on next page)

(continued from previous page)

```
| enabled    | True                             |
| id         | a8ebafc275c54d389dfc1bff8b4fe286 |
| name       | magnum                           |
+------------+----------------------------------+
```

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

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

Note: This command provides no output.

- Create the `magnum` service entity:

```
$ openstack service create --name magnum \
  --description "OpenStack Container Infrastructure Management␣
↪Service" \
  container-infra
+-------------+----------------------------------------------------------
↪-+
| Field       | Value                                                    ␣
↪ |
+-------------+----------------------------------------------------------
↪-+
| description | OpenStack Container Infrastructure Management␣
↪Service |
| enabled     | True                                                     ␣
↪ |
| id          | 194faf83e8fd4e028e5ff75d3d8d0df2                         ␣
↪ |
| name        | magnum                                                   ␣
↪ |
| type        | container-infra                                          ␣
↪ |
+-------------+----------------------------------------------------------
↪-+
```

4. Create the Container Infrastructure Management service API endpoints:

```
$ openstack endpoint create --region RegionOne \
  container-infra public http://CONTROLLER_IP:9511/v1
+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | cb137e6366ad495bb521cfe92d8b8858 |
| interface    | public                           |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | 0f7f62a1f1a247d2a4cb237642814d0e |
```

(continues on next page)

(continued from previous page)

```
| service_name | magnum                           |
| service_type | container-infra                  |
| url          | http://CONTROLLER_IP:9511/v1     |
+--------------+----------------------------------+

$ openstack endpoint create --region RegionOne \
  container-infra internal http://CONTROLLER_IP:9511/v1
+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | 17cbc3b6f51449a0a818118d6d62868d |
| interface    | internal                         |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | 0f7f62a1f1a247d2a4cb237642814d0e |
| service_name | magnum                           |
| service_type | container-infra                  |
| url          | http://CONTROLLER_IP:9511/v1     |
+--------------+----------------------------------+

$ openstack endpoint create --region RegionOne \
  container-infra admin http://CONTROLLER_IP:9511/v1
+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | 30f8888e6b6646d7b5cd14354c95a684 |
| interface    | admin                            |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | 0f7f62a1f1a247d2a4cb237642814d0e |
| service_name | magnum                           |
| service_type | container-infra                  |
| url          | http://CONTROLLER_IP:9511/v1     |
+--------------+----------------------------------+
```

Replace `CONTROLLER_IP` with the IP magnum listens to. Alternatively, you can use a hostname which is reachable by the Compute instances.

5. Magnum requires additional information in the Identity service to manage COE clusters. To add this information, complete these steps:

   - Create the `magnum` domain that contains projects and users:

```
$ openstack domain create --description "Owns users and projects \
  created by magnum" magnum
  +-------------+----------------------------------------------+
  | Field       | Value                                        |
  +-------------+----------------------------------------------+
```

(continues on next page)

(continued from previous page)

```
| description | Owns users and projects created by magnum |
| enabled     | True                                      |
| id          | 66e0469de9c04eda9bc368e001676d20          |
| name        | magnum                                    |
+-------------+-------------------------------------------+
```

- Create the `magnum_domain_admin` user to manage projects and users in the `magnum` domain:

```
$ openstack user create --domain magnum --password-prompt \
  magnum_domain_admin
  User Password:
  Repeat User Password:
  +-----------+----------------------------------+
  | Field     | Value                            |
  +-----------+----------------------------------+
  | domain_id | 66e0469de9c04eda9bc368e001676d20 |
  | enabled   | True                             |
  | id        | 529b81cf35094beb9784c6d06c090c2b |
  | name      | magnum_domain_admin              |
  +-----------+----------------------------------+
```

- Add the `admin` role to the `magnum_domain_admin` user in the `magnum` domain to enable administrative management privileges by the `magnum_domain_admin` user:

```
$ openstack role add --domain magnum --user-domain magnum --user \
  magnum_domain_admin admin
```

Note: This command provides no output.

Install and configure components

1. Install the common and library packages:

```
# DEBIAN_FRONTEND=noninteractive apt-get install magnum-api magnum-
↪conductor
```

2. Edit the `/etc/magnum/magnum.conf` file:

   - In the `[api]` section, configure the host:

```
[api]
...
host = CONTROLLER_IP
```

     Replace `CONTROLLER_IP` with the IP address on which you wish magnum api should listen.

   - In the `[certificates]` section, select `barbican` (or `x509keypair` if you dont have barbican installed):

     – Use barbican to store certificates:

```
[certificates]
...
cert_manager_type = barbican
```

Important: Barbican is recommended for production environments.

- To store x509 certificates in magnums database:

```
[certificates]
...
cert_manager_type = x509keypair
```

- In the `[cinder_client]` section, configure the region name:

```
[cinder_client]
...
region_name = RegionOne
```

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

```
[database]
...
connection = mysql+pymysql://magnum:MAGNUM_DBPASS@controller/magnum
```

Replace `MAGNUM_DBPASS` with the password you chose for the magnum database.

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

```
[keystone_authtoken]
...
memcached_servers = controller:11211
auth_version = v3
www_authenticate_uri = http://controller:5000/v3
project_domain_id = default
project_name = service
user_domain_id = default
password = MAGNUM_PASS
username = magnum
auth_url = http://controller:5000
auth_type = password
admin_user = magnum
admin_password = MAGNUM_PASS
admin_tenant_name = service

[trust]
...
trustee_domain_name = magnum
trustee_domain_admin_name = magnum_domain_admin
trustee_domain_admin_password = DOMAIN_ADMIN_PASS
trustee_keystone_interface = KEYSTONE_INTERFACE
```

Replace MAGNUM_PASS with the password you chose for the magnum user in the Identity service and DOMAIN_ADMIN_PASS with the password you chose for the magnum_domain_admin user.

Replace KEYSTONE_INTERFACE with either public or internal depending on your network configuration. If your instances cannot reach internal keystone endpoint which is often the case in production environments it should be set to public. Default to public

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

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

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

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

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

3. Populate Magnum database:

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

Finalize installation

- Restart the Container Infrastructure Management services:

```
# service magnum-api restart
# service magnum-conductor restart
```

Install and configure for openSUSE and SUSE Linux Enterprise

This section describes how to install and configure the Container Infrastructure Management service for openSUSE Leap 42.2 and SUSE Linux Enterprise Server 12 SP2.

Prerequisites

Before you install and configure the Container Infrastructure Management service, you must create a database, service credentials, and API endpoints.

1. To create the database, complete these steps:

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

   ```
   # mysql
   ```

   - Create the magnum database:

```
CREATE DATABASE magnum;
```

- Grant proper access to the `magnum` database:

```
GRANT ALL PRIVILEGES ON magnum.* TO 'magnum'@'localhost' \
  IDENTIFIED BY 'MAGNUM_DBPASS';
GRANT ALL PRIVILEGES ON magnum.* TO 'magnum'@'%' \
  IDENTIFIED BY 'MAGNUM_DBPASS';
```

  Replace `MAGNUM_DBPASS` with a suitable password.

- Exit the database access client.

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

```
$ . admin-openrc
```

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

   - Create the `magnum` user:

```
$ openstack user create --domain default \
  --password-prompt magnum
User Password:
Repeat User Password:
+-----------+----------------------------------+
| Field     | Value                            |
+-----------+----------------------------------+
| domain_id | default                          |
| enabled   | True                             |
| id        | a8ebafc275c54d389dfc1bff8b4fe286 |
| name      | magnum                           |
+-----------+----------------------------------+
```

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

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

Note: This command provides no output.

   - Create the `magnum` service entity:

```
$ openstack service create --name magnum \
  --description "OpenStack Container Infrastructure Management␣
↪Service" \
  container-infra
+-------------+----------------------------------------------------------
↪-+
| Field       | Value                                                    ␣
↪ |
+-------------+----------------------------------------------------------
↪-+
```

(continues on next page)

(continued from previous page)

```
| description | OpenStack Container Infrastructure Management␣
↪Service |
| enabled     | True                                                ␣
↪ |
| id          | 194faf83e8fd4e028e5ff75d3d8d0df2                    ␣
↪ |
| name        | magnum                                              ␣
↪ |
| type        | container-infra                                     ␣
↪ |
+-------------+-----------------------------------------------------
↪-+
```

4. Create the Container Infrastructure Management service API endpoints:

```
$ openstack endpoint create --region RegionOne \
  container-infra public http://CONTROLLER_IP:9511/v1
+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | cb137e6366ad495bb521cfe92d8b8858 |
| interface    | public                           |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | 0f7f62a1f1a247d2a4cb237642814d0e |
| service_name | magnum                           |
| service_type | container-infra                  |
| url          | http://CONTROLLER_IP:9511/v1     |
+--------------+----------------------------------+

$ openstack endpoint create --region RegionOne \
  container-infra internal http://CONTROLLER_IP:9511/v1
+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | 17cbc3b6f51449a0a818118d6d62868d |
| interface    | internal                         |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | 0f7f62a1f1a247d2a4cb237642814d0e |
| service_name | magnum                           |
| service_type | container-infra                  |
| url          | http://CONTROLLER_IP:9511/v1     |
+--------------+----------------------------------+

$ openstack endpoint create --region RegionOne \
  container-infra admin http://CONTROLLER_IP:9511/v1
```

(continues on next page)

(continued from previous page)

```
+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | 30f8888e6b6646d7b5cd14354c95a684 |
| interface    | admin                            |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | 0f7f62a1f1a247d2a4cb237642814d0e |
| service_name | magnum                           |
| service_type | container-infra                  |
| url          | http://CONTROLLER_IP:9511/v1     |
+--------------+----------------------------------+
```

Replace `CONTROLLER_IP` with the IP magnum listens to. Alternatively, you can use a hostname which is reachable by the Compute instances.

5. Magnum requires additional information in the Identity service to manage COE clusters. To add this information, complete these steps:

   - Create the `magnum` domain that contains projects and users:

```
$ openstack domain create --description "Owns users and projects \
  created by magnum" magnum
  +-------------+-------------------------------------------+
  | Field       | Value                                     |
  +-------------+-------------------------------------------+
  | description | Owns users and projects created by magnum |
  | enabled     | True                                      |
  | id          | 66e0469de9c04eda9bc368e001676d20          |
  | name        | magnum                                    |
  +-------------+-------------------------------------------+
```

   - Create the `magnum_domain_admin` user to manage projects and users in the `magnum` domain:

```
$ openstack user create --domain magnum --password-prompt \
  magnum_domain_admin
  User Password:
  Repeat User Password:
  +-----------+----------------------------------+
  | Field     | Value                            |
  +-----------+----------------------------------+
  | domain_id | 66e0469de9c04eda9bc368e001676d20 |
  | enabled   | True                             |
  | id        | 529b81cf35094beb9784c6d06c090c2b |
  | name      | magnum_domain_admin              |
  +-----------+----------------------------------+
```

   - Add the `admin` role to the `magnum_domain_admin` user in the `magnum` domain to enable administrative management privileges by the `magnum_domain_admin` user:

```
$ openstack role add --domain magnum --user-domain magnum --user \
  magnum_domain_admin admin
```

Note: This command provides no output.

Install and configure components

1. Install the packages:

```
# zypper install openstack-magnum-api openstack-magnum-conductor python-
↪magnumclient
```

2. Edit the `/etc/magnum/magnum.conf` file:

   - In the `[api]` section, configure the host:

```
[api]
...
host = CONTROLLER_IP
```

     Replace `CONTROLLER_IP` with the IP address on which you wish magnum api should listen.

   - In the `[certificates]` section, select `barbican` (or `x509keypair` if you dont have barbican installed):

     – Use barbican to store certificates:

```
[certificates]
...
cert_manager_type = barbican
```

Important: Barbican is recommended for production environments.

     – To store x509 certificates in magnums database:

```
[certificates]
...
cert_manager_type = x509keypair
```

   - In the `[cinder_client]` section, configure the region name:

```
[cinder_client]
...
region_name = RegionOne
```

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

```
[database]
...
connection = mysql+pymysql://magnum:MAGNUM_DBPASS@controller/magnum
```

Replace MAGNUM_DBPASS with the password you chose for the magnum database.

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

```
[keystone_authtoken]
...
memcached_servers = controller:11211
auth_version = v3
www_authenticate_uri = http://controller:5000/v3
project_domain_id = default
project_name = service
user_domain_id = default
password = MAGNUM_PASS
username = magnum
auth_url = http://controller:5000
auth_type = password
admin_user = magnum
admin_password = MAGNUM_PASS
admin_tenant_name = service

[trust]
...
trustee_domain_name = magnum
trustee_domain_admin_name = magnum_domain_admin
trustee_domain_admin_password = DOMAIN_ADMIN_PASS
trustee_keystone_interface = KEYSTONE_INTERFACE
```

Replace MAGNUM_PASS with the password you chose for the magnum user in the Identity service and DOMAIN_ADMIN_PASS with the password you chose for the magnum_domain_admin user.

Replace KEYSTONE_INTERFACE with either public or internal depending on your network configuration. If your instances cannot reach internal keystone endpoint which is often the case in production environments it should be set to public. Default to public

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

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

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

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

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

3. Populate Magnum database:

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

Finalize installation

- Start the Container Infrastructure Management services and configure them to start when the system boots:

```
# systemctl enable openstack-magnum-api.service \
  openstack-magnum-conductor.service
# systemctl start openstack-magnum-api.service \
  openstack-magnum-conductor.service
```

Install and configure for Red Hat Enterprise Linux and CentOS

This section describes how to install and configure the Container Infrastructure Management service for Red Hat Enterprise Linux 7 and CentOS 7.

Prerequisites

Before you install and configure the Container Infrastructure Management service, you must create a database, service credentials, and API endpoints.

1. To create the database, complete these steps:
   - Use the database access client to connect to the database server as the `root` user:

     ```
     # mysql
     ```

   - Create the `magnum` database:

     ```
     CREATE DATABASE magnum;
     ```

   - Grant proper access to the `magnum` database:

     ```
     GRANT ALL PRIVILEGES ON magnum.* TO 'magnum'@'localhost' \
       IDENTIFIED BY 'MAGNUM_DBPASS';
     GRANT ALL PRIVILEGES ON magnum.* TO 'magnum'@'%' \
       IDENTIFIED BY 'MAGNUM_DBPASS';
     ```

     Replace `MAGNUM_DBPASS` with a suitable password.

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

   ```
   $ . admin-openrc
   ```

3. To create the service credentials, complete these steps:
   - Create the `magnum` user:

```
$ openstack user create --domain default \
  --password-prompt magnum
User Password:
Repeat User Password:
+-----------+----------------------------------+
| Field     | Value                            |
+-----------+----------------------------------+
| domain_id | default                          |
| enabled   | True                             |
| id        | a8ebafc275c54d389dfc1bff8b4fe286 |
| name      | magnum                           |
+-----------+----------------------------------+
```

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

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

Note: This command provides no output.

- Create the `magnum` service entity:

```
$ openstack service create --name magnum \
  --description "OpenStack Container Infrastructure Management␣
↪Service" \
  container-infra
+-------------+----------------------------------------------------------
↪-+
| Field       | Value                                                    ␣
↪ |
+-------------+----------------------------------------------------------
↪-+
| description | OpenStack Container Infrastructure Management␣
↪Service |
| enabled     | True                                                     ␣
↪ |
| id          | 194faf83e8fd4e028e5ff75d3d8d0df2                         ␣
↪ |
| name        | magnum                                                   ␣
↪ |
| type        | container-infra                                          ␣
↪ |
+-------------+----------------------------------------------------------
↪-+
```

4. Create the Container Infrastructure Management service API endpoints:

```
$ openstack endpoint create --region RegionOne \
  container-infra public http://CONTROLLER_IP:9511/v1
+--------------+----------------------------------+
```

(continues on next page)

(continued from previous page)

```
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | cb137e6366ad495bb521cfe92d8b8858 |
| interface    | public                           |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | 0f7f62a1f1a247d2a4cb237642814d0e |
| service_name | magnum                           |
| service_type | container-infra                  |
| url          | http://CONTROLLER_IP:9511/v1     |
+--------------+----------------------------------+

$ openstack endpoint create --region RegionOne \
  container-infra internal http://CONTROLLER_IP:9511/v1
+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | 17cbc3b6f51449a0a818118d6d62868d |
| interface    | internal                         |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | 0f7f62a1f1a247d2a4cb237642814d0e |
| service_name | magnum                           |
| service_type | container-infra                  |
| url          | http://CONTROLLER_IP:9511/v1     |
+--------------+----------------------------------+

$ openstack endpoint create --region RegionOne \
  container-infra admin http://CONTROLLER_IP:9511/v1
+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | 30f8888e6b6646d7b5cd14354c95a684 |
| interface    | admin                            |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | 0f7f62a1f1a247d2a4cb237642814d0e |
| service_name | magnum                           |
| service_type | container-infra                  |
| url          | http://CONTROLLER_IP:9511/v1     |
+--------------+----------------------------------+
```

Replace `CONTROLLER_IP` with the IP magnum listens to. Alternatively, you can use a hostname which is reachable by the Compute instances.

5. Magnum requires additional information in the Identity service to manage COE clusters. To add this information, complete these steps:

- Create the magnum domain that contains projects and users:

```
$ openstack domain create --description "Owns users and projects \
  created by magnum" magnum
  +-------------+-------------------------------------------+
  | Field       | Value                                     |
  +-------------+-------------------------------------------+
  | description | Owns users and projects created by magnum |
  | enabled     | True                                      |
  | id          | 66e0469de9c04eda9bc368e001676d20          |
  | name        | magnum                                    |
  +-------------+-------------------------------------------+
```

- Create the magnum_domain_admin user to manage projects and users in the magnum domain:

```
$ openstack user create --domain magnum --password-prompt \
  magnum_domain_admin
  User Password:
  Repeat User Password:
  +-----------+----------------------------------+
  | Field     | Value                            |
  +-----------+----------------------------------+
  | domain_id | 66e0469de9c04eda9bc368e001676d20 |
  | enabled   | True                             |
  | id        | 529b81cf35094beb9784c6d06c090c2b |
  | name      | magnum_domain_admin              |
  +-----------+----------------------------------+
```

- Add the admin role to the magnum_domain_admin user in the magnum domain to enable administrative management privileges by the magnum_domain_admin user:

```
$ openstack role add --domain magnum --user-domain magnum --user \
  magnum_domain_admin admin
```

Note: This command provides no output.

Install and configure components

1. Install the packages:

```
# yum install openstack-magnum-api openstack-magnum-conductor python-
↪magnumclient
```

2. Edit the /etc/magnum/magnum.conf file:

- In the [api] section, configure the host:

```
[api]
...
host = CONTROLLER_IP
```

Replace CONTROLLER_IP with the IP address on which you wish magnum api should listen.

- In the [certificates] section, select barbican (or x509keypair if you dont have barbican installed):
  - Use barbican to store certificates:

```
[certificates]
...
cert_manager_type = barbican
```

Important: Barbican is recommended for production environments.

  - To store x509 certificates in magnums database:

```
[certificates]
...
cert_manager_type = x509keypair
```

- In the [cinder_client] section, configure the region name:

```
[cinder_client]
...
region_name = RegionOne
```

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

```
[database]
...
connection = mysql+pymysql://magnum:MAGNUM_DBPASS@controller/magnum
```

Replace MAGNUM_DBPASS with the password you chose for the magnum database.

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

```
[keystone_authtoken]
...
memcached_servers = controller:11211
auth_version = v3
www_authenticate_uri = http://controller:5000/v3
project_domain_id = default
project_name = service
user_domain_id = default
password = MAGNUM_PASS
username = magnum
auth_url = http://controller:5000
auth_type = password
admin_user = magnum
admin_password = MAGNUM_PASS
admin_tenant_name = service

[trust]
```

(continues on next page)

(continued from previous page)

```
...
trustee_domain_name = magnum
trustee_domain_admin_name = magnum_domain_admin
trustee_domain_admin_password = DOMAIN_ADMIN_PASS
trustee_keystone_interface = KEYSTONE_INTERFACE
```

Replace MAGNUM_PASS with the password you chose for the magnum user in the Identity service and DOMAIN_ADMIN_PASS with the password you chose for the `magnum_domain_admin` user.

Replace KEYSTONE_INTERFACE with either `public` or `internal` depending on your network configuration. If your instances cannot reach internal keystone endpoint which is often the case in production environments it should be set to `public`. Default to `public`

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

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

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

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

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

- Additionally, edit the `/etc/magnum/magnum.conf` file:
  - In the `[oslo_concurrency]` section, configure the `lock_path`:

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

3. Populate Magnum database:

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

Finalize installation

- Start the Container Infrastructure Management services and configure them to start when the system boots:

```
# systemctl enable openstack-magnum-api.service \
  openstack-magnum-conductor.service
# systemctl start openstack-magnum-api.service \
  openstack-magnum-conductor.service
```

Install and configure for Ubuntu

This section describes how to install and configure the Container Infrastructure Management service for Ubuntu 14.04 (LTS).

Prerequisites

Before you install and configure the Container Infrastructure Management service, you must create a database, service credentials, and API endpoints.

1. To create the database, complete these steps:
   - Use the database access client to connect to the database server as the `root` user:

     ```
     # mysql
     ```

   - Create the `magnum` database:

     ```
     CREATE DATABASE magnum;
     ```

   - Grant proper access to the `magnum` database:

     ```
     GRANT ALL PRIVILEGES ON magnum.* TO 'magnum'@'localhost' \
       IDENTIFIED BY 'MAGNUM_DBPASS';
     GRANT ALL PRIVILEGES ON magnum.* TO 'magnum'@'%' \
       IDENTIFIED BY 'MAGNUM_DBPASS';
     ```

     Replace `MAGNUM_DBPASS` with a suitable password.

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

   ```
   $ . admin-openrc
   ```

3. To create the service credentials, complete these steps:
   - Create the `magnum` user:

     ```
     $ openstack user create --domain default \
       --password-prompt magnum
     User Password:
     Repeat User Password:
     +-----------+----------------------------------+
     | Field     | Value                            |
     +-----------+----------------------------------+
     | domain_id | default                          |
     | enabled   | True                             |
     | id        | a8ebafc275c54d389dfc1bff8b4fe286 |
     | name      | magnum                           |
     +-----------+----------------------------------+
     ```

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

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

Note: This command provides no output.

- Create the magnum service entity:

```
$ openstack service create --name magnum \
  --description "OpenStack Container Infrastructure Management␣
↪Service" \
  container-infra
+-------------+----------------------------------------------------------
↪-+
| Field       | Value                                                    ␣
↪ |
+-------------+----------------------------------------------------------
↪-+
| description | OpenStack Container Infrastructure Management␣
↪Service |
| enabled     | True                                                     ␣
↪ |
| id          | 194faf83e8fd4e028e5ff75d3d8d0df2                         ␣
↪ |
| name        | magnum                                                   ␣
↪ |
| type        | container-infra                                          ␣
↪ |
+-------------+----------------------------------------------------------
↪-+
```

4. Create the Container Infrastructure Management service API endpoints:

```
$ openstack endpoint create --region RegionOne \
  container-infra public http://CONTROLLER_IP:9511/v1
+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | cb137e6366ad495bb521cfe92d8b8858 |
| interface    | public                           |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | 0f7f62a1f1a247d2a4cb237642814d0e |
| service_name | magnum                           |
| service_type | container-infra                  |
| url          | http://CONTROLLER_IP:9511/v1     |
+--------------+----------------------------------+

$ openstack endpoint create --region RegionOne \
  container-infra internal http://CONTROLLER_IP:9511/v1
```

(continues on next page)

(continued from previous page)

```
+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | 17cbc3b6f51449a0a818118d6d62868d |
| interface    | internal                         |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | 0f7f62a1f1a247d2a4cb237642814d0e |
| service_name | magnum                           |
| service_type | container-infra                  |
| url          | http://CONTROLLER_IP:9511/v1     |
+--------------+----------------------------------+

$ openstack endpoint create --region RegionOne \
  container-infra admin http://CONTROLLER_IP:9511/v1
+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | 30f8888e6b6646d7b5cd14354c95a684 |
| interface    | admin                            |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | 0f7f62a1f1a247d2a4cb237642814d0e |
| service_name | magnum                           |
| service_type | container-infra                  |
| url          | http://CONTROLLER_IP:9511/v1     |
+--------------+----------------------------------+
```

Replace `CONTROLLER_IP` with the IP magnum listens to. Alternatively, you can use a hostname which is reachable by the Compute instances.

5. Magnum requires additional information in the Identity service to manage COE clusters. To add this information, complete these steps:

   - Create the `magnum` domain that contains projects and users:

   ```
   $ openstack domain create --description "Owns users and projects \
     created by magnum" magnum
     +-------------+----------------------------------------------+
     | Field       | Value                                        |
     +-------------+----------------------------------------------+
     | description | Owns users and projects created by magnum |
     | enabled     | True                                         |
     | id          | 66e0469de9c04eda9bc368e001676d20             |
     | name        | magnum                                       |
     +-------------+----------------------------------------------+
   ```

   - Create the `magnum_domain_admin` user to manage projects and users in the `magnum` domain:

```
$ openstack user create --domain magnum --password-prompt \
  magnum_domain_admin
  User Password:
  Repeat User Password:
  +-----------+----------------------------------+
  | Field     | Value                            |
  +-----------+----------------------------------+
  | domain_id | 66e0469de9c04eda9bc368e001676d20 |
  | enabled   | True                             |
  | id        | 529b81cf35094beb9784c6d06c090c2b |
  | name      | magnum_domain_admin              |
  +-----------+----------------------------------+
```

- Add the `admin` role to the `magnum_domain_admin` user in the `magnum` domain to enable administrative management privileges by the `magnum_domain_admin` user:

```
$ openstack role add --domain magnum --user-domain magnum --user \
  magnum_domain_admin admin
```

Note: This command provides no output.

Install and configure components

1. Install the common and library packages:

```
# DEBIAN_FRONTEND=noninteractive apt-get install magnum-api magnum-
↪conductor python-magnumclient
```

2. Edit the `/etc/magnum/magnum.conf` file:

   - In the `[api]` section, configure the host:

```
[api]
...
host = CONTROLLER_IP
```

     Replace `CONTROLLER_IP` with the IP address on which you wish magnum api should listen.

   - In the `[certificates]` section, select `barbican` (or `x509keypair` if you dont have barbican installed):

     – Use barbican to store certificates:

```
[certificates]
...
cert_manager_type = barbican
```

Important: Barbican is recommended for production environments.

  - To store x509 certificates in magnums database:

```
[certificates]
...
cert_manager_type = x509keypair
```

- In the `[cinder_client]` section, configure the region name:

```
[cinder_client]
...
region_name = RegionOne
```

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

```
[database]
...
connection = mysql+pymysql://magnum:MAGNUM_DBPASS@controller/magnum
```

  Replace `MAGNUM_DBPASS` with the password you chose for the magnum database.

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

```
[keystone_authtoken]
...
memcached_servers = controller:11211
auth_version = v3
www_authenticate_uri = http://controller:5000/v3
project_domain_id = default
project_name = service
user_domain_id = default
password = MAGNUM_PASS
username = magnum
auth_url = http://controller:5000
auth_type = password
admin_user = magnum
admin_password = MAGNUM_PASS
admin_tenant_name = service

[trust]
...
trustee_domain_name = magnum
trustee_domain_admin_name = magnum_domain_admin
trustee_domain_admin_password = DOMAIN_ADMIN_PASS
trustee_keystone_interface = KEYSTONE_INTERFACE
```

  Replace MAGNUM_PASS with the password you chose for the magnum user in the Identity service and DOMAIN_ADMIN_PASS with the password you chose for the `magnum_domain_admin` user.

  Replace KEYSTONE_INTERFACE with either `public` or `internal` depending on your network configuration. If your instances cannot reach internal keystone endpoint which is often the case in production environments it should be set to `public`. Default to `public`

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

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

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

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

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

3. Populate Magnum database:

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

Finalize installation

- Restart the Container Infrastructure Management services:

```
# service magnum-api restart
# service magnum-conductor restart
```

Install from source code and configure

This section describes how to install and configure the Container Infrastructure Management service for from source code.

Prerequisites

Before you install and configure the Container Infrastructure Management service, you must create a database, service credentials, and API endpoints.

1. To create the database, complete these steps:
   - Use the database access client to connect to the database server as the root user:

```
# mysql
```

   - Create the magnum database:

```
CREATE DATABASE magnum;
```

   - Grant proper access to the magnum database:

```
GRANT ALL PRIVILEGES ON magnum.* TO 'magnum'@'localhost' \
  IDENTIFIED BY 'MAGNUM_DBPASS';
GRANT ALL PRIVILEGES ON magnum.* TO 'magnum'@'%' \
  IDENTIFIED BY 'MAGNUM_DBPASS';
```

Replace `MAGNUM_DBPASS` with a suitable password.

- Exit the database access client.

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

```
$ . admin-openrc
```

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

- Create the `magnum` user:

```
$ openstack user create --domain default \
  --password-prompt magnum
User Password:
Repeat User Password:
+-----------+----------------------------------+
| Field     | Value                            |
+-----------+----------------------------------+
| domain_id | default                          |
| enabled   | True                             |
| id        | a8ebafc275c54d389dfc1bff8b4fe286 |
| name      | magnum                           |
+-----------+----------------------------------+
```

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

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

Note: This command provides no output.

- Create the `magnum` service entity:

```
$ openstack service create --name magnum \
  --description "OpenStack Container Infrastructure Management␣
↪Service" \
  container-infra
+-------------+----------------------------------------------------------
↪-+
| Field       | Value                                                    ␣
↪ |
+-------------+----------------------------------------------------------
↪-+
| description | OpenStack Container Infrastructure Management␣
↪Service |
| enabled     | True                                                     ␣
↪ |
| id          | 194faf83e8fd4e028e5ff75d3d8d0df2                         ␣
↪ |
| name        | magnum                                                   ␣
↪ |
| type        | container-infra                                          ␣
↪ |
```

(continues on next page)

(continued from previous page)

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

4. Create the Container Infrastructure Management service API endpoints:

```
$ openstack endpoint create --region RegionOne \
  container-infra public http://CONTROLLER_IP:9511/v1
+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | cb137e6366ad495bb521cfe92d8b8858 |
| interface    | public                           |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | 0f7f62a1f1a247d2a4cb237642814d0e |
| service_name | magnum                           |
| service_type | container-infra                  |
| url          | http://CONTROLLER_IP:9511/v1     |
+--------------+----------------------------------+

$ openstack endpoint create --region RegionOne \
  container-infra internal http://CONTROLLER_IP:9511/v1
+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | 17cbc3b6f51449a0a818118d6d62868d |
| interface    | internal                         |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | 0f7f62a1f1a247d2a4cb237642814d0e |
| service_name | magnum                           |
| service_type | container-infra                  |
| url          | http://CONTROLLER_IP:9511/v1     |
+--------------+----------------------------------+

$ openstack endpoint create --region RegionOne \
  container-infra admin http://CONTROLLER_IP:9511/v1
+--------------+----------------------------------+
| Field        | Value                            |
+--------------+----------------------------------+
| enabled      | True                             |
| id           | 30f8888e6b6646d7b5cd14354c95a684 |
| interface    | admin                            |
| region       | RegionOne                        |
| region_id    | RegionOne                        |
| service_id   | 0f7f62a1f1a247d2a4cb237642814d0e |
| service_name | magnum                           |
```

(continues on next page)

(continued from previous page)

```
| service_type | container-infra                    |
| url          | http://CONTROLLER_IP:9511/v1       |
+--------------+------------------------------------+
```

Replace `CONTROLLER_IP` with the IP magnum listens to. Alternatively, you can use a hostname which is reachable by the Compute instances.

5. Magnum requires additional information in the Identity service to manage COE clusters. To add this information, complete these steps:

   - Create the `magnum` domain that contains projects and users:

```
$ openstack domain create --description "Owns users and projects \
  created by magnum" magnum
  +-------------+----------------------------------------------+
  | Field       | Value                                        |
  +-------------+----------------------------------------------+
  | description | Owns users and projects created by magnum |
  | enabled     | True                                         |
  | id          | 66e0469de9c04eda9bc368e001676d20             |
  | name        | magnum                                       |
  +-------------+----------------------------------------------+
```

   - Create the `magnum_domain_admin` user to manage projects and users in the `magnum` domain:

```
$ openstack user create --domain magnum --password-prompt \
  magnum_domain_admin
  User Password:
  Repeat User Password:
  +-----------+----------------------------------+
  | Field     | Value                            |
  +-----------+----------------------------------+
  | domain_id | 66e0469de9c04eda9bc368e001676d20 |
  | enabled   | True                             |
  | id        | 529b81cf35094beb9784c6d06c090c2b |
  | name      | magnum_domain_admin              |
  +-----------+----------------------------------+
```

   - Add the `admin` role to the `magnum_domain_admin` user in the `magnum` domain to enable administrative management privileges by the `magnum_domain_admin` user:

```
$ openstack role add --domain magnum --user-domain magnum --user \
  magnum_domain_admin admin
```

Note: This command provides no output.

Install and configure components

1. Install Magnum from source:

   a. Install OS-specific prerequisites:

   - Ubuntu 16.04 (xenial) or higher:

```
# apt update
# apt install python-dev libssl-dev libxml2-dev \
              libmysqlclient-dev libxslt-dev libpq-dev git \
              libffi-dev gettext build-essential
```

   - CentOS 7:

```
# yum install python-devel openssl-devel mariadb-devel \
              libxml2-devel libxslt-devel postgresql-devel git \
              libffi-devel gettext gcc
```

   - Fedora 21 / RHEL 7

```
# yum install python-devel openssl-devel mysql-devel \
              libxml2-devel libxslt-devel postgresql-devel git \
              libffi-devel gettext gcc
```

   - Fedora 22 or higher

```
# dnf install python-devel openssl-devel mysql-devel \
              libxml2-devel libxslt-devel postgresql-devel git \
              libffi-devel gettext gcc
```

   - openSUSE Leap 42.1

```
# zypper install git libffi-devel libmysqlclient-devel \
                 libopenssl-devel libxml2-devel libxslt-devel \
                 postgresql-devel python-devel gettext-runtime gcc
```

   b. Create magnum user and necessary directories:

   - Create user:

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

   - Create directories:

```
# mkdir -p /var/log/magnum
# mkdir -p /etc/magnum
```

- Set ownership to directories:

```
# chown magnum:magnum /var/log/magnum
# chown magnum:magnum /var/lib/magnum
# chown magnum:magnum /etc/magnum
```

c. Install virtualenv and python prerequisites:

- Install virtualenv and create one for magnums installation:

```
# easy_install -U virtualenv
# su -s /bin/sh -c "virtualenv /var/lib/magnum/env" magnum
```

- Install python prerequisites:

```
# su -s /bin/sh -c "/var/lib/magnum/env/bin/pip install tox␣
↪pymysql \
  python-memcached" magnum
```

d. Clone and install magnum:

```
# cd /var/lib/magnum
# git clone https://opendev.org/openstack/magnum
# chown -R magnum:magnum magnum
# cd magnum
# su -s /bin/sh -c "/var/lib/magnum/env/bin/pip install -r␣
↪requirements.txt" magnum
# su -s /bin/sh -c "/var/lib/magnum/env/bin/python setup.py install"␣
↪magnum
```

e. Copy api-paste.ini:

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

f. Generate a sample configuration file:

```
# su -s /bin/sh -c "/var/lib/magnum/env/bin/tox -e genconfig" magnum
# su -s /bin/sh -c "cp etc/magnum/magnum.conf.sample /etc/magnum/
↪magnum.conf" magnum
```

e. Optionally, if you want to customize the policies for Magnum API accesses, you can generate a sample policy file, put it into `/etc/magnum` folder for further modifications:

```
# su -s /bin/sh -c "/var/lib/magnum/env/bin/tox -e genpolicy" magnum
# su -s /bin/sh -c "cp etc/magnum/policy.yaml.sample /etc/magnum/
↪policy.yaml" magnum
```

2. Edit the `/etc/magnum/magnum.conf` file:

- In the `[api]` section, configure the host:

```
[api]
...
host = CONTROLLER_IP
```

Replace CONTROLLER_IP with the IP address on which you wish magnum api should listen.

- In the [certificates] section, select barbican (or x509keypair if you dont have barbican installed):
  - Use barbican to store certificates:

```
[certificates]
...
cert_manager_type = barbican
```

Important: Barbican is recommended for production environments.

  - To store x509 certificates in magnums database:

```
[certificates]
...
cert_manager_type = x509keypair
```

- In the [cinder_client] section, configure the region name:

```
[cinder_client]
...
region_name = RegionOne
```

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

```
[database]
...
connection = mysql+pymysql://magnum:MAGNUM_DBPASS@controller/magnum
```

Replace MAGNUM_DBPASS with the password you chose for the magnum database.

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

```
[keystone_authtoken]
...
memcached_servers = controller:11211
auth_version = v3
www_authenticate_uri = http://controller:5000/v3
project_domain_id = default
project_name = service
user_domain_id = default
password = MAGNUM_PASS
username = magnum
auth_url = http://controller:5000
auth_type = password
admin_user = magnum
admin_password = MAGNUM_PASS
admin_tenant_name = service

[trust]
```

(continues on next page)

(continued from previous page)

```
...
trustee_domain_name = magnum
trustee_domain_admin_name = magnum_domain_admin
trustee_domain_admin_password = DOMAIN_ADMIN_PASS
trustee_keystone_interface = KEYSTONE_INTERFACE
```

Replace MAGNUM_PASS with the password you chose for the magnum user in the Identity service and DOMAIN_ADMIN_PASS with the password you chose for the `magnum_domain_admin` user.

Replace KEYSTONE_INTERFACE with either `public` or `internal` depending on your network configuration. If your instances cannot reach internal keystone endpoint which is often the case in production environments it should be set to `public`. Default to `public`

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

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

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

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

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

- Additionally, edit the `/etc/magnum/magnum.conf` file:
  - In the `[oslo_concurrency]` section, configure the `lock_path`:

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

  - If you decide to customize Magnum policies in `1.e`, then in the `[oslo_policy]` section, configure the `policy_file`:

```
[oslo_policy]
...
policy_file = /etc/magnum/policy.yaml
```

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

chown magnum:magnum /etc/magnum/magnum.conf

---

3. Populate Magnum database:

```
# su -s /bin/sh -c "/var/lib/magnum/env/bin/magnum-db-manage upgrade"␣
↪magnum
```

4. Set magnum for log rotation:

```
# cd /var/lib/magnum/magnum
# cp doc/examples/etc/logrotate.d/magnum.logrotate /etc/logrotate.d/magnum
```

Finalize installation

1. Create init scripts and services:
   - Ubuntu 16.04 or higher, Fedora 21 or higher/RHEL 7/CentOS 7 or openSUSE Leap 42.1:

```
# cd /var/lib/magnum/magnum
# cp doc/examples/etc/systemd/system/magnum-api.service \
  /etc/systemd/system/magnum-api.service
# cp doc/examples/etc/systemd/system/magnum-conductor.service \
  /etc/systemd/system/magnum-conductor.service
```

2. Start magnum-api and magnum-conductor:
   - Ubuntu 16.04 or higher, Fedora 21 or higher/RHEL 7/CentOS 7 or openSUSE Leap 42.1:

```
# systemctl enable magnum-api
# systemctl enable magnum-conductor
```

```
# systemctl start magnum-api
# systemctl start magnum-conductor
```

3. Verify that magnum-api and magnum-conductor services are running:
   - Ubuntu 16.04 or higher, Fedora 21 or higher/RHEL 7/CentOS 7 or openSUSE Leap 42.1:

```
# systemctl status magnum-api
# systemctl status magnum-conductor
```

Install the command-line client

1. Install OS-specific prerequisites:
   - Fedora 21/RHEL 7/CentOS 7

```
# yum install python-devel openssl-devel python-virtualenv \
              libffi-devel git gcc
```

   - Fedora 22 or higher

```
# dnf install python-devel openssl-devel python-virtualenv \
              libffi-devel git gcc
```

- Ubuntu

```
# apt update
# apt install python-dev libssl-dev python-virtualenv \
              libffi-dev git gcc
```

- openSUSE Leap 42.1

```
# zypper install python-devel libopenssl-devel python-virtualenv \
                 libffi-devel git gcc
```

2. Install the client in a virtual environment:

```
$ cd ~
$ git clone https://opendev.org/openstack/python-magnumclient
$ cd python-magnumclient
$ virtualenv .magnumclient-env
$ .magnumclient-env/bin/pip install -r requirements.txt
$ .magnumclient-env/bin/python setup.py install
```

3. Now, you can export the client in your PATH:

```
$ export PATH=$PATH:${PWD}/.magnumclient-env/bin/magnum
```

Note: The command-line client can be installed on the controller node or on a different host than the service. It is good practice to install it as a non-root user.

3.1.3 Verify operation

Verify operation of the Container Infrastructure Management service.

Note: Perform these commands on the controller node.

1. Source the `admin` tenant credentials:

```
$ . admin-openrc
```

2. To list out the health of the internal services, namely conductor, of magnum, use:

```
$ openstack coe service list
+----+-----------------------+------------------+-------+
| id | host                  | binary           | state |
+----+-----------------------+------------------+-------+
| 1  | controller            | magnum-conductor | up    |
+----+-----------------------+------------------+-------+
```

Note: This output should indicate a `magnum-conductor` component on the controller node.

3.1.4 Launch an instance

In environments that include the Container Infrastructure Management service, you can provision container clusters made up of virtual machines or baremetal servers. The Container Infrastructure Management service uses Cluster Templates to describe how a *Cluster* is constructed. In each of the following examples you will create a Cluster Template for a specific COE and then you will provision a Cluster using the corresponding Cluster Template. Then, you can use the appropriate COE client or endpoint to create containers.

Create an external network (Optional)

To create a magnum cluster, you need an external network. If there are no external networks, create one.

1. Create an external network with an appropriate provider based on your cloud provider support for your case:

```
$ openstack network create public --provider-network-type vxlan \
                                  --external \
                                  --project service
+---------------------------+--------------------------------------+
| Field                     | Value                                |
+---------------------------+--------------------------------------+
| admin_state_up            | UP                                   |
| availability_zone_hints   |                                      |
| availability_zones        |                                      |
| created_at                | 2017-03-27T10:09:04Z                 |
| description               |                                      |
| dns_domain                | None                                 |
| id                        | 372170ca-7d2e-48a2-8449-670e4ab66c23 |
| ipv4_address_scope        | None                                 |
| ipv6_address_scope        | None                                 |
| is_default                | False                                |
| mtu                       | 1450                                 |
| name                      | public                               |
| port_security_enabled     | True                                 |
| project_id                | 224c32c0dd2e49cbaadfd1cda069f149     |
| provider:network_type     | vxlan                                |
| provider:physical_network | None                                 |
| provider:segmentation_id  | 3                                    |
| qos_policy_id             | None                                 |
| revision_number           | 4                                    |
| router:external           | External                             |
| segments                  | None                                 |
| shared                    | False                                |
| status                    | ACTIVE                               |
| subnets                   |                                      |
| updated_at                | 2017-03-27T10:09:04Z                 |
+---------------------------+--------------------------------------+
$ openstack subnet create public-subnet --network public \
                                  --subnet-range 192.168.1.0/24 \
                                  --gateway 192.168.1.1 \
```

(continues on next page)

(continued from previous page)

```
                                --ip-version 4
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| allocation_pools  | 192.168.1.2-192.168.1.254            |
| cidr              | 192.168.1.0/24                       |
| created_at        | 2017-03-27T10:46:15Z                 |
| description       |                                      |
| dns_nameservers   |                                      |
| enable_dhcp       | True                                 |
| gateway_ip        | 192.168.1.1                          |
| host_routes       |                                      |
| id                | 04185f6c-ea31-4109-b20b-fd7f935b3828 |
| ip_version        | 4                                    |
| ipv6_address_mode | None                                 |
| ipv6_ra_mode      | None                                 |
| name              | public-subnet                        |
| network_id        | 372170ca-7d2e-48a2-8449-670e4ab66c23 |
| project_id        | d9e40a0aff30441083d9f279a0ff50de     |
| revision_number   | 2                                    |
| segment_id        | None                                 |
| service_types     |                                      |
| subnetpool_id     | None                                 |
| updated_at        | 2017-03-27T10:46:15Z                 |
+-------------------+--------------------------------------+
```

Create a keypair (Optional)

To create a magnum cluster, you need a keypair which will be passed in all compute instances of the cluster. If you dont have a keypair in your project, create one.

1. Create a keypair on the Compute service:

```
$ openstack keypair create --public-key ~/.ssh/id_rsa.pub mykey
+-------------+-------------------------------------------------+
| Field       | Value                                           |
+-------------+-------------------------------------------------+
| fingerprint | 05:be:32:07:58:a7:e8:0b:05:9b:81:6d:80:9a:4e:b1 |
| name        | mykey                                           |
| user_id     | 2d4398dbd5274707bf100a9dbbe85819                |
+-------------+-------------------------------------------------+
```

Upload the images required for your clusters to the Image service

The VM versions of Kubernetes and Docker Swarm drivers require a Fedora Atomic image. The following is stock Fedora Atomic image, built by the Atomic team and tested by the Magnum team.

1. Download the image:

```
$ wget https://download.fedoraproject.org/pub/alt/atomic/stable/Fedora-
↪Atomic-27-20180419.0/CloudImages/x86_64/images/Fedora-Atomic-27-
↪20180419.0.x86_64.qcow2
```

2. Register the image to the Image service setting the `os_distro` property to `fedora-atomic`:

```
$ openstack image create \
                      --disk-format=qcow2 \
                      --container-format=bare \
                      --file=Fedora-Atomic-27-20180419.0.x86_64.qcow2\
                      --property os_distro='fedora-atomic' \
                      fedora-atomic-latest
+------------------+------------------------------------------------------
↪+
| Field            | Value                                                ␣
↪|
+------------------+------------------------------------------------------
↪+
| checksum         | a987b691e23dce54c03d7a57c104b195                     ␣
↪|
| container_format | bare                                                 ␣
↪|
| created_at       | 2016-09-14T12:58:01Z                                 ␣
↪|
| disk_format      | qcow2                                                ␣
↪|
| file             | /v2/images/81b25935-3400-441a-9f2e-f984a46c89dd/file␣
↪|
| id               | 81b25935-3400-441a-9f2e-f984a46c89dd                 ␣
↪|
| min_disk         | 0                                                    ␣
↪|
| min_ram          | 0                                                    ␣
↪|
| name             | fedora-atomic-latest                                 ␣
↪|
| owner            | c4b42942156741dfbc4775dbcb032841                     ␣
↪|
| properties       | os_distro='fedora-atomic'                            ␣
↪|
| protected        | False                                                ␣
↪|
| schema           | /v2/schemas/image                                    ␣
↪|
```

(continues on next page)

(continued from previous page)

```
| size              | 507928064                                                    ␣
↪|
| status            | active                                                       ␣
↪|
| tags              |                                                              ␣
↪|
| updated_at        | 2016-09-14T12:58:03Z                                         ␣
↪|
| virtual_size      | None                                                         ␣
↪|
| visibility        | private                                                      ␣
↪|
+-------------------+--------------------------------------------------------------
↪+
```

Provision a Docker Swarm cluster and create a container

Following this example, you will provision a Docker Swarm cluster with one master and one node. Then, using dockers native API you will create a container.

1. Create a cluster template for a Docker Swarm cluster using the `fedora-atomic-latest` image, `m1.small` as the flavor for the master and the node, `public` as the external network and `8.8.8.8` for the DNS nameserver, using the following command:

```
$ openstack coe cluster template create swarm-cluster-template \
                    --image fedora-atomic-latest \
                    --external-network public \
                    --dns-nameserver 8.8.8.8 \
                    --master-flavor m1.small \
                    --flavor m1.small \
                    --coe swarm
+----------------------+--------------------------------------+
| Property             | Value                                |
+----------------------+--------------------------------------+
| insecure_registry    | -                                    |
| labels               | {}                                   |
| updated_at           | -                                    |
| floating_ip_enabled  | True                                 |
| fixed_subnet         | -                                    |
| master_flavor_id     | m1.small                             |
| uuid                 | 47c6ce77-50ae-43bd-8e2a-06980392693d |
| no_proxy             | -                                    |
| https_proxy          | -                                    |
| tls_disabled         | False                                |
| keypair_id           | mykey                                |
| public               | False                                |
| http_proxy           | -                                    |
| docker_volume_size   | -                                    |
| server_type          | vm                                   |
```

(continues on next page)

(continued from previous page)

```
| external_network_id   | public                               |
| cluster_distro        | fedora-atomic                        |
| image_id              | fedora-atomic-latest                 |
| volume_driver         | -                                    |
| registry_enabled      | False                                |
| docker_storage_driver | devicemapper                         |
| apiserver_port        | -                                    |
| name                  | swarm-cluster-template               |
| created_at            | 2016-09-14T13:05:11+00:00            |
| network_driver        | docker                               |
| fixed_network         | -                                    |
| coe                   | swarm                                |
| flavor_id             | m1.small                             |
| master_lb_enabled     | False                                |
| dns_nameserver        | 8.8.8.8                              |
+-----------------------+--------------------------------------+
```

2. Create a cluster with one node and one master using `mykey` as the keypair, using the following command:

```
$ openstack coe cluster create swarm-cluster \
                        --cluster-template swarm-cluster-template \
                        --master-count 1 \
                        --node-count 1 \
                        --keypair mykey
Request to create cluster 2582f192-480e-4329-ac05-32a8e5b1166b has been␣
↪accepted.
```

Your cluster is now being created. Creation time depends on your infrastructures performance. You can check the status of your cluster using the commands: `openstack coe cluster list` or `openstack coe cluster show swarm-cluster`.

```
$ openstack coe cluster list
+--------------------------------------+---------------+---------+--------
↪----+-------------+-----------------+
| uuid                                 | name          | keypair | node_
↪count | master_count | status          |
+--------------------------------------+---------------+---------+--------
↪----+-------------+-----------------+
| 2582f192-480e-4329-ac05-32a8e5b1166b | swarm-cluster | mykey   | 1     ␣
↪    | 1            | CREATE_COMPLETE |
+--------------------------------------+---------------+---------+--------
↪----+-------------+-----------------+
```

```
$ openstack coe cluster show swarm-cluster
+---------------------+------------------------------------------------------
↪---------+
| Property            | Value                                                ␣
↪         |
+---------------------+------------------------------------------------------
↪---------+
```

(continues on next page)

(continued from previous page)

```
| status               | CREATE_COMPLETE                                                ␣
↪          |
| cluster_template_id | 47c6ce77-50ae-43bd-8e2a-06980392693d                            ␣
↪          |
| uuid                | 2582f192-480e-4329-ac05-32a8e5b1166b                            ␣
↪          |
| stack_id            | 3d7bbf1c-49bd-4930-84e0-ab71ba200687                            ␣
↪          |
| status_reason       | Stack CREATE completed successfully                             ␣
↪          |
| created_at          | 2016-09-14T13:36:54+00:00                                       ␣
↪          |
| name                | swarm-cluster                                                   ␣
↪          |
| updated_at          | 2016-09-14T13:38:08+00:00                                       ␣
↪          |
| discovery_url       | https://discovery.etcd.io/
↪a5ece414689287eca62e35555512bfd5 |
| api_address         | tcp://172.24.4.10:2376                                          ␣
↪          |
| coe_version         | 1.2.5                                                           ␣
↪          |
| master_addresses    | ['172.24.4.10']                                                 ␣
↪          |
| create_timeout      | 60                                                              ␣
↪          |
| node_addresses      | ['172.24.4.8']                                                  ␣
↪          |
| master_count        | 1                                                               ␣
↪          |
| container_version   | 1.12.6                                                          ␣
↪          |
| node_count          | 1                                                               ␣
↪          |
+---------------------+-----------------------------------------------------
↪---------+
```

3. Add the credentials of the above cluster to your environment:

```
$ mkdir myclusterconfig
$ $(openstack coe cluster config swarm-cluster --dir myclusterconfig)
```

The above command will save the authentication artifacts in the *myclusterconfig* directory and it will export the environment variables: DOCKER_HOST, DOCKER_CERT_PATH and DOCKER_TLS_VERIFY. Sample output:

```
export DOCKER_HOST=tcp://172.24.4.10:2376
export DOCKER_CERT_PATH=myclusterconfig
export DOCKER_TLS_VERIFY=True
```

4. Create a container:

```
$ docker run busybox echo "Hello from Docker!"
Hello from Docker!
```

5. Delete the cluster:

```
$ openstack coe cluster delete swarm-cluster
Request to delete cluster swarm-cluster has been accepted.
```

Provision a Kubernetes cluster and create a deployment

Following this example, you will provision a Kubernetes cluster with one master and one node. Then, using Kubernetess native client `kubectl`, you will create a deployment.

1. Create a cluster template for a Kubernetes cluster using the `fedora-coreos-latest` image, `m1.small` as the flavor for the master and the node, `public` as the external network and `8.8.8.8` for the DNS nameserver, using the following command:

```
$ openstack coe cluster template create kubernetes-cluster-template \
                    --image fedora-coreos-latest \
                    --external-network public \
                    --dns-nameserver 8.8.8.8 \
                    --master-flavor m1.small \
                    --flavor m1.small \
                    --coe kubernetes
+-----------------------+--------------------------------------+
| Property              | Value                                |
+-----------------------+--------------------------------------+
| insecure_registry     | -                                    |
| labels                | {}                                   |
| updated_at            | -                                    |
| floating_ip_enabled   | True                                 |
| fixed_subnet          | -                                    |
| master_flavor_id      | m1.small                             |
| uuid                  | 0a601cc4-8fef-41aa-8036-d113e719ed7a |
| no_proxy              | -                                    |
| https_proxy           | -                                    |
| tls_disabled          | False                                |
| keypair_id            | -                                    |
| public                | False                                |
| http_proxy            | -                                    |
| docker_volume_size    | -                                    |
| server_type           | vm                                   |
| external_network_id   | public                               |
| cluster_distro        | fedora-atomic                        |
| image_id              | fedora-atomic-latest                 |
| volume_driver         | -                                    |
| registry_enabled      | False                                |
| docker_storage_driver | devicemapper                         |
| apiserver_port        | -                                    |
| name                  | kubernetes-cluster-template          |
```

(continues on next page)

(continued from previous page)

```
| created_at            | 2017-05-16T09:53:00+00:00                |
| network_driver        | flannel                                  |
| fixed_network         | -                                        |
| coe                   | kubernetes                               |
| flavor_id             | m1.small                                 |
| master_lb_enabled     | False                                    |
| dns_nameserver        | 8.8.8.8                                  |
+-----------------------+------------------------------------------+
```

2. Create a cluster with one node and one master using `mykey` as the keypair, using the following command:

```
$ openstack coe cluster create kubernetes-cluster \
                        --cluster-template kubernetes-cluster-template \
                        --master-count 1 \
                        --node-count 1 \
                        --keypair mykey
Request to create cluster b1ef3528-ac03-4459-bbf7-22649bfbc84f has been␣
↪accepted.
```

Your cluster is now being created. Creation time depends on your infrastructures performance. You can check the status of your cluster using the commands: `openstack coe cluster list` or `openstack coe cluster show kubernetes-cluster`.

```
$ openstack coe cluster list
+--------------------------------------+--------------------+---------+---
↪---------+--------------+-----------------+
| uuid                                 | name               | keypair |␣
↪node_count | master_count | status          |
+--------------------------------------+--------------------+---------+---
↪---------+--------------+-----------------+
| b1ef3528-ac03-4459-bbf7-22649bfbc84f | kubernetes-cluster | mykey   | 1␣
↪         | 1            | CREATE_COMPLETE |
+--------------------------------------+--------------------+---------+---
↪---------+--------------+-----------------+
```

```
$ openstack coe cluster show kubernetes-cluster
+---------------------+------------------------------------------------------
↪---------+
| Property            | Value                                                ␣
↪         |
+---------------------+------------------------------------------------------
↪---------+
| status              | CREATE_COMPLETE                                      ␣
↪         |
| cluster_template_id | 0a601cc4-8fef-41aa-8036-d113e719ed7a                 ␣
↪         |
| node_addresses      | ['172.24.4.5']                                       ␣
↪         |
| uuid                | b1ef3528-ac03-4459-bbf7-22649bfbc84f                 ␣
↪         |
```

(continues on next page)

(continued from previous page)

```
| stack_id              | 8296624c-3c0e-45e1-967e-b6ff05105a3b
↪          |
| status_reason         | Stack CREATE completed successfully
↪          |
| created_at            | 2017-05-16T09:58:02+00:00
↪          |
| updated_at            | 2017-05-16T10:00:02+00:00
↪          |
| coe_version           | v1.6.7
↪          |
| keypair               | default
↪          |
| api_address           | https://172.24.4.13:6443
↪          |
| master_addresses      | ['172.24.4.13']
↪          |
| create_timeout        | 60
↪          |
| node_count            | 1
↪          |
| discovery_url         | https://discovery.etcd.io/
↪69c7cd3b3b06c98b4771410bd166a7c6 |
| master_count          | 1
↪          |
| container_version     | 1.12.6
↪          |
| name                  | kubernetes-cluster
↪          |
+-----------------------+------------------------------------------------------
↪---------+
```

3. Add the credentials of the above cluster to your environment:

```
$ mkdir -p ~/clusters/kubernetes-cluster
$ $(openstack coe cluster config kubernetes-cluster --dir ~/clusters/
↪kubernetes-cluster)
```

The above command will save the authentication artifacts in the directory `~/clusters/kubernetes-cluster` and it will export the `KUBECONFIG` environment variable:

```
export KUBECONFIG=/home/user/clusters/kubernetes-cluster/config
```

4. You can list the controller components of your Kubernetes cluster and check if they are `Running`:

```
$ kubectl -n kube-system get po
NAME
↪      READY     STATUS    RESTARTS   AGE
kube-controller-manager-ku-hesuip7l3i-0-5mqijvszepxw-kube-master-
↪rqwmwne7rjh2   1/1       Running   0          1h
kube-proxy-ku-hesuip7l3i-0-5mqijvszepxw-kube-master-rqwmwne7rjh2
↪      1/1       Running   0          1h
```

(continues on next page)

(continued from previous page)

```
kube-proxy-ku-wmmticfvdr-0-k53p22xmlxvx-kube-minion-x4ly6zfhrrui
↪       1/1       Running   0          1h
kube-scheduler-ku-hesuip7l3i-0-5mqijvszepxw-kube-master-rqwmwne7rjh2
↪       1/1       Running   0          1h
kubernetes-dashboard-3203831700-zvj2d
↪       1/1       Running   0          1h
```

5. Now, you can create a nginx deployment and verify it is running:

```
$ kubectl run nginx --image=nginx --replicas=5
deployment "nginx" created
$ kubectl get po
NAME                     READY     STATUS    RESTARTS   AGE
nginx-701339712-2ngt8    1/1       Running   0          15s
nginx-701339712-j8r3d    1/1       Running   0          15s
nginx-701339712-mb6jb    1/1       Running   0          15s
nginx-701339712-q115k    1/1       Running   0          15s
nginx-701339712-tb5lp    1/1       Running   0          15s
```

6. Delete the cluster:

```
$ openstack coe cluster delete kubernetes-cluster
Request to delete cluster kubernetes-cluster has been accepted.
```

3.1.5 Next steps

Your OpenStack environment now includes the magnum service.

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

The Container Infrastructure Management service codenamed (magnum) is an OpenStack API service developed by the OpenStack Containers Team making container orchestration engines (COE) such as Docker Swarm, Kubernetes and Mesos available as first class resources in OpenStack. Magnum uses Heat to orchestrate an OS image which contains Docker and Kubernetes and runs that image in either virtual machines or bare metal in a cluster configuration.

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

CHAPTER

FOUR

USER DOCUMENTATION

4.1 Magnum User Guide

This guide is intended for users who use Magnum to deploy and manage clusters of hosts for a Container Orchestration Engine. It describes the infrastructure that Magnum creates and how to work with them.

Section 1-3 describe Magnum itself, including an overview, the CLI and Horizon interface. Section 4-9 describe the Container Orchestration Engine (COE) supported along with a guide on how to select one that best meets your needs and how to develop a driver for a new COE. Section 10-15 describe the low level OpenStack infrastructure that is created and managed by Magnum to support the COEs.

1. *Overview*
2. *Python Client*
3. *Horizon Interface*
4. *Cluster Drivers*
5. *Cluster Type Definition*
6. *Heat Stack Templates*
7. *Choosing a COE*
8. *Native Clients*
9. *Kubernetes*
10. *Swarm*
11. *Mesos*
12. *Transport Layer Security*
13. *Networking*
14. *High Availability*
15. *Scaling*
16. *Storage*
17. *Image Management*
18. *Notification*
19. *Container Monitoring*
20. *Kubernetes Post Install Manifest*

21. *Kubernetes External Load Balancer*
22. *Rolling Upgrade*
23. *Keystone Authentication and Authorization for Kubernetes*
24. *Node Groups*
25. *Kubernetes Health Monitoring*

4.1.1 Overview

Magnum is an OpenStack API service developed by the OpenStack Containers Team making container orchestration engines (COE) such as Docker Swarm, Kubernetes and Apache Mesos available as first class resources in OpenStack.

Magnum uses Heat to orchestrate an OS image which contains Docker and COE and runs that image in either virtual machines or bare metal in a cluster configuration.

Magnum offers complete life-cycle management of COEs in an OpenStack environment, integrated with other OpenStack services for a seamless experience for OpenStack users who wish to run containers in an OpenStack environment.

Following are few salient features of Magnum:

- Standard API based complete life-cycle management for Container Clusters
- Multi-tenancy for container clusters
- Choice of COE: Kubernetes, Swarm, Mesos, DC/OS
- Choice of container cluster deployment model: VM or Bare-metal
- Keystone-based multi-tenant security and auth management
- Neutron based multi-tenant network control and isolation
- Cinder based volume service for containers
- Integrated with OpenStack: SSO experience for cloud users
- Secure container cluster access (TLS enabled)

More details: Magnum Project Wiki

ClusterTemplate

A ClusterTemplate (previously known as BayModel) is a collection of parameters to describe how a cluster can be constructed. Some parameters are relevant to the infrastructure of the cluster, while others are for the particular COE. In a typical workflow, a user would create a ClusterTemplate, then create one or more clusters using the ClusterTemplate. A cloud provider can also define a number of ClusterTemplates and provide them to the users. A ClusterTemplate cannot be updated or deleted if a cluster using this ClusterTemplate still exists.

The definition and usage of the parameters of a ClusterTemplate are as follows. They are loosely grouped as: mandatory, infrastructure, COE specific.

<name> Name of the ClusterTemplate to create. The name does not have to be unique. If multiple ClusterTemplates have the same name, you will need to use the UUID to select the ClusterTemplate

when creating a cluster or updating, deleting a ClusterTemplate. If a name is not specified, a random name will be generated using a string and a number, for example pi-13-model.

coe <coe> Specify the Container Orchestration Engine to use. Supported COEs include kubernetes, swarm, mesos. If your environment has additional cluster drivers installed, refer to the cluster driver documentation for the new COE names. This is a mandatory parameter and there is no default value.

image <image> The name or UUID of the base image in Glance to boot the servers for the cluster. The image must have the attribute os_distro defined as appropriate for the cluster driver. For the currently supported images, the os_distro names are:

COE	os_distro
Kubernetes	fedora-coreos
Swarm	fedora-atomic
Mesos	ubuntu

This is a mandatory parameter and there is no default value. Note that the os_distro attribute is case sensitive.

keypair <keypair> The name of the SSH keypair to configure in the cluster servers for ssh access. You will need the key to be able to ssh to the servers in the cluster. The login name is specific to the cluster driver. If keypair is not provided in template it will be required at Cluster create. This value will be overridden by any keypair value that is provided during Cluster create.

external-network <external-network> The name or network ID of a Neutron network to provide connectivity to the external internet for the cluster. This network must be an external network, i.e. its attribute router:external must be True. The servers in the cluster will be connected to a private network and Magnum will create a router between this private network and the external network. This will allow the servers to download images, access discovery service, etc, and the containers to install packages, etc. In the opposite direction, floating IPs will be allocated from the external network to provide access from the external internet to servers and the container services hosted in the cluster. This is a mandatory parameter and there is no default value.

--public Access to a ClusterTemplate is normally limited to the admin, owner or users within the same tenant as the owners. Setting this flag makes the ClusterTemplate public and accessible by other users. The default is not public.

server-type <server-type> The servers in the cluster can be VM or baremetal. This parameter selects the type of server to create for the cluster. The default is vm. Possible values are vm, bm.

network-driver <network-driver> The name of a network driver for providing the networks for the containers. Note that this is different and separate from the Neutron network for the cluster. The operation and networking model are specific to the particular driver; refer to the *Networking* section for more details. Supported network drivers and the default driver are:

COE	Network-Driver	Default
Kubernetes	flannel, calico	flannel
Swarm	docker, flannel	flannel
Mesos	docker	docker

Note that the network driver name is case sensitive.

volume-driver <volume-driver> The name of a volume driver for managing the persistent storage for the containers. The functionality supported are specific to the driver. Supported volume drivers and the default driver are:

COE	Volume-Driver	Default
Kubernetes	cinder	No Driver
Swarm	rexray	No Driver
Mesos	rexray	No Driver

Note that the volume driver name is case sensitive.

dns-nameserver <dns-nameserver> The DNS nameserver for the servers and containers in the cluster to use. This is configured in the private Neutron network for the cluster. The default is 8.8.8.8.

flavor <flavor> The nova flavor id for booting the node servers. The default is m1.small. This value can be overridden at cluster creation.

master-flavor <master-flavor> The nova flavor id for booting the master or manager servers. The default is m1.small. This value can be overridden at cluster creation.

http-proxy <http-proxy> The IP address for a proxy to use when direct http access from the servers to sites on the external internet is blocked. This may happen in certain countries or enterprises, and the proxy allows the servers and containers to access these sites. The format is a URL including a port number. The default is None.

https-proxy <https-proxy> The IP address for a proxy to use when direct https access from the servers to sites on the external internet is blocked. This may happen in certain countries or enterprises, and the proxy allows the servers and containers to access these sites. The format is a URL including a port number. The default is None.

no-proxy <no-proxy> When a proxy server is used, some sites should not go through the proxy and should be accessed normally. In this case, you can specify these sites as a comma separated list of IPs. The default is None.

docker-volume-size <docker-volume-size> If specified, container images will be stored in a cinder volume of the specified size in GB. Each cluster node will have a volume attached of the above size. If not specified, images will be stored in the compute instances local disk. For the devicemapper storage driver, must specify volume and the minimum value is 3GB. For the overlay and overlay2 storage driver, the minimum value is 1GB or None(no volume). This value can be overridden at cluster creation.

docker-storage-driver <docker-storage-driver> The name of a driver to manage the storage for the images and the containers writable layer. The default is devicemapper.

labels <KEY1=VALUE1,KEY2=VALUE2;KEY3=VALUE3> Arbitrary labels in the form of key=value pairs. The accepted keys and valid values are defined in the cluster drivers. They are used as a way to pass additional parameters that are specific to a cluster driver. Refer to the subsection on labels for a list of the supported key/value pairs and their usage. The value can be overridden at cluster creation.

--tls-disabled Transport Layer Security (TLS) is normally enabled to secure the cluster. In some cases, users may want to disable TLS in the cluster, for instance during development or to troubleshoot certain problems. Specifying this parameter will disable TLS so that users can access the COE endpoints without a certificate. The default is TLS enabled.

--registry-enabled Docker images by default are pulled from the public Docker registry, but in some cases, users may want to use a private registry. This option provides an alternative registry based on the Registry V2: Magnum will create a local registry in the cluster backed by swift to host the images. Refer to Docker Registry 2.0 for more details. The default is to use the public registry.

--master-lb-enabled Since multiple masters may exist in a cluster, a load balancer is created to provide the API endpoint for the cluster and to direct requests to the masters. In some cases, such as when the LBaaS service is not available, this option can be set to false to create a cluster without the load balancer. In this case, one of the masters will serve as the API endpoint. The default is true, i.e. to create the load balancer for the cluster.

Labels

Labels is a general method to specify supplemental parameters that are specific to certain COE or associated with certain options. Their format is key/value pair and their meaning is interpreted by the drivers that uses them. The drivers do validate the key/value pairs. Their usage is explained in details in the appropriate sections, however, since there are many possible labels, the following table provides a summary to help give a clearer picture. The label keys in the table are linked to more details elsewhere in the user guide.

label key	label value	default
flannel_network_cidr	IPv4 CIDR	10.100.0.0/16
flannel_backend	• udp • vxlan • host-gw	vxlan
flannel_network_subnetlen	size of subnet to assign to node	24
rexray_preempt	• true • false	false
mesos_slave_isolation	• filesystem/posix • filesystem/linux • filesystem/shared • posix/cpu • posix/mem • posix/disk • cgroups/cpu • cgroups/mem • docker/runtime • namespaces/pid	

continues on next page

Table 1 – continued from previous page

label key	label value	default
mesos_slave_image_providers	• appc • docker • appc,docker	
mesos_slave_work_dir	(directory name)	
mesos_slave_executor_env_variables	(file name)	
heapster_enabled	• true • false	false
metrics_server_chart_tag	see below	see below
metrics_server_enabled	• true • false	true
monitoring_enabled	• true • false	false
monitoring_retention_days	see below	see below
monitoring_retention_size	see below	see below
monitoring_storage_class_name	see below	see below
monitoring_interval_seconds	see below	see below
monitoring_ingress_enabled	• true • false	false
cluster_basic_auth_secret	see below	see below
cluster_root_domain_name	see below	see below
prometheus_operator_chart_tag	see below	see below
prometheus_adapter_enabled	• true • false	true
prometheus_adapter_chart_tag	see below	see below
prometheus_adapter_configmap	(rules CM name)	
swarm_strategy	• spread • binpack • random	spread
traefik_ingress_controller_tag	see below	see below
admission_control_list	see below	see below

continues on next page

Table 1 – continued from previous page

label key	label value	default
prometheus_monitoring (deprecated)	• true • false	false
grafana_admin_passwd	(any string)	admin
hyperkube_prefix	see below	see below
kube_tag	see below	see below
cloud_provider_tag	see below	see below
etcd_tag	see below	see below
coredns_tag	see below	see below
flannel_tag	see below	see below
flannel_cni_tag	see below	see below
heat_container_agent_tag	see below	see below
kube_dashboard_enabled	• true • false	true
kube_dashboard_version	see below	see below
metrics_scraper_tag	see below	see below
influx_grafana_dashboard_enabled	• true • false	false
docker_volume_type	see below	see below
boot_volume_size	see below	see below
boot_volume_type	see below	see below
etcd_volume_size	etcd storage volume size	0
etcd_volume_type	see below	see below
container_infra_prefix	see below	
availability_zone	AZ for the cluster nodes	
cert_manager_api	see below	false
ingress_controller	see below	
ingress_controller_role	see below	ingress
octavia_ingress_controller_tag	see below	see below
nginx_ingress_controller_tag	see below	see below
nginx_ingress_controller_chart_tag	see below	see below
kubelet_options	extra kubelet args	
kubeapi_options	extra kubeapi args	
kubescheduler_options	extra kubescheduler args	
kubecontroller_options	extra kubecontroller args	
kubeproxy_options	extra kubeproxy args	
cgroup_driver	• systemd • cgroupfs	cgroupfs

continues on next page

Table 1 – continued from previous page

label key	label value	default
cloud_provider_enabled	• true • false	see below
service_cluster_ip_range	IPv4 CIDR for k8s service portals	10.254.0.0/16
keystone_auth_enabled	see below	true
k8s_keystone_auth_tag	see below	see below
tiller_enabled	• true • false	false
tiller_tag	see below	
tiller_namespace	see below	see below
helm_client_url	see below	see below
helm_client_sha256	see below	see below
helm_client_tag	see below	see below
master_lb_floating_ip_enabled	• true • false	see below
master_lb_allowed_cidrs	see below	
auto_healing_enabled	• true • false	false
auto_healing_controller	see below	draino
magnum_auto_healer_tag	see below	see below
auto_scaling_enabled	• true • false	false
node_problem_detector_tag	see below	see below
draino_tag	see below	see below
autoscaler_tag	see below	see below
min_node_count	see below	see below
max_node_count	see below	see below
npd_enabled	• true • false	true
use_podman	• true • false	see below

continues on next page

Table 1 – continued from previous page

label key	label value	default
selinux_mode	• enforcing • permissive • disabled	see below
container_runtime	• • containerd	
containerd_version	see below	see below
containerd_tarball_url	see below	see below
containerd_tarball_sha256	see below	see below
calico_tag	see below	see below
calico_ipv4pool	see below	10.100.0.0/16
calico_ipv4pool_ipip	see below	Off
fixed_subnet_cidr	see below	

Cluster

A cluster (previously known as bay) is an instance of the ClusterTemplate of a COE. Magnum deploys a cluster by referring to the attributes defined in the particular ClusterTemplate as well as a few additional parameters for the cluster. Magnum deploys the orchestration templates provided by the cluster driver to create and configure all the necessary infrastructure. When ready, the cluster is a fully operational COE that can host containers.

Infrastructure

The infrastructure of the cluster consists of the resources provided by the various OpenStack services. Existing infrastructure, including infrastructure external to OpenStack, can also be used by the cluster, such as DNS, public network, public discovery service, Docker registry. The actual resources created depends on the COE type and the options specified; therefore you need to refer to the cluster driver documentation of the COE for specific details. For instance, the option master-lb-enabled in the ClusterTemplate will cause a load balancer pool along with the health monitor and floating IP to be created. It is important to distinguish resources in the IaaS level from resources in the PaaS level. For instance, the infrastructure networking in OpenStack IaaS is different and separate from the container networking in Kubernetes or Swarm PaaS.

Typical infrastructure includes the following.

Servers The servers host the containers in the cluster and these servers can be VM or bare metal. VMs are provided by Nova. Since multiple VMs are hosted on a physical server, the VMs provide the isolation needed for containers between different tenants running on the same physical server. Bare metal servers are provided by Ironic and are used when peak performance with virtually no overhead is needed for the containers.

Identity Keystone provides the authentication and authorization for managing the cluster infrastructure.

Network Networking among the servers is provided by Neutron. Since COE currently are not multi-tenant, isolation for multi-tenancy on the networking level is done by using a private network for

each cluster. As a result, containers belonging to one tenant will not be accessible to containers or servers of another tenant. Other networking resources may also be used, such as load balancer and routers. Networking among containers can be provided by Kuryr if needed.

Storage Cinder provides the block storage that can be used to host the containers and as persistent storage for the containers.

Security Barbican provides the storage of secrets such as certificates used for Transport Layer Security (TLS) within the cluster.

Life cycle

The set of life cycle operations on the cluster is one of the key value that Magnum provides, enabling clusters to be managed painlessly on OpenStack. The current operations are the basic CRUD operations, but more advanced operations are under discussion in the community and will be implemented as needed.

NOTE The OpenStack resources created for a cluster are fully accessible to the cluster owner. Care should be taken when modifying or reusing these resources to avoid impacting Magnum operations in unexpected manners. For instance, if you launch your own Nova instance on the cluster private network, Magnum would not be aware of this instance. Therefore, the cluster-delete operation will fail because Magnum would not delete the extra Nova instance and the private Neutron network cannot be removed while a Nova instance is still attached.

NOTE Currently Heat nested templates are used to create the resources; therefore if an error occurs, you can troubleshoot through Heat. For more help on Heat stack troubleshooting, refer to the *Magnum Troubleshooting Guide*.

Create

NOTE bay-<command> are the deprecated versions of these commands and are still support in current release. They will be removed in a future version. Any references to the term bay will be replaced in the parameters when using the bay versions of the commands. For example, in bay-create baymodel is used as the baymodel parameter for this command instead of cluster-template.

The cluster-create command deploys a cluster, for example:

```
openstack coe cluster create mycluster \
                  --cluster-template mytemplate \
                  --node-count 8 \
                  --master-count 3
```

The cluster-create operation is asynchronous; therefore you can initiate another cluster-create operation while the current cluster is being created. If the cluster fails to be created, the infrastructure created so far may be retained or deleted depending on the particular orchestration engine. As a common practice, a failed cluster is retained during development for troubleshooting, but they are automatically deleted in production. The current cluster drivers use Heat templates and the resources of a failed cluster-create are retained.

The definition and usage of the parameters for cluster-create are as follows:

<name> Name of the cluster to create. If a name is not specified, a random name will be generated using a string and a number, for example gamma-7-cluster.

cluster-template <cluster-template> The ID or name of the ClusterTemplate to use. This is a mandatory parameter. Once a ClusterTemplate is used to create a cluster, it cannot be deleted or modified until all clusters that use the ClusterTemplate have been deleted.

keypair <keypair> The name of the SSH keypair to configure in the cluster servers for ssh access. You will need the key to be able to ssh to the servers in the cluster. The login name is specific to the cluster driver. If keypair is not provided it will attempt to use the value in the ClusterTemplate. If the ClusterTemplate is also missing a keypair value then an error will be returned. The keypair value provided here will override the keypair value from the ClusterTemplate.

node-count <node-count> The number of servers that will serve as node in the cluster. The default is 1.

master-count <master-count> The number of servers that will serve as master for the cluster. The default is 1. Set to more than 1 master to enable High Availability. If the option master-lb-enabled is specified in the ClusterTemplate, the master servers will be placed in a load balancer pool.

discovery-url <discovery-url> The custom discovery url for node discovery. This is used by the COE to discover the servers that have been created to host the containers. The actual discovery mechanism varies with the COE. In some cases, Magnum fills in the server info in the discovery service. In other cases, if the discovery-url is not specified, Magnum will use the public discovery service at:

```
https://discovery.etcd.io
```

In this case, Magnum will generate a unique url here for each cluster and store the info for the servers.

timeout <timeout> The timeout for cluster creation in minutes. The value expected is a positive integer and the default is 60 minutes. If the timeout is reached during cluster-create, the operation will be aborted and the cluster status will be set to CREATE_FAILED.

--master-lb-enabled Indicates whether created clusters should have a load balancer for master nodes or not.

List

The cluster-list command lists all the clusters that belong to the tenant, for example:

```
openstack coe cluster list
```

Show

The cluster-show command prints all the details of a cluster, for example:

```
openstack coe cluster show mycluster
```

The properties include those not specified by users that have been assigned default values and properties from new resources that have been created for the cluster.

Update

A cluster can be modified using the cluster-update command, for example:

```
openstack coe cluster update mycluster replace node_count=8
```

The parameters are positional and their definition and usage are as follows.

<cluster> This is the first parameter, specifying the UUID or name of the cluster to update.

<op> This is the second parameter, specifying the desired change to be made to the cluster attributes. The allowed changes are add, replace and remove.

<attribute=value> This is the third parameter, specifying the targeted attributes in the cluster as a list separated by blank space. To add or replace an attribute, you need to specify the value for the attribute. To remove an attribute, you only need to specify the name of the attribute. Currently the only attribute that can be replaced or removed is node_count. The attributes name, master_count and discovery_url cannot be replaced or delete. The table below summarizes the possible change to a cluster.

Attribute	add	replace	remove
node_count	no	add/remove nodes in default-worker nodegroup.	reset to default of 1
master_count	no	no	no
name	no	no	no
discovery_url	no	no	no

The cluster-update operation cannot be initiated when another operation is in progress.

NOTE: The attribute names in cluster-update are slightly different from the corresponding names in the cluster-create command: the dash - is replaced by an underscore _. For instance, node-count in cluster-create is node_count in cluster-update.

Scale

Scaling a cluster means adding servers to or removing servers from the cluster. Currently, this is done through the cluster-update operation by modifying the node-count attribute, for example:

```
openstack coe cluster update mycluster replace node_count=2
```

When some nodes are removed, Magnum will attempt to find nodes with no containers to remove. If some nodes with containers must be removed, Magnum will log a warning message.

Delete

The cluster-delete operation removes the cluster by deleting all resources such as servers, network, storage; for example:

```
openstack coe cluster delete mycluster
```

The only parameter for the cluster-delete command is the ID or name of the cluster to delete. Multiple clusters can be specified, separated by a blank space.

If the operation fails, there may be some remaining resources that have not been deleted yet. In this case, you can troubleshoot through Heat. If the templates are deleted manually in Heat, you can delete the cluster in Magnum to clean up the cluster from Magnum database.

The cluster-delete operation can be initiated when another operation is still in progress.

4.1.2 Python Client

Installation

Follow the instructions in the OpenStack Installation Guide to enable the repositories for your distribution:

- RHEL/CentOS/Fedora
- Ubuntu/Debian
- openSUSE/SUSE Linux Enterprise

Install using distribution packages for RHEL/CentOS/Fedora:

```
$ sudo yum install python-magnumclient
```

Install using distribution packages for Ubuntu/Debian:

```
$ sudo apt-get install python-magnumclient
```

Install using distribution packages for openSUSE and SUSE Enterprise Linux:

```
$ sudo zypper install python3-magnumclient
```

Verifying installation

Execute the *openstack coe cluster list* command to confirm that the client is installed and in the system path:

```
$ openstack coe cluster list
```

Using the command-line client

Refer to the OpenStack Command-Line Interface Reference for a full list of the commands supported by the *openstack coe* command-line client.

4.1.3 Horizon Interface

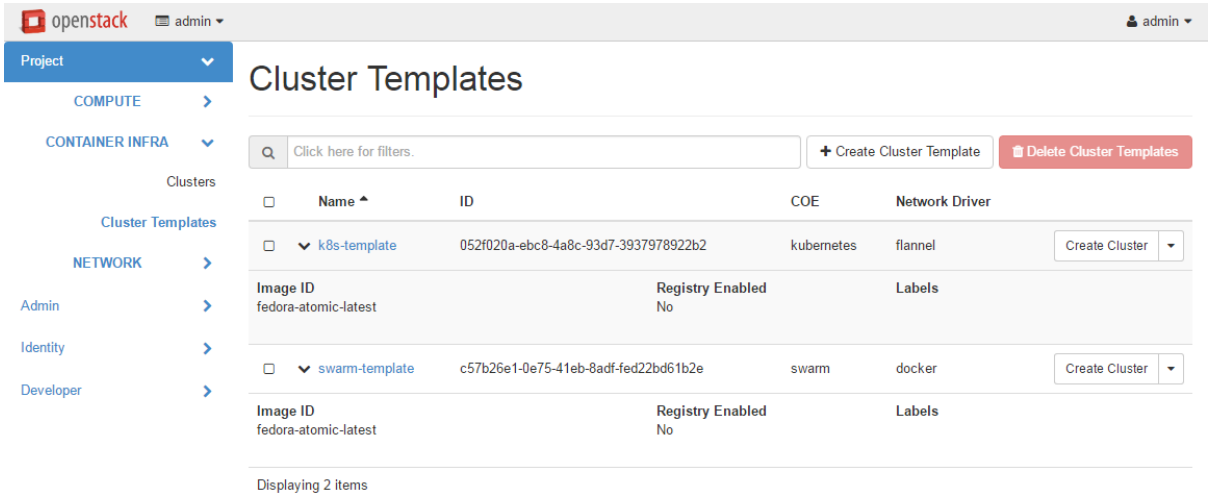
Magnum provides a Horizon plugin so that users can access the Container Infrastructure Management service through the OpenStack browser-based graphical UI. The plugin is available from magnum-ui. It is not installed by default in the standard Horizon service, but you can follow the instruction for installing a Horizon plugin.

In Horizon, the container infrastructure panel is part of the Project view and it currently supports the following operations:

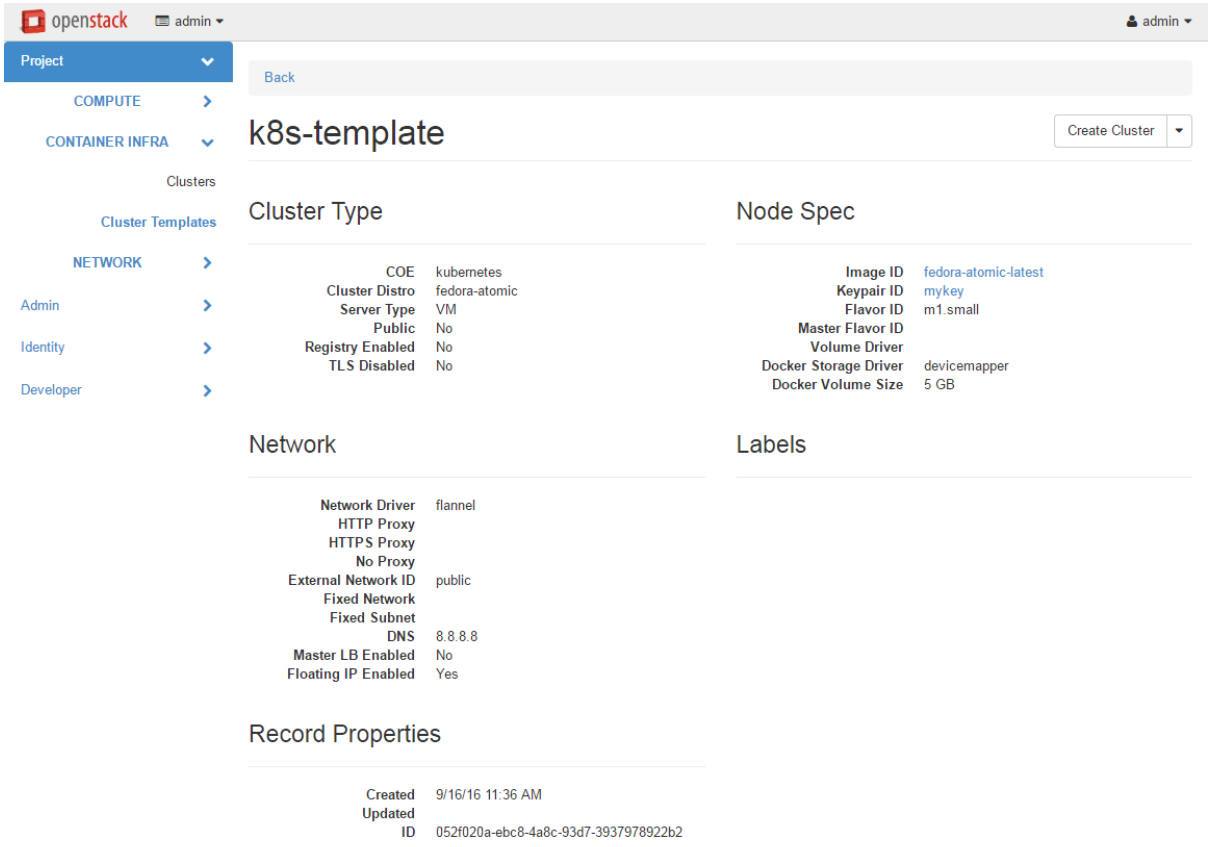
- View list of cluster templates
- View details of a cluster template
- Create a cluster template
- Delete a cluster template
- View list of clusters
- View details of a cluster
- Create a cluster
- Delete a cluster
- Get the Certificate Authority for a cluster
- Sign a user key and obtain a signed certificate for accessing the secured COE API endpoint in a cluster.

Other operations are not yet supported and the CLI should be used for these.

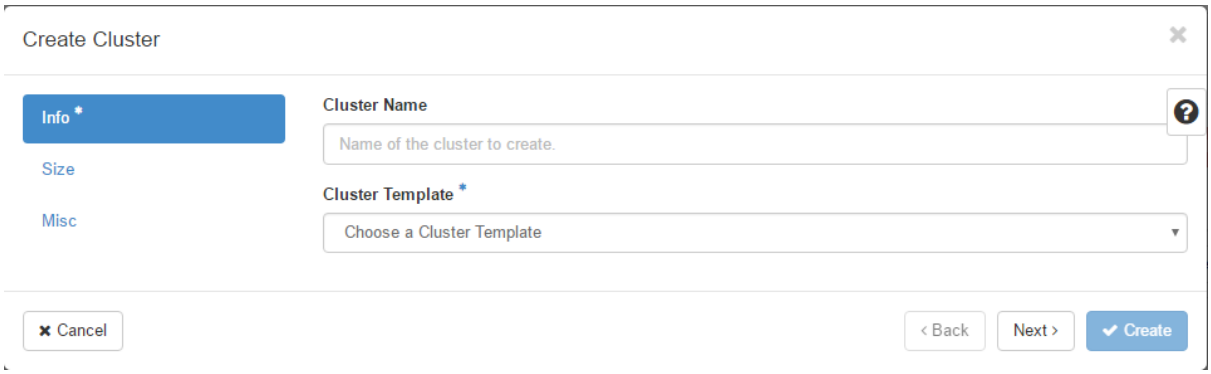
Following is the screenshot of the Horizon view showing the list of cluster templates.



Following is the screenshot of the Horizon view showing the details of a cluster template.



Following is the screenshot of the dialog to create a new cluster.



4.1.4 Cluster Drivers

A cluster driver is a collection of python code, heat templates, scripts, images, and documents for a particular COE on a particular distro. Magnum presents the concept of ClusterTemplates and clusters. The implementation for a particular cluster type is provided by the cluster driver. In other words, the cluster driver provisions and manages the infrastructure for the COE. Magnum includes default drivers for the following COE and distro pairs:

COE	distro
Kubernetes	Fedora CoreOS
Swarm	Fedora Atomic
Mesos	Ubuntu

Magnum is designed to accommodate new cluster drivers to support custom COEs and this section describes how a new cluster driver can be constructed and enabled in Magnum.

Directory structure

Magnum expects the components to be organized in the following directory structure under the directory drivers:

```
COE_Distro/
   image/
   templates/
   api.py
   driver.py
   monitor.py
   scale.py
   template_def.py
   version.py
```

The minimum required components are:

driver.py Python code that implements the controller operations for the particular COE. The driver must implement: Currently supported: `cluster_create`, `cluster_update`, `cluster_delete`.

templates A directory of orchestration templates for managing the lifecycle of clusters, including creation, configuration, update, and deletion. Currently only Heat templates are supported, but in the future other orchestration mechanism such as Ansible may be supported.

template_def.py Python code that maps the parameters from the ClusterTemplate to the input parameters for the orchestration and invokes the orchestration in the templates directory.

version.py Tracks the latest version of the driver in this directory. This is defined by a `version` attribute and is represented in the form of `1.0.0`. It should also include a `Driver` attribute with descriptive name such as `fedora_swarm_atomic`.

The remaining components are optional:

image Instructions for obtaining or building an image suitable for the COE.

api.py Python code to interface with the COE.

monitor.py Python code to monitor the resource utilization of the cluster.

scale.py Python code to scale the cluster by adding or removing nodes.

Sample cluster driver

To help developers in creating new COE drivers, a minimal cluster driver is provided as an example. The docker cluster driver will simply deploy a single VM running Ubuntu with the latest Docker version installed. It is not a true cluster, but the simplicity will help to illustrate the key concepts.

To be filled in

Installing a cluster driver

To be filled in

4.1.5 Cluster Type Definition

There are three key pieces to a Cluster Type Definition:

1. Heat Stack template - The HOT file that Magnum will use to generate a cluster using a Heat Stack.
2. Template definition - Magnums interface for interacting with the Heat template.
3. Definition Entry Point - Used to advertise the available Cluster Types.

The Heat Stack Template

The Heat Stack Template is where most of the real work happens. The result of the Heat Stack Template should be a full Container Orchestration Environment.

The Template Definition

Template definitions are a mapping of Magnum object attributes and Heat template parameters, along with Magnum consumable template outputs. A Cluster Type Definition indicates which Cluster Types it can provide. Cluster Types are how Magnum determines which of the enabled Cluster Type Definitions it will use for a given cluster.

The Definition Entry Point

Entry points are a standard discovery and import mechanism for Python objects. Each Template Definition should have an Entry Point in the *magnum.template_definitions* group. This example exposes its Template Definition as *example_template = example_template:ExampleTemplate* in the *magnum.template_definitions* group.

Installing Cluster Templates

Because Cluster Type Definitions are basically Python projects, they can be worked with like any other Python project. They can be cloned from version control and installed or uploaded to a package index and installed via utilities such as pip.

Enabling a Cluster Type is as simple as adding its Entry Point to the *enabled_definitions* config option in magnum.conf.:

```
# Setup python environment and install Magnum

$ virtualenv .venv
$ . .venv/bin/active
(.venv)$ git clone https://opendev.org/openstack/magnum
(.venv)$ cd magnum
(.venv)$ python setup.py install
```

(continues on next page)

(continued from previous page)

```
# List installed templates, notice default templates are enabled

(.venv)$ magnum-template-manage list-templates
Enabled Templates
  magnum_vm_atomic_k8s: /home/example/.venv/local/lib/python2.7/site-packages/
↪magnum/templates/kubernetes/kubecluster.yaml
  magnum_vm_coreos_k8s: /home/example/.venv/local/lib/python2.7/site-packages/
↪magnum/templates/kubernetes/kubecluster-coreos.yaml
Disabled Templates

# Install example template

(.venv)$ cd contrib/templates/example
(.venv)$ python setup.py install

# List installed templates, notice example template is disabled

(.venv)$ magnum-template-manage list-templates
Enabled Templates
  magnum_vm_atomic_k8s: /home/example/.venv/local/lib/python2.7/site-packages/
↪magnum/templates/kubernetes/kubecluster.yaml
  magnum_vm_coreos_k8s: /home/example/.venv/local/lib/python2.7/site-packages/
↪magnum/templates/kubernetes/kubecluster-coreos.yaml
Disabled Templates
  example_template: /home/example/.venv/local/lib/python2.7/site-packages/
↪ExampleTemplate-0.1-py2.7.egg/example_template/example.yaml

# Enable example template by setting enabled_definitions in magnum.conf

(.venv)$ sudo mkdir /etc/magnum
(.venv)$ sudo bash -c "cat > /etc/magnum/magnum.conf << END_CONF
[bay]
enabled_definitions=magnum_vm_atomic_k8s,magnum_vm_coreos_k8s,example_template
END_CONF"

# List installed templates, notice example template is now enabled

(.venv)$ magnum-template-manage list-templates
Enabled Templates
  example_template: /home/example/.venv/local/lib/python2.7/site-packages/
↪ExampleTemplate-0.1-py2.7.egg/example_template/example.yaml
  magnum_vm_atomic_k8s: /home/example/.venv/local/lib/python2.7/site-packages/
↪magnum/templates/kubernetes/kubecluster.yaml
  magnum_vm_coreos_k8s: /home/example/.venv/local/lib/python2.7/site-packages/
↪magnum/templates/kubernetes/kubecluster-coreos.yaml
Disabled Templates

# Use --details argument to get more details about each template
```

(continues on next page)

(continued from previous page)

```
(.venv)$ magnum-template-manage list-templates --details
Enabled Templates
  example_template: /home/example/.venv/local/lib/python2.7/site-packages/
↪ExampleTemplate-0.1-py2.7.egg/example_template/example.yaml
     Server_Type  OS       CoE
     vm         example  example_coe
  magnum_vm_atomic_k8s: /home/example/.venv/local/lib/python2.7/site-packages/
↪magnum/templates/kubernetes/kubecluster.yaml
     Server_Type   OS             CoE
     vm        fedora-atomic  kubernetes
  magnum_vm_coreos_k8s: /home/example/.venv/local/lib/python2.7/site-packages/
↪magnum/templates/kubernetes/kubecluster-coreos.yaml
     Server_Type  OS      CoE
     vm         coreos  kubernetes
Disabled Templates
```

4.1.6 Heat Stack Templates

Heat Stack Templates are what Magnum passes to Heat to generate a cluster. For each ClusterTemplate resource in Magnum, a Heat stack is created to arrange all of the cloud resources needed to support the container orchestration environment. These Heat stack templates provide a mapping of Magnum object attributes to Heat template parameters, along with Magnum consumable stack outputs. Magnum passes the Heat Stack Template to the Heat service to create a Heat stack. The result is a full Container Orchestration Environment.

4.1.7 Choosing a COE

Magnum supports a variety of COE options, and allows more to be added over time as they gain popularity. As an operator, you may choose to support the full variety of options, or you may want to offer a subset of the available choices. Given multiple choices, your users can run one or more clusters, and each may use a different COE. For example, I might have multiple clusters that use Kubernetes, and just one cluster that uses Swarm. All of these clusters can run concurrently, even though they use different COE software.

Choosing which COE to use depends on what tools you want to use to manage your containers once you start your app. If you want to use the Docker tools, you may want to use the Swarm cluster type. Swarm will spread your containers across the various nodes in your cluster automatically. It does not monitor the health of your containers, so it cant restart them for you if they stop. It will not automatically scale your app for you (as of Swarm version 1.2.2). You may view this as a plus. If you prefer to manage your application yourself, you might prefer swarm over the other COE options.

Kubernetes (as of v1.2) is more sophisticated than Swarm (as of v1.2.2). It offers an attractive YAML file description of a pod, which is a grouping of containers that run together as part of a distributed application. This file format allows you to model your application deployment using a declarative style. It has support for auto scaling and fault recovery, as well as features that allow for sophisticated software deployments, including canary deploys and blue/green deploys. Kubernetes is very popular, especially for web applications.

Apache Mesos is a COE that has been around longer than Kubernetes or Swarm. It allows for a variety of different frameworks to be used along with it, including Marathon, Aurora, Chronos, Hadoop, and a number of others.

The Apache Mesos framework design can be used to run alternate COE software directly on Mesos. Although this approach is not widely used yet, it may soon be possible to run Mesos with Kubernetes and Swarm as frameworks, allowing you to share the resources of a cluster between multiple different COEs. Until this option matures, we encourage Magnum users to create multiple clusters, and use the COE in each cluster that best fits the anticipated workload.

Finding the right COE for your workload is up to you, but Magnum offers you a choice to select among the prevailing leading options. Once you decide, see the next sections for examples of how to create a cluster with your desired COE.

4.1.8 Native Clients

Magnum preserves the native user experience with a COE and does not provide a separate API or client. This means you will need to use the native client for the particular cluster type to interface with the clusters. In the typical case, there are two clients to consider:

COE level This is the orchestration or management level such as Kubernetes, Swarm, Mesos and its frameworks.

Container level This is the low level container operation. Currently it is Docker for all clusters.

The clients can be CLI and/or browser-based. You will need to refer to the documentation for the specific native client and appropriate version for details, but following are some pointers for reference.

Kubernetes CLI is the tool kubectl, which can be simply copied from a node in the cluster or downloaded from the Kubernetes release. For instance, if the cluster is running Kubernetes release 1.2.0, the binary for kubectl can be downloaded as and set up locally as follows:

```
curl -O https://storage.googleapis.com/kubernetes-release/release/v1.2.0/bin/
↪linux/amd64/kubectl
chmod +x kubectl
sudo mv kubectl /usr/local/bin/kubectl
```

Kubernetes also provides a browser UI. If the cluster has the Kubernetes Dashboard running; it can be accessed using:

```
eval $(openstack coe cluster config <cluster-name>)
kubectl proxy

The browser can be accessed at http://localhost:8001/ui
```

For Swarm, the main CLI is docker, along with associated tools such as docker-compose, etc. Specific version of the binaries can be obtained from the Docker Engine installation.

Mesos cluster uses the Marathon framework and details on the Marathon UI can be found in the section *Using Marathon*.

Depending on the client requirement, you may need to use a version of the client that matches the version in the cluster. To determine the version of the COE and container, use the command cluster-show and look for the attribute *coe_version* and *container_version*:

```
openstack coe cluster show k8s-cluster
+--------------------+--------------------------------------------------------
↪----+
| Property           | Value                                                  ␣
↪    |
+--------------------+--------------------------------------------------------
↪----+
| status             | CREATE_COMPLETE                                        ␣
↪    |
| uuid               | 04952c60-a338-437f-a7e7-d016d1d00e65                   ␣
↪    |
| stack_id           | b7bf72ce-b08e-4768-8201-e63a99346898                   ␣
↪    |
| status_reason      | Stack CREATE completed successfully                    ␣
↪    |
| created_at         | 2016-07-25T23:14:06+00:00                              ␣
↪    |
| updated_at         | 2016-07-25T23:14:10+00:00                              ␣
↪    |
| create_timeout     | 60                                                     ␣
↪    |
| coe_version        | v1.2.0                                                 ␣
↪    |
| api_address        | https://192.168.19.86:6443                             ␣
↪    |
| cluster_template_id| da2825a0-6d09-4208-b39e-b2db666f1118                   ␣
↪    |
| master_addresses   | ['192.168.19.87']                                      ␣
↪    |
| node_count         | 1                                                      ␣
↪    |
| node_addresses     | ['192.168.19.88']                                      ␣
↪    |
| master_count       | 1                                                      ␣
↪    |
| container_version  | 1.9.1                                                  ␣
↪    |
| discovery_url      | https://discovery.etcd.io/
↪3b7fb09733429d16679484673ba3bfd5 |
| name               | k8s-cluster                                            ␣
↪    |
+--------------------+--------------------------------------------------------
↪----+
```

4.1.9 Kubernetes

Kubernetes uses a range of terminology that we refer to in this guide. We define these common terms in the *Glossary* for your reference.

When Magnum deploys a Kubernetes cluster, it uses parameters defined in the ClusterTemplate and specified on the cluster-create command, for example:

```
openstack coe cluster template create k8s-cluster-template \
                          --image fedora-coreos-latest \
                          --keypair testkey \
                          --external-network public \
                          --dns-nameserver 8.8.8.8 \
                          --flavor m1.small \
                          --docker-volume-size 5 \
                          --network-driver flannel \
                          --coe kubernetes

openstack coe cluster create k8s-cluster \
                      --cluster-template k8s-cluster-template \
                      --master-count 3 \
                      --node-count 8
```

Refer to the *ClusterTemplate* and *Cluster* sections for the full list of parameters. Following are further details relevant to a Kubernetes cluster:

Number of masters (master-count) Specified in the cluster-create command to indicate how many servers will run as master in the cluster. Having more than one will provide high availability. The masters will be in a load balancer pool and the virtual IP address (VIP) of the load balancer will serve as the Kubernetes API endpoint. For external access, a floating IP associated with this VIP is available and this is the endpoint shown for Kubernetes in the cluster-show command.

Number of nodes (node-count) Specified in the cluster-create command to indicate how many servers will run as node in the cluster to host the users pods. The nodes are registered in Kubernetes using the Nova instance name.

Network driver (network-driver) Specified in the ClusterTemplate to select the network driver. The supported and default network driver is flannel, an overlay network providing a flat network for all pods. Refer to the *Networking* section for more details.

Volume driver (volume-driver) Specified in the ClusterTemplate to select the volume driver. The supported volume driver is cinder, allowing Cinder volumes to be mounted in containers for use as persistent storage. Data written to these volumes will persist after the container exits and can be accessed again from other containers, while data written to the union file system hosting the container will be deleted. Refer to the *Storage* section for more details.

Storage driver (docker-storage-driver) Specified in the ClusterTemplate to select the Docker storage driver. The default is devicemapper. Refer to the *Storage* section for more details.

NOTE: For Fedora CoreOS driver, devicemapper is not supported.

Image (image) Specified in the ClusterTemplate to indicate the image to boot the servers. The image binary is loaded in Glance with the attribute os_distro = fedora-coreos. Current supported images is Fedora CoreOS (download from Fedora CoreOS)

TLS (tls-disabled) Transport Layer Security is enabled by default, so you need a key and signed certificate to access the Kubernetes API and CLI. Magnum handles its own key and certificate when interfacing with the Kubernetes cluster. In development mode, TLS can be disabled. Refer to the Transport Layer Security_ section for more details.

What runs on the servers The servers for Kubernetes master host containers in the kube-system name space to run the Kubernetes proxy, scheduler and controller manager. The masters will not host users pods. Kubernetes API server, docker daemon, etcd and flannel run as systemd services. The servers for Kubernetes node also host a container in the kube-system name space to run the Kubernetes proxy, while Kubernetes kubelet, docker daemon and flannel run as systemd services.

Log into the servers You can log into the master servers using the login fedora and the keypair specified in the ClusterTemplate.

In addition to the common attributes in the ClusterTemplate, you can specify the following attributes that are specific to Kubernetes by using the labels attribute.

admission_control_list This label corresponds to Kubernetes parameter for the API server admission-control. For more details, refer to the Admission Controllers. The default value corresponds to the one recommended in this doc for our current Kubernetes version.

boot_volume_size This label overrides the default_boot_volume_size of instances which is useful if your flavors are boot from volume only. The default value is 0, meaning that cluster instances will not boot from volume.

boot_volume_type This label overrides the default_boot_volume_type of instances which is useful if your flavors are boot from volume only. The default value is , meaning that Magnum will randomly select a Cinder volume type from all available options.

etcd_volume_size This label sets the size of a volume holding the etcd storage data. The default value is 0, meaning the etcd data is not persisted (no volume).

etcd_volume_type This label overrides the default_etcd_volume_type holding the etcd storage data. The default value is , meaning that Magnum will randomly select a Cinder volume type from all available options.

container_infra_prefix Prefix of all container images used in the cluster (kubernetes components, coredns, kubernetes-dashboard, node-exporter). For example, kubernetes-apiserver is pulled from docker.io/openstackmagnum/kubernetes-apiserver, with this label it can be changed to myregistry.example.com/mycloud/kubernetes-apiserver. Similarly, all other components used in the cluster will be prefixed with this label, which assumes an operator has cloned all expected images in myregistry.example.com/mycloud.

Images that must be mirrored:

- docker.io/coredns/coredns:1.3.1
- quay.io/coreos/etcd:v3.4.6
- docker.io/k8scloudprovider/k8s-keystone-auth:v1.18.0
- docker.io/k8scloudprovider/openstack-cloud-controller-manager:v1.18.0
- gcr.io/google_containers/pause:3.1

Images that might be needed when use_podman is false:

- docker.io/openstackmagnum/kubernetes-apiserver
- docker.io/openstackmagnum/kubernetes-controller-manager

- docker.io/openstackmagnum/kubernetes-kubelet
- docker.io/openstackmagnum/kubernetes-proxy
- docker.io/openstackmagnum/kubernetes-scheduler

Images that might be needed:

- k8s.gcr.io/hyperkube:v1.18.2
- docker.io/grafana/grafana:5.1.5
- docker.io/prom/node-exporter:latest
- docker.io/prom/prometheus:latest
- docker.io/traefik:v1.7.28
- gcr.io/google_containers/kubernetes-dashboard-amd64:v1.5.1
- gcr.io/google_containers/metrics-server-amd64:v0.3.6
- k8s.gcr.io/node-problem-detector:v0.6.2
- docker.io/planetlabs/draino:abf028a
- docker.io/openstackmagnum/cluster-autoscaler:v1.18.1
- quay.io/calico/cni:v3.13.1
- quay.io/calico/pod2daemon-flexvol:v3.13.1
- quay.io/calico/kube-controllers:v3.13.1
- quay.io/calico/node:v3.13.1
- quay.io/coreos/flannel-cni:v0.3.0
- quay.io/coreos/flannel:v0.12.0-amd64

Images that might be needed if monitoring_enabled is true:

- quay.io/prometheus/alertmanager:v0.20.0
- docker.io/squareup/ghostunnel:v1.5.2
- docker.io/jettech/kube-webhook-certgen:v1.0.0
- quay.io/coreos/prometheus-operator:v0.37.0
- quay.io/coreos/configmap-reload:v0.0.1
- quay.io/coreos/prometheus-config-reloader:v0.37.0
- quay.io/prometheus/prometheus:v2.15.2

Images that might be needed if cinder_csi_enabled is true:

- docker.io/k8scloudprovider/cinder-csi-plugin:v1.18.0
- quay.io/k8scsi/csi-attacher:v2.0.0
- quay.io/k8scsi/csi-provisioner:v1.4.0
- quay.io/k8scsi/csi-snapshotter:v1.2.2
- quay.io/k8scsi/csi-resizer:v0.3.0

- quay.io/k8scsi/csi-node-driver-registrar:v1.1.0

hyperkube_prefix This label allows users to specify a custom prefix for Hyperkube container source since official Hyperkube images have been discontinued for *kube_tag* greater than 1.18.x. If you wish you use 1.19.x onwards, you may want to use unofficial sources like *docker.io/rancher/*, *ghcr.io/openstackmagnum/* or your own container registry. If *container_infra_prefix* label is defined, it still takes precedence over this label. Default: docker.io/rancher/

kube_tag This label allows users to select a specific Kubernetes release based on its container tag for Fedora CoreOS image. If unset, the current Magnum versions default Kubernetes release is installed. Take a look at the Wiki for a compatibility matrix between Kubernetes and Magnum Releases. Stein default: v1.11.6 Train default: v1.15.7 Ussuri default: v1.18.2 Victoria default: v1.18.16 Yoga default: v1.23.3-rancher1

heapster_enabled heapster_enabled is used to enable disable the installation of heapster. Ussuri default: false Train default: true

cloud_provider_tag This label allows users to override the default openstack-cloud-controller-manager container image tag. Refer to openstack-cloud-controller-manager page for available tags. Stein default: v0.2.0 Train default: v1.15.0 Ussuri default: v1.18.0

etcd_tag This label allows users to select a specific etcd version, based on its container tag. If unset, the current Magnum versions a default etcd version. Stein default: v3.2.7 Train default: 3.2.26 Ussuri default: v3.4.6

coredns_tag This label allows users to select a specific coredns version, based on its container tag. If unset, the current Magnum versions a default etcd version. Stein default: 1.3.1 Train default: 1.3.1 Ussuri default: 1.6.6

flannel_tag This label allows users to select a specific flannel version, based on its container tag: Queens Rocky <https://quay.io/repository/coreos/flannel?tab=tags>‘_ If unset, the default version will be used. Stein default: v0.10.0-amd64 Train default: v0.11.0-amd64 Ussuri default: v0.12.0-amd64

flannel_cni_tag This label allows users to select a specific flannel_cni version, based on its container tag. This container adds the cni plugins in the host under /opt/cni/bin. If unset, the current Magnum versions a default flannel version. Stein default: v0.3.0 Train default: v0.3.0 Ussuri default: v0.3.0

heat_container_agent_tag This label allows users to select a specific heat_container_agent version, based on its container tag. Train-default: train-stable-3 Ussuri-default: ussuri-stable-1 Victoria-default: victoria-stable-1 Wallaby-default: wallaby-stable-1

kube_dashboard_enabled This label triggers the deployment of the kubernetes dashboard. The default value is 1, meaning it will be enabled.

cert_manager_api This label enables the kubernetes certificate manager api.

kubelet_options This label can hold any additional options to be passed to the kubelet. For more details, refer to the kubelet admin guide. By default no additional options are passed.

kubeproxy_options This label can hold any additional options to be passed to the kube proxy. For more details, refer to the kube proxy admin guide. By default no additional options are passed.

kubecontroller_options This label can hold any additional options to be passed to the kube controller manager. For more details, refer to the kube controller manager admin guide. By default no additional options are passed.

kubeapi_options This label can hold any additional options to be passed to the kube api server. For more details, refer to the kube api admin guide. By default no additional options are passed.

kubescheduler_options This label can hold any additional options to be passed to the kube scheduler. For more details, refer to the kube scheduler admin guide. By default no additional options are passed.

influx_grafana_dashboard_enabled The kubernetes dashboard comes with heapster enabled. If this label is set, an influxdb and grafana instance will be deployed, heapster will push data to influx and grafana will project them.

cgroup_driver This label tells kubelet which Cgroup driver to use. Ideally this should be identical to the Cgroup driver that Docker has been started with.

cloud_provider_enabled Add cloud_provider_enabled label for the k8s_fedora_atomic driver. Defaults to the value of cluster_user_trust (default: false unless explicitly set to true in magnum.conf due to CVE-2016-7404). Consequently, cloud_provider_enabled label cannot be overridden to true when cluster_user_trust resolves to false. For specific kubernetes versions, if cinder is selected as a volume_driver, it is implied that the cloud provider will be enabled since they are combined.

cinder_csi_enabled When true, out-of-tree Cinder CSI driver will be enabled. Requires cinder to be selected as a volume_driver and consequently also requires label cloud_provider_enabled to be true (see cloud_provider_enabled section). Ussuri default: false Victoria default: true

cinder_csi_plugin_tag This label allows users to override the default cinder-csi-plugin container image tag. Refer to cinder-csi-plugin page for available tags. Train default: v1.16.0 Ussuri default: v1.18.0

csi_attacher_tag This label allows users to override the default container tag for CSI attacher. For additional tags, refer to CSI attacher page. Ussuri-default: v2.0.0

csi_provisioner_tag This label allows users to override the default container tag for CSI provisioner. For additional tags, refer to CSI provisioner page. Ussuri-default: v1.4.0

csi_snapshotter_tag This label allows users to override the default container tag for CSI snapshotter. For additional tags, refer to CSI snapshotter page. Ussuri-default: v1.2.2

csi_resizer_tag This label allows users to override the default container tag for CSI resizer. For additional tags, refer to CSI resizer page. Ussuri-default: v0.3.0

csi_node_driver_registrar_tag This label allows users to override the default container tag for CSI node driver registrar. For additional tags, refer to CSI node driver registrar page. Ussuri-default: v1.1.0

keystone_auth_enabled If this label is set to True, Kubernetes will support use Keystone for authorization and authentication.

k8s_keystone_auth_tag This label allows users to override the default k8s-keystone-auth container image tag. Refer to k8s-keystone-auth page for available tags. Stein default: v1.13.0 Train default: v1.14.0 Ussuri default: v1.18.0

tiller_enabled If set to true, tiller will be deployed in the kube-system namespace. Ussuri default: false Train default: false

tiller_tag This label allows users to override the default container tag for Tiller. For additional tags, refer to Tiller page and look for tags<v3.0.0. Train default: v2.12.3 Ussuri default: v2.16.7

tiller_namespace The namespace in which Tiller and Helm v2 chart install jobs are installed. Default: magnum-tiller

helm_client_url URL of the helm client binary. Default:

helm_client_sha256 SHA256 checksum of the helm client binary. Ussuri default: 018f9908cb950701a5d59e757653a790c66d8eda288625dbb185354ca6f41f6b

helm_client_tag This label allows users to override the default container tag for Helm client. For additional tags, refer to Helm client page. You must use identical tiller_tag if you wish to use Tiller (for helm_client_tag<v3.0.0). Ussuri default: v3.2.1

master_lb_floating_ip_enabled Controls if Magnum allocates floating IP for the load balancer of master nodes. This label only takes effect when the template property `master_lb_enabled` is set. If not specified, the default value is the same as template property `floating_ip_enabled`.

master_lb_allowed_cidrs A CIDR list which can be used to control the access for the load balancer of master nodes. The input format is comma delimited list. For example, 192.168.0.0/16,10.0.0.0/24. Default: (which opens to 0.0.0.0/0)

auto_healing_enabled If set to true, auto healing feature will be enabled. Defaults to false.

auto_healing_controller This label sets the auto-healing service to be used. Currently `draino` and `magnum-auto-healer` are supported. The default is `draino`. For more details, see draino doc and magnum-auto-healer doc.

draino_tag This label allows users to select a specific Draino version.

magnum_auto_healer_tag This label allows users to override the default magnum-auto-healer container image tag. Refer to magnum-auto-healer page for available tags. Stein default: v1.15.0 Train default: v1.15.0 Ussuri default: v1.18.0

auto_scaling_enabled If set to true, auto scaling feature will be enabled. Default: false.

autoscaler_tag This label allows users to override the default cluster-autoscaler container image tag. Refer to cluster-autoscaler page for available tags. Stein default: v1.0 Train default: v1.0 Ussuri default: v1.18.1

npd_enabled Set Node Problem Detector service enabled or disabled. Default: true

node_problem_detector_tag This label allows users to select a specific Node Problem Detector version.

min_node_count The minmium node count of the cluster when doing auto scaling or auto healing. Default: 1

max_node_count The maxmium node count of the cluster when doing auto scaling or auto healing.

use_podman Choose whether system containers etcd, kubernetes and the heat-agent will be installed with podman or atomic. This label is relevant for k8s_fedora drivers.

k8s_fedora_atomic_v1 defaults to use_podman=false, meaning atomic will be used pulling containers from docker.io/openstackmagnum. use_podman=true is accepted as well, which will pull containers by k8s.gcr.io.

k8s_fedora_coreos_v1 defaults and accepts only use_podman=true.

Note that, to use kubernetes version greater or equal to v1.16.0 with the k8s_fedora_atomic_v1 driver, you need to set use_podman=true. This is necessary since v1.16 dropped the containerized flag in kubelet. https://github.com/kubernetes/kubernetes/pull/80043/files

selinux_mode Choose SELinux mode between enforcing, permissive and disabled. This label is currently only relevant for k8s_fedora drivers.

k8s_fedora_atomic_v1 driver defaults to selinux_mode=permissive because this was the only way atomic containers were able to start Kubernetes services. On the other hand, if the opt-in use_podman=true label is supplied, selinux_mode=enforcing is supported. Note that if

selinux_mode=disabled is chosen, this only takes full effect once the instances are manually rebooted but they will be set to permissive mode in the meantime.

k8s_fedora_coreos_v1 driver defaults to selinux_mode=enforcing.

container_runtime The container runtime to use. Empty value means, use docker from the host. Since ussuri, apart from empty (host-docker), containerd is also an option.

containerd_version The containerd version to use as released in https://github.com/containerd/containerd/releases and https://storage.googleapis.com/cri-containerd-release/ Victoria default: 1.4.4 Ussuri default: 1.2.8

containerd_tarball_url Url with the tarball of containerds binaries.

containerd_tarball_sha256 sha256 of the tarball fetched with containerd_tarball_url or from https://github.com/containerd/containerd/releases.

kube_dashboard_version Default version of Kubernetes dashboard. Train default: v1.8.3 Ussuri default: v2.0.0

metrics_scraper_tag The version of metrics-scraper used by kubernetes dashboard. Ussuri default: v1.0.4

fixed_subnet_cidr CIDR of the fixed subnet created by Magnum when a user has not specified an existing fixed_subnet during cluster creation. Ussuri default: 10.0.0.0/24

External load balancer for services

All Kubernetes pods and services created in the cluster are assigned IP addresses on a private container network so they can access each other and the external internet. However, these IP addresses are not accessible from an external network.

To publish a service endpoint externally so that the service can be accessed from the external network, Kubernetes provides the external load balancer feature. This is done by simply specifying in the service manifest the attribute type: LoadBalancer. Magnum enables and configures the Kubernetes plugin for OpenStack so that it can interface with Neutron and manage the necessary networking resources.

When the service is created, Kubernetes will add an external load balancer in front of the service so that the service will have an external IP address in addition to the internal IP address on the container network. The service endpoint can then be accessed with this external IP address. Kubernetes handles all the life cycle operations when pods are modified behind the service and when the service is deleted.

Refer to the *Kubernetes External Load Balancer* section for more details.

Ingress Controller

In addition to the LoadBalancer described above, Kubernetes can also be configured with an Ingress Controller. Ingress can provide load balancing, SSL termination and name-based virtual hosting.

Magnum allows selecting one of multiple controller options via the ingress_controller label. Check the Kubernetes documentation to define your own Ingress resources.

Traefik: Traefiks pods by default expose port 80 and 443 for http(s) traffic on the nodes they are running. In kubernetes cluster, these ports are closed by default. Cluster administrator needs to add a rule in the worker nodes security group. For example:

```
openstack security group rule create <SECURITY_GROUP> \
  --protocol tcp \
  --dst-port 80:80
openstack security group rule create <SECURITY_GROUP> \
  --protocol tcp \
  --dst-port 443:443
```

ingress_controller This label sets the Ingress Controller to be used. Currently traefik, nginx and octavia are supported. The default is , meaning no Ingress Controller is configured. For more details about octavia-ingress-controller please refer to cloud-provider-openstack document To use nginx ingress controller, tiller_enabled must be true when using helm_client_tag<v3.0.0.

ingress_controller_role This label defines the role nodes should have to run an instance of the Ingress Controller. This gives operators full control on which nodes should be running an instance of the controller, and should be set in multiple nodes for availability. Default is ingress. An example of setting this in a Kubernetes node would be:

```
kubectl label node <node-name> role=ingress
```

This label is not used for octavia-ingress-controller.

octavia_ingress_controller_tag The image tag for octavia-ingress-controller. Train-default: v1.15.0

nginx_ingress_controller_tag The image tag for nginx-ingress-controller. Stein-default: 0.23.0 Train-default: 0.26.1 Ussuru-default: 0.26.1 Victoria-default: 0.32.0

nginx_ingress_controller_chart_tag The chart version for nginx-ingress-controller. Train-default: v1.24.7 Ussuru-default: v1.24.7 Victoria-default: v1.36.3

traefik_ingress_controller_tag The image tag for traefik_ingress_controller_tag. Stein-default: v1.7.10

DNS

CoreDNS is a critical service in Kubernetes cluster for service discovery. To get high availability for CoreDNS pod for Kubernetes cluster, now Magnum supports the autoscaling of CoreDNS using cluster-proportional-autoscaler. With cluster-proportional-autoscaler, the replicas of CoreDNS pod will be autoscaled based on the nodes and cores in the clsuter to prevent single point failure.

The scaling parameters and data points are provided via a ConfigMap to the autoscaler and it refreshes its parameters table every poll interval to be up to date with the latest desired scaling parameters. Using ConfigMap means user can do on-the-fly changes(including control mode) without rebuilding or restarting the scaler containers/pods. Please refer Autoscale the DNS Service in a Cluster for more info.

Keystone authN and authZ

Now cloud-provider-openstack provides a good webhook between OpenStack Keystone and Kubernetes, so that user can do authorization and authentication with a Keystone user/role against the Kubernetes cluster. If label *keystone-auth-enabled* is set True, then user can use their OpenStack credentials and roles to access resources in Kubernetes.

Assume you have already got the configs with command *eval $(openstack coe cluster config <cluster ID>)*, then to configure the kubectl client, the following commands are needed:

1. Run *kubectl config set-credentials openstackuser auth-provider=openstack*
2. Run *kubectl config set-context cluster=<your cluster name> user=openstackuser openstackuser@kubernetes*
3. Run *kubectl config use-context openstackuser@kubernetes* to activate the context

NOTE: Please make sure the version of kubectl is 1.8+ and make sure OS_DOMAIN_NAME is included in the rc file.

Now try *kubectl get pods*, you should be able to see response from Kubernetes based on current users role.

Please refer the doc of k8s-keystone-auth in cloud-provider-openstack for more information.

4.1.10 Swarm

A Swarm cluster is a pool of servers running Docker daemon that is managed as a single Docker host. One or more Swarm managers accepts the standard Docker API and manage this pool of servers. Magnum deploys a Swarm cluster using parameters defined in the ClusterTemplate and specified on the cluster-create command, for example:

```
openstack coe cluster template create swarm-cluster-template \
                          --image fedora-atomic-latest \
                          --keypair testkey \
                          --external-network public \
                          --dns-nameserver 8.8.8.8 \
                          --flavor m1.small \
                          --docker-volume-size 5 \
                          --coe swarm

openstack coe cluster create swarm-cluster \
                  --cluster-template swarm-cluster-template \
                  --master-count 3 \
                  --node-count 8
```

Refer to the *ClusterTemplate* and *Cluster* sections for the full list of parameters. Following are further details relevant to Swarm:

What runs on the servers There are two types of servers in the Swarm cluster: managers and nodes. The Docker daemon runs on all servers. On the servers for manager, the Swarm manager is run as a Docker container on port 2376 and this is initiated by the systemd service swarm-manager. Etcd is also run on the manager servers for discovery of the node servers in the cluster. On the servers for node, the Swarm agent is run as a Docker container on port 2375 and this is initiated

by the systemd service swarm-agent. On start up, the agents will register themselves in etcd and the managers will discover the new node to manage.

Number of managers (master-count) Specified in the cluster-create command to indicate how many servers will run as managers in the cluster. Having more than one will provide high availability. The managers will be in a load balancer pool and the load balancer virtual IP address (VIP) will serve as the Swarm API endpoint. A floating IP associated with the load balancer VIP will serve as the external Swarm API endpoint. The managers accept the standard Docker API and perform the corresponding operation on the servers in the pool. For instance, when a new container is created, the managers will select one of the servers based on some strategy and schedule the containers there.

Number of nodes (node-count) Specified in the cluster-create command to indicate how many servers will run as nodes in the cluster to host your Docker containers. These servers will register themselves in etcd for discovery by the managers, and interact with the managers. Docker daemon is run locally to host containers from users.

Network driver (network-driver) Specified in the ClusterTemplate to select the network driver. The supported drivers are docker and flannel, with docker as the default. With the docker driver, containers are connected to the docker0 bridge on each node and are assigned local IP address. With the flannel driver, containers are connected to a flat overlay network and are assigned IP address by Flannel. Refer to the *Networking* section for more details.

Volume driver (volume-driver) Specified in the ClusterTemplate to select the volume driver to provide persistent storage for containers. The supported volume driver is rexray. The default is no volume driver. When rexray or other volume driver is deployed, you can use the Docker volume command to create, mount, unmount, delete volumes in containers. Cinder block storage is used as the backend to support this feature. Refer to the *Storage* section for more details.

Storage driver (docker-storage-driver) Specified in the ClusterTemplate to select the Docker storage driver. The default is devicemapper. Refer to the *Storage* section for more details.

Image (image) Specified in the ClusterTemplate to indicate the image to boot the servers for the Swarm manager and node. The image binary is loaded in Glance with the attribute os_distro = fedora-atomic. Current supported image is Fedora Atomic (download from Fedora)

TLS (tls-disabled) Transport Layer Security is enabled by default to secure the Swarm API for access by both the users and Magnum. You will need a key and a signed certificate to access the Swarm API and CLI. Magnum handles its own key and certificate when interfacing with the Swarm cluster. In development mode, TLS can be disabled. Refer to the Transport Layer Security_ section for details on how to create your key and have Magnum sign your certificate.

Log into the servers You can log into the manager and node servers with the account fedora and the keypair specified in the ClusterTemplate.

In addition to the common attributes in the ClusterTemplate, you can specify the following attributes that are specific to Swarm by using the labels attribute.

swarm_strategy This label corresponds to Swarm parameter for master strategy. For more details, refer to the Swarm Strategy. Valid values for this label are:

- spread
- binpack
- random

4.1.11 Mesos

A Mesos cluster consists of a pool of servers running as Mesos slaves, managed by a set of servers running as Mesos masters. Mesos manages the resources from the slaves but does not itself deploy containers. Instead, one of more Mesos frameworks running on the Mesos cluster would accept user requests on their own endpoint, using their particular API. These frameworks would then negotiate the resources with Mesos and the containers are deployed on the servers where the resources are offered.

Magnum deploys a Mesos cluster using parameters defined in the ClusterTemplate and specified on the cluster-create command, for example:

```
openstack coe cluster template create mesos-cluster-template \
                        --image ubuntu-mesos \
                        --keypair testkey \
                        --external-network public \
                        --dns-nameserver 8.8.8.8 \
                        --flavor m1.small \
                        --coe mesos

openstack coe cluster create mesos-cluster \
                    --cluster-template mesos-cluster-template \
                    --master-count 3 \
                    --node-count 8
```

Refer to the *ClusterTemplate* and *Cluster* sections for the full list of parameters. Following are further details relevant to Mesos:

What runs on the servers There are two types of servers in the Mesos cluster: masters and slaves. The Docker daemon runs on all servers. On the servers for master, the Mesos master is run as a process on port 5050 and this is initiated by the upstart service mesos-master. Zookeeper is also run on the master servers, initiated by the upstart service zookeeper. Zookeeper is used by the master servers for electing the leader among the masters, and by the slave servers and frameworks to determine the current leader. The framework Marathon is run as a process on port 8080 on the master servers, initiated by the upstart service marathon. On the servers for slave, the Mesos slave is run as a process initiated by the upstart service mesos-slave.

Number of master (master-count) Specified in the cluster-create command to indicate how many servers will run as masters in the cluster. Having more than one will provide high availability. If the load balancer option is specified, the masters will be in a load balancer pool and the load balancer virtual IP address (VIP) will serve as the Mesos API endpoint. A floating IP associated with the load balancer VIP will serve as the external Mesos API endpoint.

Number of agents (node-count) Specified in the cluster-create command to indicate how many servers will run as Mesos slave in the cluster. Docker daemon is run locally to host containers from users. The slaves report their available resources to the master and accept request from the master to deploy tasks from the frameworks. In this case, the tasks will be to run Docker containers.

Network driver (network-driver) Specified in the ClusterTemplate to select the network driver. Currently docker is the only supported driver: containers are connected to the docker0 bridge on each node and are assigned local IP address. Refer to the *Networking* section for more details.

Volume driver (volume-driver) Specified in the ClusterTemplate to select the volume driver to provide persistent storage for containers. The supported volume driver is rexray. The default is no volume driver. When rexray or other volume driver is deployed, you can use the Docker volume command

to create, mount, unmount, delete volumes in containers. Cinder block storage is used as the backend to support this feature. Refer to the *Storage* section for more details.

Storage driver (docker-storage-driver) This is currently not supported for Mesos.

Image (image) Specified in the ClusterTemplate to indicate the image to boot the servers for the Mesos master and slave. The image binary is loaded in Glance with the attribute os_distro = ubuntu. You can download the ready-built image, or you can create the image as described below in the *Building Mesos image* section.

TLS (tls-disabled) Transport Layer Security is currently not implemented yet for Mesos.

Log into the servers You can log into the manager and node servers with the account ubuntu and the keypair specified in the ClusterTemplate.

In addition to the common attributes in the baymodel, you can specify the following attributes that are specific to Mesos by using the labels attribute.

rexray_preempt When the volume driver rexray is used, you can mount a data volume backed by Cinder to a host to be accessed by a container. In this case, the label rexray_preempt can optionally be set to True or False to enable any host to take control of the volume regardless of whether other hosts are using the volume. This will in effect unmount the volume from the current host and remount it on the new host. If this label is set to false, then rexray will ensure data safety for locking the volume before remounting. The default value is False.

mesos_slave_isolation This label corresponds to the Mesos parameter for slave isolation. The isolators are needed to provide proper isolation according to the runtime configurations specified in the container image. For more details, refer to the Mesos configuration and the Mesos container image support. Valid values for this label are:

- filesystem/posix
- filesystem/linux
- filesystem/shared
- posix/cpu
- posix/mem
- posix/disk
- cgroups/cpu
- cgroups/mem
- docker/runtime
- namespaces/pid

mesos_slave_image_providers This label corresponds to the Mesos parameter for agent image_providers, which tells Mesos containerizer what types of container images are allowed. For more details, refer to the Mesos configuration and the Mesos container image support. Valid values are:

- appc
- docker
- appc,docker

mesos_slave_work_dir This label corresponds to the Mesos parameter work_dir for slave. For more details, refer to the Mesos configuration. Valid value is a directory path to use as the work directory for the framework, for example:

```
mesos_slave_work_dir=/tmp/mesos
```

mesos_slave_executor_env_variables This label corresponds to the Mesos parameter for slave executor_environment_variables, which passes additional environment variables to the executor and subsequent tasks. For more details, refer to the Mesos configuration. Valid value is the name of a JSON file, for example:

```
mesos_slave_executor_env_variables=/home/ubuntu/test.json
```

The JSON file should contain environment variables, for example:

```
{
   "PATH": "/bin:/usr/bin",
   "LD_LIBRARY_PATH": "/usr/local/lib"
}
```

By default the executor will inherit the slaves environment variables.

Building Mesos image

The boot image for Mesos cluster is an Ubuntu 14.04 base image with the following middleware pre-installed:

- docker
- zookeeper
- mesos
- marathon

The cluster driver provides two ways to create this image, as follows.

Diskimage-builder

To run the diskimage-builder tool manually, use the provided elements. Following are the typical steps to use the diskimage-builder tool on an Ubuntu server:

```
$ sudo apt-get update
$ sudo apt-get install git qemu-utils python-pip
$ sudo pip install diskimage-builder

$ git clone https://opendev.org/openstack/magnum
$ git clone https://opendev.org/openstack/dib-utils.git
$ git clone https://opendev.org/openstack/tripleo-image-elements.git
$ git clone https://opendev.org/openstack/heat-templates.git
$ export PATH="${PWD}/dib-utils/bin:$PATH"
$ export ELEMENTS_PATH=tripleo-image-elements/elements:heat-templates/hot/
↪software-config/elements:magnum/magnum/drivers/mesos_ubuntu_v1/image/mesos
```

(continues on next page)

(continued from previous page)

```
$ export DIB_RELEASE=trusty

$ disk-image-create ubuntu vm docker mesos \
    os-collect-config os-refresh-config os-apply-config \
    heat-config heat-config-script \
    -o ubuntu-mesos.qcow2
```

Dockerfile

To build the image as above but within a Docker container, use the provided Dockerfile. The output image will be saved as /tmp/ubuntu-mesos.qcow2. Following are the typical steps to run a Docker container to build the image:

```
$ git clone https://opendev.org/openstack/magnum
$ cd magnum/magnum/drivers/mesos_ubuntu_v1/image
$ sudo docker build -t magnum/mesos-builder .
$ sudo docker run -v /tmp:/output --rm -ti --privileged magnum/mesos-builder
...
Image file /output/ubuntu-mesos.qcow2 created...
```

Using Marathon

Marathon is a Mesos framework for long running applications. Docker containers can be deployed via Marathons REST API. To get the endpoint for Marathon, run the cluster-show command and look for the property api_address. Marathons endpoint is port 8080 on this IP address, so the web console can be accessed at:

```
http://<api_address>:8080/
```

Refer to Marathon documentation for details on running applications. For example, you can post a JSON app description to `http://<api_address>:8080/apps` to deploy a Docker container:

```
$ cat > app.json << END
{
  "container": {
    "type": "DOCKER",
    "docker": {
      "image": "libmesos/ubuntu"
    }
  },
  "id": "ubuntu",
  "instances": 1,
  "cpus": 0.5,
  "mem": 512,
  "uris": [],
  "cmd": "while sleep 10; do date -u +%T; done"
}
```

(continues on next page)

(continued from previous page)

```
END
$ API_ADDRESS=$(openstack coe cluster show mesos-cluster | awk '/ api_address␣
↪/{print $4}')
$ curl -X POST -H "Content-Type: application/json" \
    http://${API_ADDRESS}:8080/v2/apps -d@app.json
```

4.1.12 Transport Layer Security

Magnum uses TLS to secure communication between a clusters services and the outside world. TLS is a complex subject, and many guides on it exist already. This guide will not attempt to fully describe TLS, but instead will only cover the necessary steps to get a client set up to talk to a cluster with TLS. A more in-depth guide on TLS can be found in the OpenSSL Cookbook by Ivan Risti.

TLS is employed at 3 points in a cluster:

1. By Magnum to communicate with the cluster API endpoint
2. By the cluster worker nodes to communicate with the master nodes
3. By the end-user when they use the native client libraries to interact with the cluster. This applies to both a CLI or a program that uses a client for the particular cluster. Each client needs a valid certificate to authenticate and communicate with a cluster.

The first two cases are implemented internally by Magnum and are not exposed to the users, while the last case involves the users and is described in more details below.

Deploying a secure cluster

Current TLS support is summarized below:

COE	TLS support
Kubernetes	yes
Swarm	yes
Mesos	no

For cluster type with TLS support, e.g. Kubernetes and Swarm, TLS is enabled by default. To disable TLS in Magnum, you can specify the parameter tls-disabled in the ClusterTemplate. Please note it is not recommended to disable TLS due to security reasons.

In the following example, Kubernetes is used to illustrate a secure cluster, but the steps are similar for other cluster types that have TLS support.

First, create a ClusterTemplate; by default TLS is enabled in Magnum, therefore it does not need to be specified via a parameter:

```
openstack coe cluster template create secure-kubernetes \
                          --keypair default \
                          --external-network public \
                          --image fedora-coreos-latest \
                          --dns-nameserver 8.8.8.8 \
                          --flavor m1.small \
```

(continues on next page)

(continued from previous page)

```
                        --docker-volume-size 3 \
                        --coe kubernetes \
                        --network-driver flannel


+-----------------------+--------------------------------------+
| Property              | Value                                |
+-----------------------+--------------------------------------+
| insecure_registry     | None                                 |
| http_proxy            | None                                 |
| updated_at            | None                                 |
| master_flavor_id      | None                                 |
| uuid                  | 5519b24a-621c-413c-832f-c30424528b31 |
| no_proxy              | None                                 |
| https_proxy           | None                                 |
| tls_disabled          | False                                |
| keypair_id            | time4funkey                          |
| public                | False                                |
| labels                | {}                                   |
| docker_volume_size    | 5                                    |
| server_type           | vm                                   |
| external_network_id   | public                               |
| cluster_distro        | fedora-coreos                        |
| image_id              | fedora-coreos-latest                 |
| volume_driver         | None                                 |
| registry_enabled      | False                                |
| docker_storage_driver | devicemapper                         |
| apiserver_port        | None                                 |
| name                  | secure-kubernetes                    |
| created_at            | 2016-07-25T23:09:50+00:00            |
| network_driver        | flannel                              |
| fixed_network         | None                                 |
| coe                   | kubernetes                           |
| flavor_id             | m1.small                             |
| dns_nameserver        | 8.8.8.8                              |
+-----------------------+--------------------------------------+
```

Now create a cluster. Use the ClusterTemplate name as a template for cluster creation:

```
openstack coe cluster create secure-k8s-cluster \
                      --cluster-template secure-kubernetes \
                      --node-count 1


+--------------------+------------------------------------------------------------
↪----+
| Property           | Value                                                      ␣
↪    |
+--------------------+------------------------------------------------------------
↪----+
| status             | CREATE_IN_PROGRESS                                         ␣
↪    |
```

(continues on next page)

(continued from previous page)

```
| uuid                | 3968ffd5-678d-4555-9737-35f191340fda                                 ␣
↪    |
| stack_id            | c96b66dd-2109-4ae2-b510-b3428f1e8761                                 ␣
↪    |
| status_reason       | None                                                                 ␣
↪    |
| created_at          | 2016-07-25T23:14:06+00:00                                            ␣
↪    |
| updated_at          | None                                                                 ␣
↪    |
| create_timeout      | 0                                                                    ␣
↪    |
| api_address         | None                                                                 ␣
↪    |
| coe_version         | -                                                                    ␣
↪    |
| cluster_template_id| 5519b24a-621c-413c-832f-c30424528b31                                  ␣
↪    |
| master_addresses    | None                                                                 ␣
↪    |
| node_count          | 1                                                                    ␣
↪    |
| node_addresses      | None                                                                 ␣
↪    |
| master_count        | 1                                                                    ␣
↪    |
| container_version   | -                                                                    ␣
↪    |
| discovery_url       | https://discovery.etcd.io/
↪ba52a8178e7364d43a323ee4387cf28e |
| name                | secure-k8s-cluster                                                   ␣
↪     |
+--------------------+------------------------------------------------------------
↪----+
```

Now run cluster-show command to get the details of the cluster and verify that the api_address is https:

```
openstack coe cluster show secure-k8scluster
+--------------------+------------------------------------------------------------
↪----+
| Property           | Value                                                        ␣
↪    |
+--------------------+------------------------------------------------------------
↪----+
| status             | CREATE_COMPLETE                                              ␣
↪    |
| uuid               | 04952c60-a338-437f-a7e7-d016d1d00e65                         ␣
↪    |
| stack_id           | b7bf72ce-b08e-4768-8201-e63a99346898                         ␣
↪    |
```

(continues on next page)

(continued from previous page)

```
| status_reason       | Stack CREATE completed successfully                                 ␣
↪     |
| created_at          | 2016-07-25T23:14:06+00:00                                           ␣
↪     |
| updated_at          | 2016-07-25T23:14:10+00:00                                           ␣
↪     |
| create_timeout      | 60                                                                  ␣
↪     |
| coe_version         | v1.2.0                                                              ␣
↪     |
| api_address         | https://192.168.19.86:6443                                          ␣
↪     |
| cluster_template_id| da2825a0-6d09-4208-b39e-b2db666f1118                                 ␣
↪     |
| master_addresses    | ['192.168.19.87']                                                   ␣
↪     |
| node_count          | 1                                                                   ␣
↪     |
| node_addresses      | ['192.168.19.88']                                                   ␣
↪     |
| master_count        | 1                                                                   ␣
↪     |
| container_version   | 1.9.1                                                               ␣
↪     |
| discovery_url       | https://discovery.etcd.io/
↪3b7fb09733429d16679484673ba3bfd5 |
| name                | secure-k8s-cluster                                                  ␣
↪     |
+---------------------+---------------------------------------------------------------
↪----+
```

You can see the api_address contains https in the URL, showing that the Kubernetes services are configured securely with SSL certificates and now any communication to kube-apiserver will be over https.

Interfacing with a secure cluster

To communicate with the API endpoint of a secure cluster, you will need so supply 3 SSL artifacts:

1. Your client key
2. A certificate for your client key that has been signed by a Certificate Authority (CA)
3. The certificate of the CA

There are two ways to obtain these 3 artifacts.

Automated

Magnum provides the command cluster-config to help the user in setting up the environment and artifacts for TLS, for example:

```
openstack coe cluster config swarm-cluster --dir myclusterconfig
```

This will display the necessary environment variables, which you can add to your environment:

```
export DOCKER_HOST=tcp://172.24.4.5:2376
export DOCKER_CERT_PATH=myclusterconfig
export DOCKER_TLS_VERIFY=True
```

And the artifacts are placed in the directory specified:

```
ca.pem
cert.pem
key.pem
```

You can now use the native client to interact with the COE. The variables and artifacts are unique to the cluster.

The parameters for coe cluster config are as follows:

dir <dirname> Directory to save the certificate and config files.

--force Overwrite existing files in the directory specified.

Manual

You can create the key and certificates manually using the following steps.

Client Key Your personal private key is essentially a cryptographically generated string of bytes. It should be protected in the same manner as a password. To generate an RSA key, you can use the genrsa command of the openssl tool:

```
openssl genrsa -out key.pem 4096
```

This command generates a 4096 byte RSA key at key.pem.

Signed Certificate To authenticate your key, you need to have it signed by a CA. First generate the Certificate Signing Request (CSR). The CSR will be used by Magnum to generate a signed certificate that you will use to communicate with the cluster. To generate a CSR, openssl requires a config file that specifies a few values. Using the example template below, you can fill in the CN value with your name and save it as client.conf:

```
$ cat > client.conf << END
[req]
distinguished_name = req_distinguished_name
req_extensions     = req_ext
prompt = no
[req_distinguished_name]
CN = Your Name
```

(continues on next page)

(continued from previous page)

```
[req_ext]
extendedKeyUsage = clientAuth
END
```

For RBAC enabled kubernetes clusters you need to use the name admin and system:masters as Organization (O=):

```
$ cat > client.conf << END
[req]
distinguished_name = req_distinguished_name
req_extensions     = req_ext
prompt = no
[req_distinguished_name]
CN = admin
O = system:masters
OU=OpenStack/Magnum
C=US
ST=TX
L=Austin
[req_ext]
extendedKeyUsage = clientAuth
END
```

Once you have client.conf, you can run the openssl req command to generate the CSR:

```
openssl req -new -days 365 \
    -config client.conf \
    -key key.pem \
    -out client.csr
```

Now that you have your client CSR, you can use the Magnum CLI to send it off to Magnum to get it signed:

```
openstack coe ca sign secure-k8s-cluster client.csr > cert.pem
```

Certificate Authority The final artifact you need to retrieve is the CA certificate for the cluster. This is used by your native client to ensure you are only communicating with hosts that Magnum set up:

```
openstack coe ca show secure-k8s-cluster > ca.pem
```

Rotate Certificate To rotate the CA certificate for a cluster and invalidate all user certificates, you can use the following command:

```
openstack coe ca rotate secure-k8s-cluster
```

Please note that now the CA rotate function is only supported by Fedora CoreOS driver.

User Examples

Here are some examples for using the CLI on a secure Kubernetes and Swarm cluster. You can perform all the TLS set up automatically by:

```
eval $(openstack coe cluster config <cluster-name>)
```

Or you can perform the manual steps as described above and specify the TLS options on the CLI. The SSL artifacts are assumed to be saved in local files as follows:

```
- key.pem: your SSL key
- cert.pem: signed certificate
- ca.pem: certificate for cluster CA
```

For Kubernetes, you need to get kubectl, a kubernetes CLI tool, to communicate with the cluster:

```
curl -O https://storage.googleapis.com/kubernetes-release/release/v1.2.0/bin/
↪linux/amd64/kubectl
chmod +x kubectl
sudo mv kubectl /usr/local/bin/kubectl
```

Now lets run some kubectl commands to check the secure communication. If you used cluster-config, then you can simply run the kubectl command without having to specify the TLS options since they have been defined in the environment:

```
kubectl version
Client Version: version.Info{Major:"1", Minor:"0", GitVersion:"v1.2.0",␣
↪GitCommit:"cffae0523cfa80ddf917aba69f08508b91f603d5", GitTreeState:"clean"}
Server Version: version.Info{Major:"1", Minor:"0", GitVersion:"v1.2.0",␣
↪GitCommit:"cffae0523cfa80ddf917aba69f08508b91f603d5", GitTreeState:"clean"}
```

You can specify the TLS options manually as follows:

```
KUBERNETES_URL=$(openstack coe cluster show secure-k8s-cluster |
                 awk '/ api_address /{print $4}')
kubectl version --certificate-authority=ca.pem \
                --client-key=key.pem \
                --client-certificate=cert.pem -s $KUBERNETES_URL

kubectl create -f redis-master.yaml --certificate-authority=ca.pem \
                                    --client-key=key.pem \
                                    --client-certificate=cert.pem -s
↪$KUBERNETES_URL

pods/test2

kubectl get pods --certificate-authority=ca.pem \
                 --client-key=key.pem \
                 --client-certificate=cert.pem -s $KUBERNETES_URL
NAME           READY     STATUS    RESTARTS   AGE
redis-master   2/2       Running   0          1m
```

Beside using the environment variables, you can also configure kubectl to remember the TLS options:

```
kubectl config set-cluster secure-k8s-cluster --server=${KUBERNETES_URL} \
    --certificate-authority=${PWD}/ca.pem
kubectl config set-credentials client --certificate-authority=${PWD}/ca.pem \
    --client-key=${PWD}/key.pem --client-certificate=${PWD}/cert.pem
kubectl config set-context secure-k8scluster --cluster=secure-k8scluster --
↪user=client
kubectl config use-context secure-k8scluster
```

Then you can use kubectl commands without the certificates:

```
kubectl get pods
NAME           READY     STATUS    RESTARTS   AGE
redis-master   2/2       Running   0          1m
```

Access to Kubernetes User Interface:

```
curl -L ${KUBERNETES_URL}/ui --cacert ca.pem --key key.pem \
    --cert cert.pem
```

You may also set up kubectl proxy which will use your client certificates to allow you to browse to a local address to use the UI without installing a certificate in your browser:

```
kubectl proxy --api-prefix=/ --certificate-authority=ca.pem --client-key=key.
↪pem \
              --client-certificate=cert.pem -s $KUBERNETES_URL
```

You can then open http://localhost:8001/ui in your browser.

The examples for Docker are similar. With cluster-config set up, you can just run docker commands without TLS options. To specify the TLS options manually:

```
docker -H tcp://192.168.19.86:2376 --tlsverify \
       --tlscacert ca.pem \
       --tlskey key.pem \
       --tlscert cert.pem \
       info
```

Storing the certificates

Magnum generates and maintains a certificate for each cluster so that it can also communicate securely with the cluster. As a result, it is necessary to store the certificates in a secure manner. Magnum provides the following methods for storing the certificates and this is configured in /etc/magnum/magnum.conf in the section [certificates] with the parameter cert_manager_type.

1. Barbican: Barbican is a service in OpenStack for storing secrets. It is used by Magnum to store the certificates when cert_manager_type is configured as:

   ```
   cert_manager_type = barbican
   ```

   This is the recommended configuration for a production environment. Magnum will interface with Barbican to store and retrieve certificates, delegating the task of securing the certificates to Barbican.

2. Magnum database: In some cases, a user may want an alternative to storing the certificates that does not require Barbican. This can be a development environment, or a private cloud that has been secured by other means. Magnum can store the certificates in its own database; this is done with the configuration:

```
cert_manager_type = x509keypair
```

   This storage mode is only as secure as the controller server that hosts the database for the OpenStack services.

3. Local store: As another alternative that does not require Barbican, Magnum can simply store the certificates on the local host filesystem where the conductor is running, using the configuration:

```
cert_manager_type = local
```

   Note that this mode is only supported when there is a single Magnum conductor running since the certificates are stored locally. The local mode is not recommended for a production environment.

For the nodes, the certificates for communicating with the masters are stored locally and the nodes are assumed to be secured.

4.1.13 Networking

There are two components that make up the networking in a cluster.

1. The Neutron infrastructure for the cluster: this includes the private network, subnet, ports, routers, load balancers, etc.
2. The networking model presented to the containers: this is what the containers see in communicating with each other and to the external world. Typically this consists of a driver deployed on each node.

The two components are deployed and managed separately. The Neutron infrastructure is the integration with OpenStack; therefore, it is stable and more or less similar across different COE types. The networking model, on the other hand, is specific to the COE type and is still under active development in the various COE communities, for example, Docker libnetwork and Kubernetes Container Networking. As a result, the implementation for the networking models is evolving and new models are likely to be introduced in the future.

For the Neutron infrastructure, the following configuration can be set in the ClusterTemplate:

external-network The external Neutron network ID to connect to this cluster. This is used to connect the cluster to the external internet, allowing the nodes in the cluster to access external URL for discovery, image download, etc. If not specified, the default value is public and this is valid for a typical devstack.

fixed-network The Neutron network to use as the private network for the cluster nodes. If not specified, a new Neutron private network will be created.

dns-nameserver The DNS nameserver to use for this cluster. This is an IP address for the server and it is used to configure the Neutron subnet of the cluster (dns_nameservers). If not specified, the default DNS is 8.8.8.8, the publicly available DNS.

http-proxy, https-proxy, no-proxy The proxy for the nodes in the cluster, to be used when the cluster is behind a firewall and containers cannot access URLs on the external internet directly. For the parameter http-proxy and https-proxy, the value to provide is a URL and it will be set in the

environment variable HTTP_PROXY and HTTPS_PROXY respectively in the nodes. For the parameter no-proxy, the value to provide is an IP or list of IPs separated by comma. Likewise, the value will be set in the environment variable NO_PROXY in the nodes.

For the networking model to the container, the following configuration can be set in the ClusterTemplate:

network-driver The network driver name for instantiating container networks. Currently, the following network drivers are supported:

Driver	Kubernetes	Swarm	Mesos
Flannel	supported	supported	unsupported
Docker	unsupported	supported	supported
Calico	supported	unsupported	unsupported

If not specified, the default driver is Flannel for Kubernetes, and Docker for Swarm and Mesos.

Particular network driver may require its own set of parameters for configuration, and these parameters are specified through the labels in the ClusterTemplate. Labels are arbitrary key=value pairs.

When Flannel is specified as the network driver, the following optional labels can be added:

flannel_network_cidr IPv4 network in CIDR format to use for the entire Flannel network. If not specified, the default is 10.100.0.0/16.

flannel_network_subnetlen The size of the subnet allocated to each host. If not specified, the default is 24.

flannel_backend The type of backend for Flannel. Possible values are *udp, vxlan, host-gw*. If not specified, the default is *udp*. Selecting the best backend depends on your networking. Generally, *udp* is the most generally supported backend since there is little requirement on the network, but it typically offers the lowest performance. The *vxlan* backend performs better, but requires vxlan support in the kernel so the image used to provision the nodes needs to include this support. The *host-gw* backend offers the best performance since it does not actually encapsulate messages, but it requires all the nodes to be on the same L2 network. The private Neutron network that Magnum creates does meet this requirement; therefore if the parameter *fixed_network* is not specified in the ClusterTemplate, *host-gw* is the best choice for the Flannel backend.

When Calico is specified as the network driver, the following optional labels can be added:

calico_ipv4pool IPv4 network in CIDR format which is the IP pool, from which Pod IPs will be chosen. If not specified, the default is 10.100.0.0/16. Stein default: 192.168.0.0/16 Train default: 192.168.0.0/16 Ussuri default: 10.100.0.0/16

calico_ipv4pool_ipip IPIP Mode to use for the IPv4 POOL created at start up. Ussuri default: Off

calico_tag Tag of the calico containers used to provision the calico node Stein default: v2.6.7 Train default: v3.3.6 Ussuri default: v3.13.1 Victoria default: v3.13.1 Wallaby default: v3.13.1

Besides, the Calico network driver needs kube_tag with v1.9.3 or later, because Calico needs extra mounts for the kubelet container. See commit of atomic-system-containers for more information.

NOTE: We have seen some issues using systemd as cgroup-driver with Calico together, so we highly recommend to use cgroupfs as the cgroup-driver for Calico.

Network for VMs

Every cluster has its own private network which is created along with the cluster. All the cluster nodes also get a floating ip on the external network. This approach works by default, but can be expensive in terms of complexity and cost (public Ipv4). To reduce this expense, the following methods can be used:

1. **Create private networks but do not assign floating IPs** With this approach the cluster *will* be inaccessible from the outside. The user can add a floating ip to access it, but the certificates will not work.
2. **Create a private network and a LoadBalancer for the master node(s)** There are two type of loadbalancers in magnum, one for the api and one for the services running on the nodes. For kubernetes LoadBalancer service type see: *Kubernetes External Load Balancer*. Not recommended when using only a single master node as it will add 2 amphora vms: one for the kube API and another for etcd thus being more expensive.

All the above can also work by passing an existing private network instead of creating a new one using fixed-network and fixed-subnet.

Flannel When using flannel, the backend should be host-gw if performance is a requirement, udp is too slow and vxlan creates one more overlay network on top of the existing neutron network. On the other hand, in a flat network one should use vxlan for network isolation.

Calico Calico allows users to setup network policies in kubernetes policies for network isolation.

4.1.14 High Availability

Support for highly available clusters is a work in progress, the goal being to enable clusters spanning multiple availability zones.

As of today you can specify one single availability zone for you cluster.

availability_zone The availability zone where the cluster nodes should be deployed. If not specified, the default is None.

4.1.15 Scaling

Performance tuning for periodic task

Magnums periodic task performs a *stack-get* operation on the Heat stack underlying each of its clusters. If you have a large amount of clusters this can create considerable load on the Heat API. To reduce that load you can configure Magnum to perform one global *stack-list* per periodic task instead of one per cluster. This is disabled by default, both from the Heat and Magnum side since it causes a security issue, though: any user in any tenant holding the *admin* role can perform a global *stack-list* operation if Heat is configured to allow it for Magnum. If you want to enable it nonetheless, proceed as follows:

1. Set *periodic_global_stack_list* in magnum.conf to *True* (*False* by default).
2. Update heat policy to allow magnum list stacks. To this end, edit your heat policy file, usually etc/heat/policy.yaml“:

```
...
stacks:global_index: "rule:context_is_admin"
```

   Now restart heat.

Containers and nodes

Scaling containers and nodes refers to increasing or decreasing allocated system resources. Scaling is a broad topic and involves many dimensions. In the context of Magnum in this guide, we consider the following issues:

- Scaling containers and scaling cluster nodes (infrastructure)
- Manual and automatic scaling

Since this is an active area of development, a complete solution covering all issues does not exist yet, but partial solutions are emerging.

Scaling containers involves managing the number of instances of the container by replicating or deleting instances. This can be used to respond to change in the workload being supported by the application; in this case, it is typically driven by certain metrics relevant to the application such as response time, etc. Other use cases include rolling upgrade, where a new version of a service can gradually be scaled up while the older version is gradually scaled down. Scaling containers is supported at the COE level and is specific to each COE as well as the version of the COE. You will need to refer to the documentation for the proper COE version for full details, but following are some pointers for reference.

For Kubernetes, pods are scaled manually by setting the count in the replication controller. Kubernetes version 1.3 and later also supports autoscaling. For Docker, the tool Docker Compose provides the command docker-compose scale which lets you manually set the number of instances of a container. For Swarm version 1.12 and later, services can also be scaled manually through the command docker service scale. Automatic scaling for Swarm is not yet available. Mesos manages the resources and does not support scaling directly; instead, this is provided by frameworks running within Mesos. With the Marathon framework currently supported in the Mesos cluster, you can use the scale operation on the Marathon UI or through a REST API call to manually set the attribute instance for a container.

Scaling the cluster nodes involves managing the number of nodes in the cluster by adding more nodes or removing nodes. There is no direct correlation between the number of nodes and the number of containers that can be hosted since the resources consumed (memory, CPU, etc) depend on the containers. However, if a certain resource is exhausted in the cluster, adding more nodes would add more resources for hosting more containers. As part of the infrastructure management, Magnum supports manual scaling through the attribute node_count in the cluster, so you can scale the cluster simply by changing this attribute:

```
openstack coe cluster update mycluster replace node_count=2
```

Refer to the section *Scale* lifecycle operation for more details.

Adding nodes to a cluster is straightforward: Magnum deploys additional VMs or baremetal servers through the heat templates and invokes the COE-specific mechanism for registering the new nodes to update the available resources in the cluster. Afterward, it is up to the COE or user to re-balance the workload by launching new container instances or re-launching dead instances on the new nodes.

Removing nodes from a cluster requires some more care to ensure continuous operation of the containers since the nodes being removed may be actively hosting some containers. Magnum performs a simple heuristic that is specific to the COE to find the best node candidates for removal, as follows:

Kubernetes Magnum scans the pods in the namespace Default to determine the nodes that are *not* hosting any (empty nodes). If the number of nodes to be removed is equal or less than the number of these empty nodes, these nodes will be removed from the cluster. If the number of nodes to be removed is larger than the number of empty nodes, a warning message will be sent to the Magnum log and the empty nodes along with additional nodes will be removed from the cluster. The additional nodes are selected randomly and the pods running on them will be deleted without warning.

For this reason, a good practice is to manage the pods through the replication controller so that the deleted pods will be relaunched elsewhere in the cluster. Note also that even when only the empty nodes are removed, there is no guarantee that no pod will be deleted because there is no locking to ensure that Kubernetes will not launch new pods on these nodes after Magnum has scanned the pods.

Swarm No node selection heuristic is currently supported. If you decrease the node_count, a node will be chosen by magnum without consideration of what containers are running on the selected node.

Mesos Magnum scans the running tasks on Marathon server to determine the nodes on which there is *no* task running (empty nodes). If the number of nodes to be removed is equal or less than the number of these empty nodes, these nodes will be removed from the cluster. If the number of nodes to be removed is larger than the number of empty nodes, a warning message will be sent to the Magnum log and the empty nodes along with additional nodes will be removed from the cluster. The additional nodes are selected randomly and the containers running on them will be deleted without warning. Note that even when only the empty nodes are removed, there is no guarantee that no container will be deleted because there is no locking to ensure that Mesos will not launch new containers on these nodes after Magnum has scanned the tasks.

Currently, scaling containers and scaling cluster nodes are handled separately, but in many use cases, there are interactions between the two operations. For instance, scaling up the containers may exhaust the available resources in the cluster, thereby requiring scaling up the cluster nodes as well. Many complex issues are involved in managing this interaction. A presentation at the OpenStack Tokyo Summit 2015 covered some of these issues along with some early proposals, Exploring Magnum and Senlin integration for autoscaling containers. This remains an active area of discussion and research.

4.1.16 Storage

Currently Cinder provides the block storage to the containers, and the storage is made available in two ways: as ephemeral storage and as persistent storage.

Ephemeral storage

The filesystem for the container consists of multiple layers from the image and a top layer that holds the modification made by the container. This top layer requires storage space and the storage is configured in the Docker daemon through a number of storage options. When the container is removed, the storage allocated to the particular container is also deleted.

Magnum can manage the containers filesystem in two ways, storing them on the local disk of the compute instances or in a separate Cinder block volume for each node in the cluster, mounts it to the node and configures it to be used as ephemeral storage. Users can specify the size of the Cinder volume with the ClusterTemplate attribute docker-volume-size. Currently the block size is fixed at cluster creation time, but future lifecycle operations may allow modifying the block size during the life of the cluster.

docker_volume_type For drivers that support additional volumes for container storage, a label named docker_volume_type is exposed so that users can select different cinder volume types for their volumes. The default volume *must* be set in default_docker_volume_type in the cinder section of magnum.conf, an obvious value is the default volume type set in cinder.conf of your cinder deployment . Please note, that docker_volume_type refers to a cinder volume type and it is unrelated to docker or kubernetes volumes.

Both local disk and the Cinder block storage can be used with a number of Docker storage drivers available.

- devicemapper: When used with a dedicated Cinder volume it is configured using direct-lvm and offers very good performance. If its used with the compute instances local disk uses a loopback device offering poor performance and its not recommended for production environments. Using the devicemapper driver does allow the use of SELinux.
- overlay When used with a dedicated Cinder volume offers as good or better performance than devicemapper. If used on the local disk of the compute instance (especially with high IOPS drives) you can get significant performance gains. However, for kernel versions less than 4.9, SELinux must be disabled inside the containers resulting in worse container isolation, although it still runs in enforcing mode on the cluster compute instances.
- overlay2 is the preferred storage driver, for all currently supported Linux distributions, and requires no extra configuration. When possible, overlay2 is the recommended storage driver. When installing Docker for the first time, overlay2 is used by default.

Persistent storage

In some use cases, data read/written by a container needs to persist so that it can be accessed later. To persist the data, a Cinder volume with a filesystem on it can be mounted on a host and be made available to the container, then be unmounted when the container exits.

Docker provides the volume feature for this purpose: the user invokes the volume create command, specifying a particular volume driver to perform the actual work. Then this volume can be mounted when a container is created. A number of third-party volume drivers support OpenStack Cinder as the backend, for example Rexray and Flocker. Magnum currently supports Rexray as the volume driver for Swarm and Mesos. Other drivers are being considered.

Kubernetes allows a previously created Cinder block to be mounted to a pod and this is done by specifying the block ID in the pod YAML file. When the pod is scheduled on a node, Kubernetes will interface with Cinder to request the volume to be mounted on this node, then Kubernetes will launch the Docker container with the proper options to make the filesystem on the Cinder volume accessible to the container in the pod. When the pod exits, Kubernetes will again send a request to Cinder to unmount the volumes filesystem, making it available to be mounted on other nodes.

Magnum supports these features to use Cinder as persistent storage using the ClusterTemplate attribute volume-driver and the support matrix for the COE types is summarized as follows:

Driver	Kubernetes	Swarm	Mesos
cinder	supported	unsupported	unsupported
rexray	unsupported	supported	supported

Following are some examples for using Cinder as persistent storage.

Using Cinder in Kubernetes

NOTE: This feature requires Kubernetes version 1.5.0 or above. The public Fedora image from Atomic currently meets this requirement.

1. Create the ClusterTemplate.

   Specify cinder as the volume-driver for Kubernetes:

```
openstack coe cluster template create k8s-cluster-template \
                          --image fedora-23-atomic-7 \
                          --keypair testkey \
                          --external-network public \
                          --dns-nameserver 8.8.8.8 \
                          --flavor m1.small \
                          --docker-volume-size 5 \
                          --network-driver flannel \
                          --coe kubernetes \
                          --volume-driver cinder
```

2. Create the cluster:

```
openstack coe cluster create k8s-cluster \
                      --cluster-template k8s-cluster-template \
                      --node-count 1
```

Kubernetes is now ready to use Cinder for persistent storage. Following is an example illustrating how Cinder is used in a pod.

1. Create the cinder volume:

```
cinder create --display-name=test-repo 1

ID=$(cinder create --display-name=test-repo 1 | awk -F'|' '$2~/^
↪[[:space:]]*id/ {print $3}')
```

   The command will generate the volume with a ID. The volume ID will be specified in Step 2.

2. Create a pod in this cluster and mount this cinder volume to the pod. Create a file (e.g nginx-cinder.yaml) describing the pod:

```
cat > nginx-cinder.yaml << END
apiVersion: v1
kind: Pod
metadata:
  name: aws-web
spec:
  containers:
    - name: web
      image: nginx
      ports:
        - name: web
          containerPort: 80
```

(continues on next page)

(continued from previous page)

```
          hostPort: 8081
          protocol: TCP
      volumeMounts:
        - name: html-volume
          mountPath: "/usr/share/nginx/html"
  volumes:
    - name: html-volume
      cinder:
        # Enter the volume ID below
        volumeID: $ID
        fsType: ext4
END
```

NOTE: The Cinder volume ID needs to be configured in the YAML file so the existing Cinder volume can be mounted in a pod by specifying the volume ID in the pod manifest as follows:

```
volumes:
- name: html-volume
  cinder:
    volumeID: $ID
    fsType: ext4
```

3. Create the pod by the normal Kubernetes interface:

```
kubectl create -f nginx-cinder.yaml
```

You can start a shell in the container to check that the mountPath exists, and on an OpenStack client you can run the command cinder list to verify that the cinder volume status is in-use.

Using Cinder in Swarm

To be filled in

Using Cinder in Mesos

1. Create the ClusterTemplate.

   Specify rexray as the volume-driver for Mesos. As an option, you can specify in a label the attributes rexray_preempt to enable any host to take control of a volume regardless if other hosts are using the volume. If this is set to false, the driver will ensure data safety by locking the volume:

```
openstack coe cluster template create mesos-cluster-template \
                          --image ubuntu-mesos \
                          --keypair testkey \
                          --external-network public \
                          --dns-nameserver 8.8.8.8 \
                          --master-flavor m1.magnum \
                          --docker-volume-size 4 \
                          --tls-disabled \
```

(continues on next page)

(continued from previous page)

```
                              --flavor m1.magnum \
                              --coe mesos \
                              --volume-driver rexray \
                              --labels rexray-preempt=true
```

2. Create the Mesos cluster:

```
openstack coe cluster create mesos-cluster \
                        --cluster-template mesos-cluster-template \
                        --node-count 1
```

3. Create the cinder volume and configure this cluster:

```
cinder create --display-name=redisdata 1
```

Create the following file

```
cat > mesos.json << END
{
  "id": "redis",
  "container": {
    "docker": {
    "image": "redis",
    "network": "BRIDGE",
    "portMappings": [
      { "containerPort": 80, "hostPort": 0, "protocol": "tcp"}
    ],
    "parameters": [
       { "key": "volume-driver", "value": "rexray" },
       { "key": "volume", "value": "redisdata:/data" }
    ]
    }
 },
 "cpus": 0.2,
 "mem": 32.0,
 "instances": 1
}
END
```

NOTE: When the Mesos cluster is created using this ClusterTemplate, the Mesos cluster will be configured so that a filesystem on an existing cinder volume can be mounted in a container by configuring the parameters to mount the cinder volume in the JSON file

```
"parameters": [
   { "key": "volume-driver", "value": "rexray" },
   { "key": "volume", "value": "redisdata:/data" }
]
```

4. Create the container using Marathon REST API

```
MASTER_IP=$(openstack coe cluster show mesos-cluster | awk '/ api_address␣
↪/{print $4}')
curl -X POST -H "Content-Type: application/json" \
http://${MASTER_IP}:8080/v2/apps -d@mesos.json
```

You can log into the container to check that the mountPath exists, and you can run the command cinder list to verify that your cinder volume status is in-use.

4.1.17 Image Management

When a COE is deployed, an image from Glance is used to boot the nodes in the cluster and then the software will be configured and started on the nodes to bring up the full cluster. An image is based on a particular distro such as Fedora, Ubuntu, etc, and is prebuilt with the software specific to the COE such as Kubernetes, Swarm, Mesos. The image is tightly coupled with the following in Magnum:

1. Heat templates to orchestrate the configuration.
2. Template definition to map ClusterTemplate parameters to Heat template parameters.
3. Set of scripts to configure software.

Collectively, they constitute the driver for a particular COE and a particular distro; therefore, developing a new image needs to be done in conjunction with developing these other components. Image can be built by various methods such as diskimagebuilder, or in some case, a distro image can be used directly. A number of drivers and the associated images is supported in Magnum as reference implementation. In this section, we focus mainly on the supported images.

All images must include support for cloud-init and the heat software configuration utility:

- os-collect-config
- os-refresh-config
- os-apply-config
- heat-config
- heat-config-script

Additional software are described as follows.

Kubernetes on Fedora CoreOS

Fedoara CoreOS publishes a stock OpenStack image that is being used to deploy Kubernetes.

The following software are managed as systemd services:

- kube-apiserver
- kube-controller-manager
- kube-scheduler
- kube-proxy
- kubelet
- docker

- etcd

The login user for this image is *core*.

Kubernetes on Ironic

This image is built manually using diskimagebuilder. The scripts and instructions are included in Magnum code repo. Currently Ironic is not fully supported yet, therefore more details will be provided when this driver has been fully tested.

Swarm on Fedora Atomic

This image can be downloaded from the public Atomic site or can be built locally using diskimagebuilder. The login for this image is *fedora*.

Mesos on Ubuntu

This image is built manually using diskimagebuilder. The instructions are provided in the section *Diskimage-builder*. The Fedora site hosts the current image ubuntu-mesos-latest.qcow2.

OS/software	version
Ubuntu	14.04
Docker	1.8.1
Mesos	0.25.0
Marathon	0.11.1

4.1.18 Notification

Magnum provides notifications about usage data so that 3rd party applications can use the data for auditing, billing, monitoring, or quota purposes. This document describes the current inclusions and exclusions for Magnum notifications.

Magnum uses Cloud Auditing Data Federation (CADF) Notification as its notification format for better support of auditing, details about CADF are documented below.

Auditing with CADF

Magnum uses the PyCADF library to emit CADF notifications, these events adhere to the DMTF CADF specification. This standard provides auditing capabilities for compliance with security, operational, and business processes and supports normalized and categorized event data for federation and aggregation.

Below table describes the event model components and semantics for each component:

model component	CADF Definition
OBSERVER	The RESOURCE that generates the CADF Event Record based on its observation (directly or indirectly) of the Actual Event.
INITIATOR	The RESOURCE that initiated, originated, or instigated the events ACTION, according to the OBSERVER.
ACTION	The operation or activity the INITIATOR has performed, has attempted to perform or has pending against the events TARGET, according to the OBSERVER.
TARGET	The RESOURCE against which the ACTION of a CADF Event Record was performed, attempted, or is pending, according to the OBSERVER.
OUTCOME	The result or status of the ACTION against the TARGET, according to the OBSERVER.

The `payload` portion of a CADF Notification is a CADF `event`, which is represented as a JSON dictionary. For example:

```
{
    "typeURI": "http://schemas.dmtf.org/cloud/audit/1.0/event",
    "initiator": {
        "typeURI": "service/security/account/user",
        "host": {
            "agent": "curl/7.22.0(x86_64-pc-linux-gnu)",
            "address": "127.0.0.1"
        },
        "id": "<initiator_id>"
    },
    "target": {
        "typeURI": "<target_uri>",
        "id": "openstack:1c2fc591-facb-4479-a327-520dade1ea15"
    },
    "observer": {
        "typeURI": "service/security",
        "id": "openstack:3d4a50a9-2b59-438b-bf19-c231f9c7625a"
    },
    "eventType": "activity",
    "eventTime": "2014-02-14T01:20:47.932842+00:00",
    "action": "<action>",
    "outcome": "success",
    "id": "openstack:f5352d7b-bee6-4c22-8213-450e7b646e9f",
}
```

Where the following are defined:

- `<initiator_id>`: ID of the user that performed the operation
- `<target_uri>`: CADF specific target URI, (i.e.: data/security/project)
- `<action>`: The action being performed, typically: `<operation>`. `<resource_type>`

Additionally there may be extra keys present depending on the operation being performed, these will be discussed below.

Note, the `eventType` property of the CADF payload is different from the `event_type` property of a notifications. The former (`eventType`) is a CADF keyword which designates the type of event that is being measured, this can be: *activity*, *monitor* or *control*. Whereas the latter (`event_type`) is described in previous sections as: *magnum.<resource_type>.<operation>*

Supported Events

The following table displays the corresponding relationship between resource types and operations. The bay type is deprecated and will be removed in a future version. Cluster is the new equivalent term.

resource type	supported operations	typeURI
bay	create, update, delete	service/magnum/bay
cluster	create, update, delete	service/magnum/cluster

Example Notification - Cluster Create

The following is an example of a notification that is sent when a cluster is created. This example can be applied for any `create`, `update` or `delete` event that is seen in the table above. The <action> and `typeURI` fields will be change.


```
{
    "event_type": "magnum.cluster.created",
    "message_id": "0156ee79-b35f-4cef-ac37-d4a85f231c69",
    "payload": {
        "typeURI": "http://schemas.dmtf.org/cloud/audit/1.0/event",
        "initiator": {
            "typeURI": "service/security/account/user",
            "id": "c9f76d3c31e142af9291de2935bde98a",
            "user_id": "0156ee79-b35f-4cef-ac37-d4a85f231c69",
            "project_id": "3d4a50a9-2b59-438b-bf19-c231f9c7625a"
        },
        "target": {
            "typeURI": "service/magnum/cluster",
            "id": "openstack:1c2fc591-facb-4479-a327-520dade1ea15"
        },
        "observer": {
            "typeURI": "service/magnum/cluster",
            "id": "openstack:3d4a50a9-2b59-438b-bf19-c231f9c7625a"
        },
        "eventType": "activity",
        "eventTime": "2015-05-20T01:20:47.932842+00:00",
        "action": "create",
        "outcome": "success",
        "id": "openstack:f5352d7b-bee6-4c22-8213-450e7b646e9f",
        "resource_info": "671da331c47d4e29bb6ea1d270154ec3"
    }
    "priority": "INFO",
    "publisher_id": "magnum.host1234",
```


(continues on next page)

(continued from previous page)

```
    "timestamp": "2016-05-20 15:03:45.960280"
}
```

4.1.19 Container Monitoring

As of this moment, monitoring is only supported for Kubernetes clusters. For details, please refer to the *Container Monitoring in Kubernetes* document.

4.1.20 Kubernetes Post Install Manifest

A new config option *post_install_manifest_url* under *[kubernetes]* section has been added to support installing cloud provider/vendor specific manifest after provisioning the k8s cluster. Its an URL pointing to the manifest file. For example, cloud admin can set their specific *StorageClass* into this file, then it will be automatically setup after the cluster is created by end user.

NOTE: The URL must be reachable from the master nodes when creating the cluster.

4.1.21 Kubernetes External Load Balancer

In a Kubernetes cluster, all masters and minions are connected to a private Neutron subnet, which in turn is connected by a router to the public network. This allows the nodes to access each other and the external internet.

All Kubernetes pods and services created in the cluster are connected to a private container network which by default is Flannel, an overlay network that runs on top of the Neutron private subnet. The pods and services are assigned IP addresses from this container network and they can access each other and the external internet. However, these IP addresses are not accessible from an external network.

To publish a service endpoint externally so that the service can be accessed from the external network, Kubernetes provides the external load balancer feature. This is done by simply specifying the attribute type: LoadBalancer in the service manifest. When the service is created, Kubernetes will add an external load balancer in front of the service so that the service will have an external IP address in addition to the internal IP address on the container network. The service endpoint can then be accessed with this external IP address. Refer to the Kubernetes service document for more details.

A Kubernetes cluster deployed by Magnum will have all the necessary configuration required for the external load balancer. This document describes how to use this feature.

Steps for the cluster administrator

Because the Kubernetes master needs to interface with OpenStack to create and manage the Neutron load balancer, we need to provide a credential for Kubernetes to use.

In the current implementation, the cluster administrator needs to manually perform this step. We are looking into several ways to let Magnum automate this step in a secure manner. This means that after the Kubernetes cluster is initially deployed, the load balancer support is disabled. If the administrator does not want to enable this feature, no further action is required. All the services will be created normally; services that specify the load balancer will also be created successfully, but a load balancer will not be created.

Note that different versions of Kubernetes require different versions of Neutron LBaaS plugin running on the OpenStack instance:

Kubernetes Version on Master	Neutron LBaaS Version Required
1.2	LBaaS v1
1.3 or later	LBaaS v2

Before enabling the Kubernetes load balancer feature, confirm that the OpenStack instance is running the required version of Neutron LBaaS plugin. To determine if your OpenStack instance is running LBaaS v1, try running the following command from your OpenStack control node:

```
neutron lb-pool-list
```

Or look for the following configuration in neutron.conf or neutron_lbaas.conf:

```
service_provider = LOADBALANCER:Haproxy:neutron_lbaas.services.loadbalancer.
↪drivers.haproxy.plugin_driver.HaproxyOnHostPluginDriver:default
```

To determine if your OpenStack instance is running LBaaS v2, try running the following command from your OpenStack control node:

```
neutron lbaas-pool-list
```

Or look for the following configuration in neutron.conf or neutron_lbaas.conf:

```
service_plugins = neutron.plugins.services.agent_loadbalancer.plugin.
↪LoadBalancerPluginv2
```

To configure LBaaS v1 or v2, refer to the Neutron documentation.

Before deleting the Kubernetes cluster, make sure to delete all the services that created load balancers. Because the Neutron objects created by Kubernetes are not managed by Heat, they will not be deleted by Heat and this will cause the cluster-delete operation to fail. If this occurs, delete the neutron objects manually (lb-pool, lb-vip, lb-member, lb-healthmonitor) and then run cluster-delete again.

Steps for the users

This feature requires the OpenStack cloud provider to be enabled. To do so, enable the cinder support (volume-driver cinder).

For the user, publishing the service endpoint externally involves the following 2 steps:

1. Specify type: LoadBalancer in the service manifest
2. After the service is created, associate a floating IP with the VIP of the load balancer pool.

The following example illustrates how to create an external endpoint for a pod running nginx.

Create a file (e.g nginx.yaml) describing a pod running nginx:

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
   app: nginx
spec:
  containers:
  - name: nginx
    image: nginx
    ports:
    - containerPort: 80
```

Create a file (e.g nginx-service.yaml) describing a service for the nginx pod:

```
apiVersion: v1
kind: Service
metadata:
  name: nginxservice
  labels:
    app: nginx
spec:
  ports:
  - port: 80
    targetPort: 80
    protocol: TCP
  selector:
    app: nginx
  type: LoadBalancer
```

Please refer to *Developer Quick-Start* on how to connect to Kubernetes running on the launched cluster. Assuming a Kubernetes cluster named k8sclusterv1 has been created, deploy the pod and service using following commands:

```
kubectl create -f nginx.yaml

kubectl create -f nginx-service.yaml
```

For more details on verifying the load balancer in OpenStack, refer to the following section on how it works.

Next, associate a floating IP to the load balancer. This can be done easily on Horizon by navigating to:

```
Compute -> Access & Security -> Floating IPs
```

Click on Allocate IP To Project and then on Associate for the new floating IP.

Alternatively, associating a floating IP can be done on the command line by allocating a floating IP, finding the port of the VIP, and associating the floating IP to the port. The commands shown below are for illustration purpose and assume that there is only one service with load balancer running in the cluster and no other load balancers exist except for those created for the cluster.

First create a floating IP on the public network:

```
neutron floatingip-create public

Created a new floatingip:

+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| fixed_ip_address    |                                      |
| floating_ip_address | 172.24.4.78                          |
| floating_network_id | 4808eacb-e1a0-40aa-97b6-ecb745af2a4d |
| id                  | b170eb7a-41d0-4c00-9207-18ad1c30fecf |
| port_id             |                                      |
| router_id           |                                      |
| status              | DOWN                                 |
| tenant_id           | 012722667dc64de6bf161556f49b8a62     |
+---------------------+--------------------------------------+
```

Note the floating IP 172.24.4.78 that has been allocated. The ID for this floating IP is shown above, but it can also be queried by:

```
FLOATING_ID=$(neutron floatingip-list | grep "172.24.4.78" | awk '{print $2}')
```

Next find the VIP for the load balancer:

```
VIP_ID=$(neutron lb-vip-list | grep TCP | grep -v pool | awk '{print $2}')
```

Find the port for this VIP:

```
PORT_ID=$(neutron lb-vip-show $VIP_ID | grep port_id | awk '{print $4}')
```

Finally associate the floating IP with the port of the VIP:

```
neutron floatingip-associate $FLOATING_ID $PORT_ID
```

The endpoint for nginx can now be accessed on a browser at this floating IP:

```
http://172.24.4.78:80
```

Alternatively, you can check for the nginx welcome message by:

```
curl http://172.24.4.78:80
```

NOTE: it is not necessary to indicate port :80 here but it is shown to correlate with the port that was specified in the service manifest.

How it works

Kubernetes is designed to work with different Clouds such as Google Compute Engine (GCE), Amazon Web Services (AWS), and OpenStack; therefore, different load balancers need to be created on the particular Cloud for the services. This is done through a plugin for each Cloud and the OpenStack plugin was developed by Angus Lees:

```
https://github.com/kubernetes/kubernetes/blob/release-1.0/pkg/cloudprovider/
↪openstack/openstack.go
```

When the Kubernetes components kube-apiserver and kube-controller-manager start up, they will use the credential provided to authenticate a client to interface with OpenStack.

When a service with load balancer is created, the plugin code will interface with Neutron in this sequence:

1. Create lb-pool for the Kubernetes service
2. Create lb-member for the minions
3. Create lb-healthmonitor
4. Create lb-vip on the private network of the Kubernetes cluster

These Neutron objects can be verified as follows. For the load balancer pool:

```
neutron lb-pool-list
+--------------------------------------+--------------------------------------
↪-----------+----------+-------------+----------+----------------+--------+
| id                                   | name                                 ␣
↪            | provider | lb_method   | protocol | admin_state_up | status |
+--------------------------------------+--------------------------------------
↪-----------+----------+-------------+----------+----------------+--------+
| 241357b3-2a8f-442e-b534-bde7cd6ba7e4 | a1f03e40f634011e59c9efa163eae8ab     ␣
↪            | haproxy  | ROUND_ROBIN | TCP      | True           | ACTIVE |
| 82b39251-1455-4eb6-a81e-802b54c2df29 | k8sclusterv1-iypacicrskib-api_pool-
↪fydshw7uvr7h  | haproxy  | ROUND_ROBIN | HTTP     | True           | ACTIVE␣
↪|
| e59ea983-c6e8-4cec-975d-89ade6b59e50 | k8sclusterv1-iypacicrskib-etcd_pool-
↪qbpo43ew2m3x | haproxy  | ROUND_ROBIN | HTTP     | True           | ACTIVE |
+--------------------------------------+--------------------------------------
↪-----------+----------+-------------+----------+----------------+--------+
```

Note that 2 load balancers already exist to implement high availability for the cluster (api and ectd). The new load balancer for the Kubernetes service uses the TCP protocol and has a name assigned by Kubernetes.

For the members of the pool:

```
neutron lb-member-list
+--------------------------------------+----------+---------------+--------+--
↪-------------+--------+
| id                                   | address  | protocol_port | weight |␣
↪admin_state_up | status |
+--------------------------------------+----------+---------------+--------+--
↪-------------+--------+
```

(continues on next page)

(continued from previous page)

```
| 9ab7dcd7-6e10-4d9f-ba66-861f4d4d627c | 10.0.0.5 |          8080 |      1 |␣
↪True           | ACTIVE |
| b179c1ad-456d-44b2-bf83-9cdc127c2b27 | 10.0.0.5 |          2379 |      1 |␣
↪True           | ACTIVE |
| f222b60e-e4a9-4767-bc44-ffa66ec22afe | 10.0.0.6 |         31157 |      1 |␣
↪True           | ACTIVE |
+--------------------------------------+----------+---------------+--------+--
↪-------------+--------+
```

Again, 2 members already exist for high availability and they serve the master node at 10.0.0.5. The new member serves the minion at 10.0.0.6, which hosts the Kubernetes service.

For the monitor of the pool:

```
neutron lb-healthmonitor-list
+--------------------------------------+------+----------------+
| id                                   | type | admin_state_up |
+--------------------------------------+------+----------------+
| 381d3d35-7912-40da-9dc9-b2322d5dda47 | TCP  | True           |
| 67f2ae8f-ffc6-4f86-ba5f-1a135f4af85c | TCP  | True           |
| d55ff0f3-9149-44e7-9b52-2e055c27d1d3 | TCP  | True           |
+--------------------------------------+------+----------------+
```

For the VIP of the pool:

```
neutron lb-vip-list
+--------------------------------------+----------------------------------+---
↪-------+----------+----------------+--------+
| id                                   | name                             |␣
↪address   | protocol | admin_state_up | status |
+--------------------------------------+----------------------------------+---
↪-------+----------+----------------+--------+
| 9ae2ebfb-b409-4167-9583-4a3588d2ff42 | api_pool.vip                     |␣
↪10.0.0.3 | HTTP     | True           | ACTIVE |
| c318aec6-8b7b-485c-a419-1285a7561152 | a1f03e40f634011e59c9efa163eae8ab |␣
↪10.0.0.7 | TCP      | True           | ACTIVE |
| fc62cf40-46ad-47bd-aa1e-48339b95b011 | etcd_pool.vip                    |␣
↪10.0.0.4 | HTTP     | True           | ACTIVE |
+--------------------------------------+----------------------------------+---
↪-------+----------+----------------+--------+
```

Note that the VIP is created on the private network of the cluster; therefore it has an internal IP address of 10.0.0.7. This address is also associated as the external address of the Kubernetes service. You can verify this in Kubernetes by running following command:

```
kubectl get services
NAME           LABELS                                    SELECTOR    IP(S)    ␣
↪        PORT(S)
kubernetes     component=apiserver,provider=kubernetes   <none>      10.254.0.
↪1       443/TCP
nginxservice   app=nginx                                 app=nginx   10.254.
↪122.191   80/TCP
```

(continues on next page)

(continued from previous page)

```
                                                                    10.0.0.7
```

On GCE, the networking implementation gives the load balancer an external address automatically. On OpenStack, we need to take the additional step of associating a floating IP to the load balancer.

4.1.22 Rolling Upgrade

Rolling upgrade is an important feature a user may want for a managed Kubernetes service.

Note: Kubernetes version upgrade is only supported by the Fedora Atomic and the Fedora CoreOS drivers.

A user can run a command as shown below to trigger a rolling ugprade for Kubernetes version upgrade or node operating system version upgrade.

```
openstack coe cluster upgrade <cluster ID> <new cluster template ID>
```

The key parameter in the command is the new cluster template ID. For Kubernetes version upgrade, a newer version for label *kube_tag* should be provided. Downgrade is not supported.

A simple operating system upgrade can be applied using a new image ID in the new cluster template. However, this entails a downtime for applications running on the cluster, because all the nodes will be rebuilt one by one.

The Fedora Atomic driver supports a more gradeful operating system upgrade. Similar to the Kubernetes version upgrade, it will cordon and drain the nodes before upgrading the operating system with rpm-ostree command. There are one of two labels which must be provided to support this feature:

- *ostree_commit*: this is a commit ID of ostree the current system should be upgraded to. An example of a commit ID is *1766b4526f1a738ba1e6e0a66264139f65340bcc28e7045f10cbe6d161eb1925*,
- *ostree_remote*: this is a remote name of ostree the current system should be rebased to. An example of a remote name is *fedora-atomic:fedora/29/x86_64/atomic-host*.

If both labels are present, *ostree_commit* takes precedence. To check if there are updates available, run *sudo rpm-ostree upgrade check* on the Atomic host which will show you the latest commit ID that can be upgraded to.

4.1.23 Keystone Authentication and Authorization for Kubernetes

Currently, there are several ways to access the Kubernetes API, such as RBAC, ABAC, Webhook, etc. Though RBAC is the best way for most of the cases, Webhook provides a good approach for Kubernetes to query an outside REST service when determining user privileges. In other words, we can use a Webhook to integrate other IAM service into Kubernetes. In our case, under the OpenStack context, were introducing the intergration with Keystone auth for Kubernetes.

Since Rocky release, we introduced a new label named *keystone_auth_enabled*, by default its True, which means user can get this very nice feature out of box.

Create roles

As cloud provider, necessary Keystone roles for Kubernetes cluster operations need to be created for different users, e.g. k8s_admin, k8s_developer, k8s_viewer

- k8s_admin role can create/update/delete Kubernetes cluster, can also associate roles to other normal users within the tenant
- k8s_developer can create/update/delete/watch Kubernetes cluster resources
- k8s_viewer can only have read access to Kubernetes cluster resources

NOTE: Those roles will be created automatically in devstack. Below is the samples commands about how to create them.

```
source ~/openstack_admin_credentials
for role in "k8s_admin" "k8s_developer" "k8s_viewer"; do openstack role␣
↪create $role; done

openstack user create demo_viewer --project demo --password password
openstack role add --user demo_viewer --project demo k8s_viewer

openstack user create demo_editor --project demo --password password
openstack role add --user demo_developer --project demo k8s_developer

openstack user create demo_admin --project demo --password password
openstack role add --user demo_admin --project demo k8s_admin
```

Those roles should be public and can be accessed by any project so that user can configure their clusters role policies with those roles.

Setup configmap for authorization policies

While the *k8s-keystone-auth* service is enabled in clusters by default, users will need specify their own authorization policy to start making use of this feature.

The user can specify their own authorization policy by either:

- Updating the placeholder *k8s-keystone-auth-policy* configmap, created by default in the *kube-system* namespace. This does not require restarting the *k8s-keystone-auth* service.
- Reading the policy from a default policy file. In devstack the policy file is created automatically.

Currently, the *k8s-keystone-auth* service supports four types of policies:

- user. The Keystone user ID or name.
- project. The Keystone project ID or name.
- role. The user role defined in Keystone.
- group. The group is not a Keystone concept actually, its supported for backward compatibility, you can use group as project ID.

For example, if we wish to configure a policy to only allow the users in project *demo* with *k8s-viewer* role in OpenStack to query the pod information from all the namespaces, then we can update the default *k8s-keystone-auth-policy* configmap as follows.

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
name: k8s-keystone-auth-policy
namespace: kube-system
data:
policies: |
    [
    {
        "resource": {
        "verbs": ["get", "list", "watch"],
        "resources": ["pods"],
        "version": "*",
        "namespace": "default"
        },
        "match": [
        {
            "type": "role",
            "values": ["k8s-viewer"]
        },
        {
            "type": "project",
            "values": ["demo"]
        }
        ]
    }
    ]
EOF
```

More on keystone authorization policies can be found in the kubernetes/cloud-provider-openstack documentation for Using the Keystone Webhook Authenticator and Authorizer

Note: If the user wishes to use an alternate name for the *k8s-keystone-auth-policy* configmap they will need to update the value of the *policy-configmap-name* parameter passed to the *k8s-keystone-auth* service and then restart the service.

Next the user needs to get a token from Keystone to have a kubeconfig for kubectl. The user can also get the config with Magnum python client.

Here is a sample of the kubeconfig:

```
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: CERT-DATA==
    server: https://172.24.4.25:6443
name: k8s-2
contexts:
- context:
    cluster: k8s-2
    user: openstackuser
```

(continues on next page)

(continued from previous page)

```
name: openstackuser@kubernetes
current-context: openstackuser@kubernetes
kind: Config
preferences: {}
users:
- name: openstackuser
user:
    exec:
    command: /bin/bash
    apiVersion: client.authentication.k8s.io/v1alpha1
    args:
    - -c
    - >
        if [ -z ${OS_TOKEN} ]; then
            echo 'Error: Missing OpenStack credential from environment␣
↪variable $OS_TOKEN' > /dev/stderr
            exit 1
        else
            echo '{ "apiVersion": "client.authentication.k8s.io/v1alpha1",
↪"kind": "ExecCredential", "status": { "token": "'"${OS_TOKEN}"'"}}'
        fi
```

After exporting the Keystone token to the `OS_TOKEN` environment variable, the user should be able to list pods with *kubectl*.

Setup configmap for role synchronization policies

To start taking advantage of role synchronization between kubernetes and openstack users need to specify an authentication synchronization policy

Users can specify their own policy by either:

- Updating the placeholder *keystone-sync-policy* configmap, created by default in the *kube-system* namespace. This does *not* require restarting *k8s-keystone-auth*
- Reading the policy from a local config file. This requires restarting the *k8s-keystone-auth* service.

For example, to set a policy which assigns the *project-1* group in kubernetes to users who have been assigned the *member* role in Keystone the user can update the default *keystone-sync-policy* configmap as follows.

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  name: keystone-sync-policy
  namespace: kube-system
data:
  syncConfig: |
    role-mappings:
      - keystone-role: member
```

(continues on next page)

(continued from previous page)

```
        groups: ["project-1"]
EOF
```

If users wish to use an alternative name for the keystone-sync-policy configmap they will need to update the value of the `--sync-configmap-name` parameter passed to the *k8s-keystone-auth* service and then restart service.

For more examples and information on configuring and using authorization synchronization policies please refer to the kubernetes/cloud-provider-openstack documentation for Authentication synchronization between Keystone and Kubernetes

4.1.24 Node Groups

Node groups can be used to create heterogeneous clusters.

This functionality is only supported for Kubernetes clusters.

When a cluster is created it already has two node groups, `default-master` and `default-worker`.

```
$ openstack coe cluster list
+--------------------------------------+------+----------+------------+------
↪--------+-----------------+---------------+
| uuid                                 | name | keypair   | node_count |␣
↪master_count | status          | health_status |
+--------------------------------------+------+----------+------------+------
↪--------+-----------------+---------------+
| ef7011bb-d404-4198-a145-e8808204cde3 | kube | default   |          1 |     ␣
↪       1 | CREATE_COMPLETE | HEALTHY       |
+--------------------------------------+------+----------+------------+------
↪--------+-----------------+---------------+

$ openstack coe nodegroup list kube
+--------------------------------------+----------------+----------+---------
↪-------------------------------+------------+-----------------+--------+
| uuid                                 | name           | flavor_id | image_
↪id                                | node_count | status          | role   |
+--------------------------------------+----------------+----------+---------
↪-------------------------------+------------+-----------------+--------+
| adc3ecfa-d11e-4da7-8c44-4092ea9dddd9 | default-master | m1.small  | Fedora-
↪AtomicHost-29-20190820.0.x86_64 |          1 | CREATE_COMPLETE | master |
| 186e131f-8103-4285-a900-eb0dcf18a670 | default-worker | m1.small  | Fedora-
↪AtomicHost-29-20190820.0.x86_64 |          1 | CREATE_COMPLETE | worker |
+--------------------------------------+----------------+----------+---------
↪-------------------------------+------------+-----------------+--------+
```

The `default-worker` node group cannot be removed or reconfigured, so the initial cluster configuration should take this into account.

Create a new node group

To add a new node group, use `openstack coe nodegroup create`. The only required parameters are the cluster ID and the name for the new node group, but several extra options are available.

Roles

Roles can be used to show the purpose of a node group, and multiple node groups can be given the same role if they share a common purpose.

```
$ openstack coe nodegroup create kube test-ng --node-count 1 --role test
```

When listing node groups, the role may be used as a filter:

```
$ openstack coe nodegroup list kube --role test
+--------------------------------------+---------+-----------+---------------
↪-----------------------+------------+--------------------+------+
| uuid                                 | name    | flavor_id | image_id      ␣
↪                       | node_count | status             | role |
+--------------------------------------+---------+-----------+---------------
↪-----------------------+------------+--------------------+------+
| b4ab1fcb-f23a-4d1f-b583-d699a2f1e2d7 | test-ng | m1.small  | Fedora-
↪AtomicHost-29-20190820.0.x86_64 |          1 | CREATE_IN_PROGRESS | test |
+--------------------------------------+---------+-----------+---------------
↪-----------------------+------------+--------------------+------+
```

The node group role will default to worker if unset, and the only reserved role is master.

Role information is available within Kubernetes as labels on the nodes.

```
$ kubectl get nodes -L magnum.openstack.org/role
NAME                              STATUS   AGE    VERSION   ROLE
kube-r6cyw4bjb4lr-master-0        Ready    5d5h   v1.16.0   master
kube-r6cyw4bjb4lr-node-0          Ready    5d5h   v1.16.0   worker
kube-test-ng-lg7bkvjgus4y-node-0  Ready    61s    v1.16.0   test
```

This information can be used for scheduling, using a node selector.

```
nodeSelector:
  magnum.openstack.org/role: test
```

The label `magnum.openstack.org/nodegroup` is also available for selecting a specific node group.

Flavor

The node group flavor will default to the minion flavor given when creating the cluster, but can be changed for each new node group.

```
$ openstack coe nodegroup create ef7011bb-d404-4198-a145-e8808204cde3 large-
↪ng --flavor m2.large
```

This can be used if you require nodes of different sizes in the same cluster, or to switch from one flavor to another by creating a new node group and deleting the old one.

Availability zone

To create clusters which span more than one availability zone, multiple node groups must be used. The availability zone is passed as a label to the node group.

```
$ openstack coe nodegroup create kube zone-a --labels availability_zone=zone-
↪a --labels ...
$ openstack coe nodegroup create kube zone-b --labels availability_zone=zone-
↪b --labels ...
$ openstack coe nodegroup create kube zone-c --labels availability_zone=zone-
↪c --labels ...
```

Where `--labels ...` are the rest of the labels that the cluster was created with, which can be obtained from the cluster with this script:

```
$ openstack coe cluster show -f json <CLUSTER_ID> |
    jq --raw-output '.labels | to_entries |
    map("--labels \(.key)=\"\(.value)\"") | join(" ")'
```

Zone information is available within the cluster as the label `topology.kubernetes.io/zone` on each node, or as the now deprecated label `failure-domain.beta.kubernetes.io/zone`.

From Kubernetes 1.16 and onwards it is possible to balance the number of pods in a deployment across availability zones (or any other label).

Resize

Resizing a node group is done with the same API as resizing a cluster, but the `--nodegroup` parameter must be used.

```
$ openstack coe cluster resize kube --nodegroup default-worker 2
Request to resize cluster ef7011bb-d404-4198-a145-e8808204cde3 has been␣
↪accepted.
```

As usual the `--nodes-to-remove` parameter may be used to remove specific nodes when decreasing the size of a node group.

Delete

Any node group except the default master and worker node groups can be deleted, by specifying the cluster and nodegroup name or ID.

```
$ openstack coe nodegroup delete ef7011bb-d404-4198-a145-e8808204cde3 test-ng
```

4.1.25 Kubernetes Health Monitoring

Currently Magnum can support health monitoring for Kubernetes cluster. There are two scenarios supported now: internal and external.

Internal Health Monitoring

Magnum has a periodic job to poll the k8s cluster if it is a reachable cluster. If the floating IP is enabled, or the master loadbalancer is enabled and the master loadbalancer has floating IP associated, then Magnum will take this cluster as reachable. Then Magnum will call the k8s API per 10 seconds to poll the health status of the cluster and then update the two attributes: *health_status* and *health_status_reason*.

External Health Montorning

Currently, only magnum-auto-healer is able to update clusters *health_status* and *health_status_reason* attributes. Both the label *auto_healing_enabled=True* and *auto_healing_controller=magnum-auto-healer* must be set, otherwise, the two attributes value will be overwritten with UNKNOWN and The cluster is not accessible. The health_status attribute can either be in *HEALTHY*, *UNHEALTHY* or *UNKNOWN* and the health_status_reason is a dictionary of the hostnames and their current health statuses and the API health status.

4.2 Container Monitoring in Kubernetes

The current monitoring capabilities that can be deployed with magnum span through different components. These are:

- **metrics-server:** is responsible for the API metrics.k8s.io requests. This includes the most basic functionality when using simple HPA metrics or when using the *kubectl top* command.
- **prometheus:** is a full fledged service that allows the user to access advanced metrics capabilities. These metrics are collected with a resolution of 30 seconds and include resources such as CPU, Memory, Disk and Network IO as well as R/W rates. These metrics of fine granularity are available on your cluster for up to a period of 14 days (default).
- **prometheus-adapter:** is an extra component that integrates with the prometheus service and allows a user to create more sophisticated HPA rules. The service integrates fully with the metrics.k8s.io API but at this time only custom.metrics.k8s.io is being actively used.

The installation of these services is controlled with the following labels:

metrics_server_enabled metrics_server_enabled is used to enable disable the installation of the metrics server. To use this service tiller_enabled must be true when using helm_client_tag<v3.0.0. Train default: true Stein default: true

monitoring_enabled Enable installation of cluster monitoring solution provided by the stable/prometheus-operator helm chart. To use this service tiller_enabled must be true when using helm_client_tag<v3.0.0. Default: false

prometheus_adapter_enabled Enable installation of cluster custom metrics provided by the stable/prometheus-adapter helm chart. This service depends on monitoring_enabled. Default: true

To control deployed versions, extra labels are available:

metrics_server_chart_tag Add metrics_server_chart_tag to select the version of the stable/metrics-server chart to install. Ussuri default: v2.8.8 Yoga default: v3.7.0

prometheus_operator_chart_tag Add prometheus_operator_chart_tag to select version of the stable/prometheus-operator chart to install. When installing the chart, helm will use the default values of the tag defined and overwrite them based on the prometheus-operator-config ConfigMap currently defined. You must certify that the versions are compatible.

prometheus_adapter_chart_tag The stable/prometheus-adapter helm chart version to use. Train-default: 1.4.0

4.2.1 Full fledged cluster monitoring

The prometheus installation provided with the *monitoring_enabled* label is in fact a multi component service. This installation is managed with the prometheus-operator helm chart and the constituent components are:

- **prometheus** (data collection, storage and search)
  - **node-exporter** (data source for the kubelet/node)
  - **kube-state-metrics** (data source for the running kubernetes objects {deployments, pods, nodes, etc})
- **alertmanager** (alarm aggregation, processing and dispatch)
- **grafana** (metrics visualization)

These components are installed in a generic way that makes it easy to have a cluster wide monitoring infrastructure running with no effort.

Warning: The existent monitoring infra does not take into account the existence of nodegroups. If you plan to use nodegroups in your cluster you can take into account the maximum number of total nodes and use *max_node_count* to correctly setup the prometheus server.

Note: Before creating your cluster take into account the scale of the cluster. This is important as the Prometheus server pod might not fit your nodes. This is particularly important if you are using *Cluster Autoscaling* as the Prometheus server will schedule resources needed to meet the maximum number of nodes that your cluster can scale up to defined by label (if existent) *max_node_count*.

The Prometheus server will consume the following resources:

```
RAM:: 256 (base) + Nodes * 40 [MB]
CPU:: 128 (base) + Nodes * 7 [mCPU]
Disk:: 15 GB for 2 weeks (depends on usage)
```

4.2.2 Tuning parameters

The existent setup configurations allows you to tune the metric infrastructure to your requisites. Below is a list of labels that can be used for specific cases:

grafana_admin_passwd This label lets users create their own *admin* user password for the Grafana interface. It expects a string value. Default: admin

monitoring_retention_days This label lets users specify the maximum retention time for data collected in the prometheus server in days. Default: 14

monitoring_interval_seconds This label lets users specify the time between metric samples in seconds. Default: 30

monitoring_retention_size This label lets users specify the maximum size (in gigibytes) for data stored by the prometheus server. This label must be used together with *monitoring_storage_class_name*. Default: 14

monitoring_storage_class_name The kubernetes storage class name to use for the prometheus pvc. Using this label will activate the usage of a pvc instead of local disk space. When using monitoring_storage_class_name 2 pvcs will be created. One for the prometheus server which size is set by *monitoring_retention_size* and one for grafana which is fixed at 1Gi. Default:

monitoring_ingress_enabled This label sets up all the underlying services to be accessible in a route by path way. This means that the services will be exposed as:

```
my.domain.com/alertmanager
my.domain.com/prometheus
my.domain.com/grafana
```

This label must be used together with *cluster_root_domain_name*. Default: false

cluster_root_domain_name The root domain name to use for the cluster automatically set up applications. Default: localhost

cluster_basic_auth_secret The kubernetes secret to use for the proxy basic auth username and password for the unprotected services {alertmanager,prometheus}. Basic auth is only set up if this file is specified. The secret must be in the same namespace as the used proxy (kube-system). Default:

```
To create this secret you can do:
$ htpasswd -c auth foo
$ kubectl create secret generic basic-auth --from-file=auth
```

prometheus_adapter_configmap The name of the prometheus-adapter rules ConfigMap to use. Using this label will overwrite the default rules. Default:

4.3 Glossary

4.3.1 Magnum Terminology

Cluster (previously Bay) A cluster is the construct in which Magnum launches container orchestration engines. After a cluster has been created the user is able to add containers to it either directly, or in the case of the Kubernetes container orchestration engine within pods - a logical construct specific to that implementation. A cluster is created based on a ClusterTemplate.

ClusterTemplate (previously BayModel) A ClusterTemplate in Magnum is roughly equivalent to a flavor in Nova. It acts as a template that defines options such as the container orchestration engine, keypair and image for use when Magnum is creating clusters using the given ClusterTemplate.

Container Orchestration Engine (COE) A container orchestration engine manages the lifecycle of one or more containers, logically represented in Magnum as a cluster. Magnum supports a number of container orchestration engines, each with their own pros and cons, including Docker Swarm, Kubernetes, and Mesos.

Labels Labels is a general method to specify supplemental parameters that are specific to certain COE or associated with certain options. Their format is key/value pair and their meaning is interpreted by the drivers that uses them.

Cluster Drivers A cluster driver is a collection of python code, heat templates, scripts, images, and documents for a particular COE on a particular distro. Magnum presents the concept of ClusterTemplates and clusters. The implementation for a particular cluster type is provided by the cluster driver. In other words, the cluster driver provisions and manages the infrastructure for the COE.

4.3.2 Kubernetes Terminology

Kubernetes uses a range of terminology that we refer to in this guide. We define these common terms for your reference:

Pod When using the Kubernetes container orchestration engine, a pod is the smallest deployable unit that can be created and managed. A pod is a co-located group of application containers that run with a shared context. When using Magnum, pods are created and managed within clusters. Refer to the pods section in Kubernetes Tasks for more information.

Replication controller A replication controller is used to ensure that at any given time a certain number of replicas of a pod are running. Pods are automatically created and deleted by the replication controller as necessary based on a template to ensure that the defined number of replicas exist. Refer to the replication controller section in the Kubernetes Tasks for more information.

Service A service is an additional layer of abstraction provided by the Kubernetes container orchestration engine which defines a logical set of pods and a policy for accessing them. This is useful because pods are created and deleted by a replication controller, for example, other pods needing to discover them can do so via the service abstraction. Refer to the services section in Kubernetes Concepts for more information.

CHAPTER

FIVE

CONTRIBUTOR GUIDE

5.1 Contributors Guide

5.1.1 Getting Started

If you are new to Magnum, this section contains information that should help you get started as a developer working on the project or contributing to the project.

So You Want to Contribute

For general information on contributing to OpenStack, please check out the contributor guide to get started. It covers all the basics that are common to all OpenStack projects: the accounts you need, the basics of interacting with our Gerrit review system, how we communicate as a community, etc.

Below will cover the more project specific information you need to get started with Magnum.

Communication

- IRC channel: #openstack-containers
- Mailing lists prefix: [magnum]
- Currently, we have a weekly team meeting at 9:00 UTC, please check here for more details.

Contacting the Core Team

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

New Feature Planning

Magnum is using a dedicated specs repo for feature requirement.

Task Tracking

We track our tasks in Storyboard

Reporting a Bug

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

Getting Your Patch Merged

Though we have a small number of core reviewers of the Magnum project, we still need two +2 before `Workflow +1`.

Project Team Lead Duties

All common PTL duties are enumerated here in the PTL guide.

Developer Quick-Start

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

See also:

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

Setup Dev Environment

Install OS-specific prerequisites:

```
# Ubuntu Xenial:
sudo apt update
sudo apt install python-dev libssl-dev libxml2-dev curl \
                 libmysqlclient-dev libxslt-dev libpq-dev git \
                 libffi-dev gettext build-essential python3-dev

# CentOS 7:
sudo yum install -y python-devel openssl-devel mariadb-devel curl \
                    libxml2-devel libxslt-devel postgresql-devel git \
                    libffi-devel gettext gcc

# Fedora/RHEL:
sudo yum install python-devel openssl-devel mysql-devel curl \
                 libxml2-devel libxslt-devel postgresql-devel git \
                 libffi-devel gettext gcc

# openSUSE/SLE 12:
```

(continues on next page)

(continued from previous page)

```
sudo zypper install git libffi-devel curl \
                    libmysqlclient-devel libopenssl-devel libxml2-devel \
                    libxslt-devel postgresql-devel python-devel \
                    gettext-runtime
```

Install pip:

```
curl -s https://bootstrap.pypa.io/get-pip.py | sudo python
```

Install common prerequisites:

```
sudo pip install virtualenv flake8 tox testrepository git-review
```

You may need to explicitly upgrade virtualenv if youve installed the one from your OS distribution and it is too old (tox will complain). You can upgrade it individually, if you need to:

```
sudo pip install -U virtualenv
```

Magnum source code should be pulled directly from git:

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

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

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

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

```
# run a specific test for Python 3.7
tox -epy37 -- test_conductor
```

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

```
# run all the Python 3.7 unit tests (in parallel!)
tox -epy37 -- --parallel
```

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

```
tox -epep8
```

To run unit test coverage and check percentage of code covered:

```
tox -e cover
```

To discover and interact with templates, please refer to /user/cluster-type-definition.

Exercising the Services Using DevStack

DevStack can be configured to enable magnum support. It is easy to develop magnum with the DevStack environment. Magnum depends on nova, glance, heat and neutron to create and schedule virtual machines to simulate bare-metal (full bare-metal support is under active development).

Minimum System Requirements

Magnum running in DevStack requires at least: 10 GB RAM, 8 CPU and 100 GB disk storage.

NOTE: Running DevStack within a virtual machine with magnum enabled is not recommended at this time.

This session has only been tested on Ubuntu 16.04 (Xenial) and Fedora 20/21. We recommend users to select one of them if it is possible.

Set-up Environment and Create a Magnum Session

Clone devstack:

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

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

We will run devstack with minimal local.conf settings required to enable magnum, heat, and neutron (neutron is enabled by default in devstack since Kilo, and heat must be enabled by yourself):

```
$ cat > /opt/stack/devstack/local.conf << END
[[local|localrc]]
DATABASE_PASSWORD=password
RABBIT_PASSWORD=password
SERVICE_TOKEN=password
SERVICE_PASSWORD=password
ADMIN_PASSWORD=password
# magnum requires the following to be set correctly
PUBLIC_INTERFACE=eth1

# Enable barbican service and use it to store TLS certificates
# For details https://docs.openstack.org/magnum/latest/user/index.html
↪#transport-layer-security
enable_plugin barbican https://opendev.org/openstack/barbican

enable_plugin heat https://opendev.org/openstack/heat

# Enable magnum plugin after dependent plugins
enable_plugin magnum https://opendev.org/openstack/magnum

# Optional:  uncomment to enable the Magnum UI plugin in Horizon
```

(continues on next page)

(continued from previous page)

```
#enable_plugin magnum-ui https://opendev.org/openstack/magnum-ui

VOLUME_BACKING_FILE_SIZE=20G
END
```

NOTE: Update PUBLIC_INTERFACE as appropriate for your system.

NOTE: Enable heat plugin is necessary.

Optionally, you can enable neutron/lbaas v2 with octavia to create load balancers for multi master clusters:

```
$ cat >> /opt/stack/devstack/local.conf << END
enable_plugin neutron-lbaas https://opendev.org/openstack/neutron-lbaas
enable_plugin octavia https://opendev.org/openstack/octavia

# Disable LBaaS(v1) service
disable_service q-lbaas
# Enable LBaaS(v2) services
enable_service q-lbaasv2
enable_service octavia
enable_service o-cw
enable_service o-hk
enable_service o-hm
enable_service o-api
END
```

Optionally, you can enable ceilometer in devstack. If ceilometer is enabled, magnum will periodically send metrics to ceilometer:

```
$ cat >> /opt/stack/devstack/local.conf << END
enable_plugin ceilometer https://opendev.org/openstack/ceilometer
END
```

If you want to deploy Docker Registry 2.0 in your cluster, you should enable swift in devstack:

```
$ cat >> /opt/stack/devstack/local.conf << END
enable_service s-proxy
enable_service s-object
enable_service s-container
enable_service s-account
END
```

More devstack configuration information can be found at https://docs.openstack.org/devstack/latest/configuration.html

More neutron configuration information can be found at https://docs.openstack.org/devstack/latest/guides/neutron.html

Run devstack:

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

NOTE: This will take a little extra time when the Fedora Atomic micro-OS image is downloaded for the first time.

At this point, two magnum process (magnum-api and magnum-conductor) will be running on devstack screens. If you make some code changes and want to test their effects, just stop and restart magnum-api and/or magnum-conductor.

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

```
. /opt/stack/devstack/openrc admin admin
```

Magnum has been tested with the Fedora Atomic micro-OS and CoreOS. Magnum will likely work with other micro-OS platforms, but each requires individual support in the heat template.

The Fedora Atomic micro-OS image will automatically be added to glance. You can add additional images manually through glance. To verify the image created when installing devstack use:

```
$ openstack image list

+--------------------------------------+---------------------------------+-
↪-------+
| ID                                   | Name                            |␣
↪Status |
+--------------------------------------+---------------------------------+-
↪-------+
| 0bc132b1-ee91-4bd8-b0fd-19deb57fb39f | Fedora-Atomic-27-20180419.0.x86_64 |␣
↪active |
| 7537bbf2-f1c3-47da-97bb-38c09007e146 | cirros-0.3.5-x86_64-disk        |␣
↪active |
+--------------------------------------+---------------------------------+-
↪-------+
```

To list the available commands and resources for magnum, use:

```
openstack help coe
```

To list out the health of the internal services, namely conductor, of magnum, use:

```
$ openstack coe service list

+----+----------------------------------------+------------------+-------+-----
↪-----+-----------------+---------------------------+-----------------------
↪---+
| id | host                                   | binary           | state |␣
↪disabled | disabled_reason | created_at                | updated_at        ␣
↪       |
+----+----------------------------------------+------------------+-------+-----
↪-----+-----------------+---------------------------+-----------------------
↪---+
| 1  | oxy-dev.hq1-0a5a3c02.hq1.abcde.com     | magnum-conductor | up    |    ␣
↪     | -               | 2016-08-31T10:03:36+00:00 | 2016-08-
↪31T10:11:41+00:00 |
```

(continues on next page)

(continued from previous page)

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

Create a keypair for use with the ClusterTemplate:

```
test -f ~/.ssh/id_rsa.pub || ssh-keygen -t rsa -N "" -f ~/.ssh/id_rsa
openstack keypair create --public-key ~/.ssh/id_rsa.pub testkey
```

Check a dns server can resolve a host name properly:

```
dig <server name> @<dns server> +short
```

For example:

```
$ dig www.openstack.org @8.8.8.8 +short
www.openstack.org.cdn.cloudflare.net.
104.20.64.68
104.20.65.68
```

Building a Kubernetes Cluster - Based on Fedora CoreOS

Create a cluster template. This is similar in nature to a flavor and describes to magnum how to construct the cluster. The ClusterTemplate specifies a Fedora CoreOS image so the clusters which use this ClusterTemplate will be based on Fedora CoreOS

```
openstack coe cluster template create k8s-cluster-template \
    --image fedora-coreos-35.20220116.3.0-openstack.x86_64 \
    --keypair testkey \
    --external-network public \
    --dns-nameserver 8.8.8.8 \
    --flavor ds1G \
    --master-flavor ds2G \
    --docker-volume-size 5 \
    --network-driver flannel \
    --docker-storage-driver overlay2 \
    --coe kubernetes
```

Create a cluster. Use the ClusterTemplate name as a template for cluster creation. This cluster will result in one master kubernetes node and one minion node

```
openstack coe cluster create k8s-cluster \
                      --cluster-template k8s-cluster-template \
                      --node-count 1
```

Clusters will have an initial status of CREATE_IN_PROGRESS. Magnum will update the status to CREATE_COMPLETE when it is done creating the cluster. Do not create containers, pods, services, or replication controllers before magnum finishes creating the cluster. They will likely not be created, and may cause magnum to become confused.

The existing clusters can be listed as follows:

```
$ openstack coe cluster list

+--------------------------------------+-------------+------------+-----------
↪---+-----------------+
| uuid                                 | name        | node_count | master_
↪count | status          |
+--------------------------------------+-------------+------------+-----------
↪---------------------+
| 9dccb1e6-02dc-4e2b-b897-10656c5339ce | k8s-cluster | 1          | 1       ␣
↪    | CREATE_COMPLETE |
+--------------------------------------+-------------+------------+-----------
↪---+-----------------+
```

More detailed information for a given cluster is obtained via:

```
openstack coe cluster show k8s-cluster
```

After a cluster is created, you can dynamically add/remove node(s) to/from the cluster by updating the node_count attribute. For example, to add one more node:

```
openstack coe cluster update k8s-cluster replace node_count=2
```

Clusters in the process of updating will have a status of UPDATE_IN_PROGRESS. Magnum will update the status to UPDATE_COMPLETE when it is done updating the cluster.

NOTE: Reducing node_count will remove all the existing pods on the nodes that are deleted. If you choose to reduce the node_count, magnum will first try to remove empty nodes with no pods running on them. If you reduce node_count by more than the number of empty nodes, magnum must remove nodes that have running pods on them. This action will delete those pods. We strongly recommend using a replication controller before reducing the node_count so any removed pods can be automatically recovered on your remaining nodes.

Heat can be used to see detailed information on the status of a stack or specific cluster:

To check the list of all cluster stacks:

```
openstack stack list
```

To check an individual clusters stack:

```
openstack stack show <stack-name or stack_id>
```

Monitoring cluster status in detail (e.g., creating, updating):

```
CLUSTER_HEAT_NAME=$(openstack stack list | \
                    awk "/\sk8s-cluster-/{print \$4}")
echo ${CLUSTER_HEAT_NAME}
openstack stack resource list ${CLUSTER_HEAT_NAME}
```

Building a Kubernetes Cluster - Based on Fedora Atomic [DEPRECATED]

Fedora Atomic Deprecation Notice:

> Fedora CoreOS is the official successor to Fedora Atomic Host. The last Fedora Atomic Host release was version 29, which has now reached end-of-life.

When building devstack from master, the Fedora atomic image is no longer created for us by default. We will need to create an image ourselves.

```
wget https://dl.fedoraproject.org/pub/alt/atomic/stable/Fedora-Atomic-27-
↪20180419.0/CloudImages/x86_64/images/Fedora-Atomic-27-20180419.0.x86_64.
↪qcow2

openstack image create Fedora-Atomic-27-20180419.0.x86_64  \
                       --public \
                       --disk-format=qcow2 \
                       --container-format=bare \
                       --property os_distro=fedora-atomic \
                       --file=Fedora-Atomic-27-20180419.0.x86_64.qcow2
```

Create a ClusterTemplate. This is similar in nature to a flavor and describes to magnum how to construct the cluster. The ClusterTemplate specifies a Fedora Atomic image so the clusters which use this ClusterTemplate will be based on Fedora Atomic

```
openstack coe cluster template create k8s-cluster-template \
                       --image Fedora-Atomic-27-20180419.0.x86_64 \
                       --keypair testkey \
                       --external-network public \
                       --dns-nameserver 8.8.8.8 \
                       --flavor m1.small \
                       --docker-volume-size 5 \
                       --network-driver flannel \
                       --coe kubernetes
```

Create a cluster. Use the ClusterTemplate name as a template for cluster creation. This cluster will result in one master kubernetes node and one minion node:

```
openstack coe cluster create k8s-cluster \
                      --cluster-template k8s-cluster-template \
                      --node-count 1
```

Building a Kubernetes Cluster - Based on CoreOS [DEPRECATED]

End-of-life announcement for CoreOS Container Linux:

> On May 26, 2020, CoreOS Container Linux will reach its end of life and will no longer receive updates. We strongly recommend that users begin migrating their workloads to another operating system as soon as possible. [] Fedora CoreOS is the official successor to CoreOS Container Linux

You can create a Kubernetes cluster based on CoreOS as an alternative to Atomic or Fedora CoreOS. First, download the official CoreOS image:

```
wget http://beta.release.core-os.net/amd64-usr/current/coreos_production_
↪openstack_image.img.bz2
bunzip2 coreos_production_openstack_image.img.bz2
```

Upload the image to glance:

```
openstack image create CoreOS  \
                    --public \
                    --disk-format=qcow2 \
                    --container-format=bare \
                    --property os_distro=coreos \
                    --file=coreos_production_openstack_image.img
```

Create a CoreOS Kubernetes ClusterTemplate, which is similar to the Atomic Kubernetes ClusterTemplate, except for pointing to a different image:

```
openstack coe cluster template create k8s-cluster-template-coreos \
                      --image CoreOS \
                      --keypair testkey \
                      --external-network public \
                      --dns-nameserver 8.8.8.8 \
                      --flavor m1.small \
                      --network-driver flannel \
                      --coe kubernetes
```

Create a CoreOS Kubernetes cluster. Use the CoreOS ClusterTemplate as a template for cluster creation:

```
openstack coe cluster create k8s-cluster \
                  --cluster-template k8s-cluster-template-coreos \
                  --node-count 2
```

Using a Kubernetes Cluster

NOTE: For the following examples, only one minion node is required in the k8s cluster created previously.

Kubernetes provides a number of examples you can use to check that things are working. You may need to download kubectl binary for interacting with k8s cluster using:

```
curl -LO https://storage.googleapis.com/kubernetes-release/release/v1.2.0/bin/
↪linux/amd64/kubectl
chmod +x ./kubectl
sudo mv ./kubectl /usr/local/bin/kubectl
```

We first need to setup the certs to allow Kubernetes to authenticate our connection. Please refer to *Transport Layer Security* for more info on using TLS keys/certs which are setup below.

To generate an RSA key, you will use the genrsa command of the openssl tool.:

```
openssl genrsa -out client.key 4096
```

To generate a CSR for client authentication, openssl requires a config file that specifies a few values.:

```
$ cat > client.conf << END
[req]
distinguished_name = req_distinguished_name
req_extensions     = req_ext
prompt = no
[req_distinguished_name]
CN = admin
O = system:masters
OU=OpenStack/Magnum
C=US
ST=TX
L=Austin
[req_ext]
extendedKeyUsage = clientAuth
END
```

Once you have client.conf, you can run the openssl req command to generate the CSR.:

```
openssl req -new -days 365 \
    -config client.conf \
    -key client.key \
    -out client.csr
```

Now that you have your client CSR, you can use the Magnum CLI to send it off to Magnum to get it signed and also download the signing cert.:

```
magnum ca-sign --cluster k8s-cluster --csr client.csr > client.crt
magnum ca-show --cluster k8s-cluster > ca.crt
```

Heres how to set up the replicated redis example. Now we create a pod for the redis-master:

```
# Using cluster-config command for faster configuration
eval $(openstack coe cluster config k8s-cluster)

# Test the cert and connection works
kubectl version

cd kubernetes/examples/redis
kubectl create -f ./redis-master.yaml
```

Now create a service to provide a discoverable endpoint for the redis sentinels in the cluster:

```
kubectl create -f ./redis-sentinel-service.yaml
```

To make it a replicated redis cluster create replication controllers for the redis slaves and sentinels:

```
sed -i 's/\(replicas: \)1/\1 2/' redis-controller.yaml
kubectl create -f ./redis-controller.yaml
```

(continues on next page)

(continued from previous page)

```
sed -i 's/\(replicas: \)1/\1 2/' redis-sentinel-controller.yaml
kubectl create -f ./redis-sentinel-controller.yaml
```

Full lifecycle and introspection operations for each object are supported. For example, openstack coe cluster create, openstack coe cluster template delete.

Now there are four redis instances (one master and three slaves) running across the cluster, replicating data between one another.

Run the openstack coe cluster show command to get the IP of the cluster host on which the redis-master is running:

```
$ openstack coe cluster show k8s-cluster

+--------------------+------------------------------------------------------
↪----+
| Property           | Value                                                ␣
↪    |
+--------------------+------------------------------------------------------
↪----+
| status             | CREATE_COMPLETE                                      ␣
↪    |
| uuid               | cff82cd0-189c-4ede-a9cb-2c0af6997709                 ␣
↪    |
| stack_id           | 7947844a-8e18-4c79-b591-ecf0f6067641                 ␣
↪    |
| status_reason      | Stack CREATE completed successfully                  ␣
↪    |
| created_at         | 2016-05-26T17:45:57+00:00                            ␣
↪    |
| updated_at         | 2016-05-26T17:50:02+00:00                            ␣
↪    |
| create_timeout     | 60                                                   ␣
↪    |
| api_address        | https://172.24.4.4:6443                              ␣
↪    |
| coe_version        | v1.2.0                                               ␣
↪    |
| cluster_template_id| e73298e7-e621-4d42-b35b-7a1952b97158                 ␣
↪    |
| master_addresses   | ['172.24.4.6']                                       ␣
↪    |
| node_count         | 1                                                    ␣
↪    |
| node_addresses     | ['172.24.4.5']                                       ␣
↪    |
| master_count       | 1                                                    ␣
↪    |
| container_version  | 1.9.1                                                ␣
↪    |
```

(continues on next page)

(continued from previous page)

```
| discovery_url      | https://discovery.etcd.io/
↪4caaa65f297d4d49ef0a085a7aecf8e0 |
| name               | k8s-cluster                                                    ␣
↪    |
+--------------------+----------------------------------------------------------
↪----+
```

The output here indicates the redis-master is running on the cluster host with IP address 172.24.4.5. To access the redis master:

```
$ ssh fedora@172.24.4.5
$ REDIS_ID=$(sudo docker ps | grep redis:v1 | grep k8s_master | awk '{print
↪$1}')
$ sudo docker exec -i -t $REDIS_ID redis-cli

127.0.0.1:6379> set replication:test true
OK
^D

$ exit  # Log out of the host
```

Log into one of the other container hosts and access a redis slave from it. You can use *nova list* to enumerate the kube-minions. For this example we will use the same host as above:

```
$ ssh fedora@172.24.4.5
$ REDIS_ID=$(sudo docker ps | grep redis:v1 | grep k8s_redis | awk '{print $1}
↪')
$ sudo docker exec -i -t $REDIS_ID redis-cli

127.0.0.1:6379> get replication:test
"true"
^D

$ exit  # Log out of the host
```

Additional useful commands from a given minion:

```
sudo docker ps  # View Docker containers on this minion
kubectl get pods  # Get pods
kubectl get rc  # Get replication controllers
kubectl get svc  # Get services
kubectl get nodes  # Get nodes
```

After you finish using the cluster, you want to delete it. A cluster can be deleted as follows:

```
openstack coe cluster delete k8s-cluster
```

Building and Using a Swarm Cluster

Create a ClusterTemplate. It is very similar to the Kubernetes ClusterTemplate, except for the absence of some Kubernetes-specific arguments and the use of swarm as the COE:

```
openstack coe cluster template create swarm-cluster-template \
                      --image Fedora-Atomic-27-20180419.0.x86_64 \
                      --keypair testkey \
                      --external-network public \
                      --dns-nameserver 8.8.8.8 \
                      --flavor m1.small \
                      --docker-volume-size 5 \
                      --coe swarm-mode
```

NOTE: If you are using Magnum behind a firewall then refer to *Using Proxies in magnum if running under firewall*.

Finally, create the cluster. Use the ClusterTemplate swarm-cluster-template as a template for cluster creation. This cluster will result in one swarm manager node and two extra agent nodes:

```
openstack coe cluster create swarm-cluster \
                     --cluster-template swarm-cluster-template \
                     --node-count 2
```

Now that we have a swarm cluster we can start interacting with it:

```
$ openstack coe cluster show swarm-cluster

+--------------------+------------------------------------------------------------
↪----+
| Property           | Value                                                      ␣
↪    |
+--------------------+------------------------------------------------------------
↪----+
| status             | CREATE_COMPLETE                                            ␣
↪    |
| uuid               | eda91c1e-6103-45d4-ab09-3f316310fa8e                       ␣
↪    |
| stack_id           | 7947844a-8e18-4c79-b591-ecf0f6067641                       ␣
↪    |
| status_reason      | Stack CREATE completed successfully                        ␣
↪    |
| created_at         | 2015-04-20T19:05:27+00:00                                  ␣
↪    |
| updated_at         | 2015-04-20T19:06:08+00:00                                  ␣
↪    |
| create_timeout     | 60                                                         ␣
↪    |
| api_address        | https://172.24.4.4:6443                                    ␣
↪    |
| coe_version        | 1.2.5                                                      ␣
↪    |
```

(continues on next page)

(continued from previous page)

```
| cluster_template_id| e73298e7-e621-4d42-b35b-7a1952b97158                 ␣
↪    |
| master_addresses   | ['172.24.4.6']                                       ␣
↪    |
| node_count         | 2                                                    ␣
↪    |
| node_addresses     | ['172.24.4.5']                                       ␣
↪    |
| master_count       | 1                                                    ␣
↪    |
| container_version  | 1.9.1                                                ␣
↪    |
| discovery_url      | https://discovery.etcd.io/
↪4caaa65f297d4d49ef0a085a7aecf8e0 |
| name               | swarm-cluster                                        ␣
↪    |
+--------------------+------------------------------------------------------
↪----+
```

We now need to setup the docker CLI to use the swarm cluster we have created with the appropriate credentials.

Create a dir to store certs and cd into it. The *DOCKER_CERT_PATH* env variable is consumed by docker which expects ca.pem, key.pem and cert.pem to be in that directory.:

```
export DOCKER_CERT_PATH=~/.docker
mkdir -p ${DOCKER_CERT_PATH}
cd ${DOCKER_CERT_PATH}
```

Generate an RSA key.:

```
openssl genrsa -out key.pem 4096
```

Create openssl config to help generated a CSR.:

```
$ cat > client.conf << END
[req]
distinguished_name = req_distinguished_name
req_extensions     = req_ext
prompt = no
[req_distinguished_name]
CN = Your Name
[req_ext]
extendedKeyUsage = clientAuth
END
```

Run the openssl req command to generate the CSR.:

```
openssl req -new -days 365 \
    -config client.conf \
```

(continues on next page)

(continued from previous page)

```
    -key key.pem \
    -out client.csr
```

Now that you have your client CSR use the Magnum CLI to get it signed and also download the signing cert.:

```
magnum ca-sign --cluster swarm-cluster --csr client.csr > cert.pem
magnum ca-show --cluster swarm-cluster > ca.pem
```

Set the CLI to use TLS . This env var is consumed by docker.:

```
export DOCKER_TLS_VERIFY="1"
```

Set the correct host to use which is the public ip address of swarm API server endpoint. This env var is consumed by docker.:

```
export DOCKER_HOST=$(openstack coe cluster show swarm-cluster | awk '/ api_
↪address /{print substr($4,7)}')
```

Next we will create a container in this swarm cluster. This container will ping the address 8.8.8.8 four times:

```
docker run --rm -it cirros:latest ping -c 4 8.8.8.8
```

You should see a similar output to:

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

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

Building and Using a Mesos Cluster

Provisioning a mesos cluster requires a Ubuntu-based image with some packages pre-installed. To build and upload such image, please refer to :ref‘building_mesos_image‘.

Alternatively, you can download and upload a pre-built image:

```
wget https://fedorapeople.org/groups/magnum/ubuntu-mesos-latest.qcow2
openstack image create ubuntu-mesos --public \
                    --disk-format=qcow2 --container-format=bare \
                    --property os_distro=ubuntu --file=ubuntu-mesos-latest.
↪qcow2
```

Then, create a ClusterTemplate by using mesos as the COE, with the rest of arguments similar to the Kubernetes ClusterTemplate:

```
openstack coe cluster template create mesos-cluster-template --image ubuntu-
↪mesos \
                      --keypair testkey \
                      --external-network public \
                      --dns-nameserver 8.8.8.8 \
                      --flavor m1.small \
                      --coe mesos
```

Finally, create the cluster. Use the ClusterTemplate mesos-cluster-template as a template for cluster creation. This cluster will result in one mesos master node and two mesos slave nodes:

```
openstack coe cluster create mesos-cluster \
                      --cluster-template mesos-cluster-template \
                      --node-count 2
```

Now that we have a mesos cluster we can start interacting with it. First we need to make sure the clusters status is CREATE_COMPLETE:

```
$ openstack coe cluster show mesos-cluster

+--------------------+------------------------------------------------------
↪----+
| Property           | Value                                                ␣
↪    |
+--------------------+------------------------------------------------------
↪----+
| status             | CREATE_COMPLETE                                      ␣
↪    |
| uuid               | ff727f0d-72ca-4e2b-9fef-5ec853d74fdf                 ␣
↪    |
| stack_id           | 7947844a-8e18-4c79-b591-ecf0f6067641                 ␣
↪    |
| status_reason      | Stack CREATE completed successfully                  ␣
↪    |
| created_at         | 2015-06-09T20:21:43+00:00                            ␣
↪    |
| updated_at         | 2015-06-09T20:28:18+00:00                            ␣
↪    |
| create_timeout     | 60                                                   ␣
↪    |
| api_address        | https://172.24.4.115:6443                            ␣
↪    |
| coe_version        | -                                                    ␣
↪    |
| cluster_template_id| 92dbda62-32d4-4435-88fc-8f42d514b347                 ␣
↪    |
| master_addresses   | ['172.24.4.115']                                     ␣
↪    |
| node_count         | 2                                                    ␣
↪    |
```

(continues on next page)

(continued from previous page)

```
| node_addresses     | ['172.24.4.116', '172.24.4.117']                          ␣
↪    |
| master_count       | 1                                                         ␣
↪    |
| container_version  | 1.9.1                                                     ␣
↪    |
| discovery_url      | None                                                      ␣
↪    |
| name               | mesos-cluster                                             ␣
↪    |
+--------------------+-----------------------------------------------------------
↪----+
```

Next we will create a container in this cluster by using the REST API of Marathon. This container will ping the address 8.8.8.8:

```
$ cat > mesos.json << END
{
  "container": {
    "type": "DOCKER",
    "docker": {
      "image": "cirros"
    }
  },
  "id": "ubuntu",
  "instances": 1,
  "cpus": 0.5,
  "mem": 512,
  "uris": [],
  "cmd": "ping 8.8.8.8"
}
END
$ MASTER_IP=$(openstack coe cluster show mesos-cluster | awk '/ api_address /
↪{print $4}')
$ curl -X POST -H "Content-Type: application/json" \
    http://${MASTER_IP}:8080/v2/apps -d@mesos.json
```

To check application and task status:

```
$ curl http://${MASTER_IP}:8080/v2/apps
$ curl http://${MASTER_IP}:8080/v2/tasks
```

You can access to the Mesos web page at http://<master>:5050/ and Marathon web console at http://<master>:8080/.

Building Developer Documentation

To build the documentation locally (e.g., to test documentation changes before uploading them for review) chdir to the magnum root folder and run tox:

```
tox -edocs
```

NOTE: The first time you run this will take some extra time as it creates a virtual environment to run in.

When complete, the documentation can be accessed from:

```
doc/build/html/index.html
```

Running functional tests

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

Prerequisite

You need to have a Magnum instance running somewhere. If you are using devstack, follow *Developer Quick-Start* to deploy Magnum in a devstack environment.

Configuration

The functional tests require a couple configuration files, so youll need to generate them yourself.

For devstack

If youre using devstack, you can copy and modify the devstack configuration:

```
cd /opt/stack/magnum
cp /opt/stack/tempest/etc/tempest.conf /opt/stack/magnum/etc/tempest.conf
cp functional_creds.conf.sample functional_creds.conf

# update the IP address
HOST=$(iniget /etc/magnum/magnum.conf api host)
sed -i "s/127.0.0.1/$HOST/" functional_creds.conf

# update admin password
. /opt/stack/devstack/openrc admin admin
iniset functional_creds.conf admin pass $OS_PASSWORD

# update demo password
. /opt/stack/devstack/openrc demo demo
iniset functional_creds.conf auth password $OS_PASSWORD
```

Set the DNS name server to be used by your cluster nodes (e.g. 8.8.8.8):

```
# update DNS name server
. /opt/stack/devstack/openrc demo demo
iniset functional_creds.conf magnum dns_nameserver <dns-svr-ip-address>
```

Create the necessary keypair and flavor:

```
. /opt/stack/devstack/openrc admin admin
openstack keypair create --public-key ~/.ssh/id_rsa.pub  default
openstack flavor create --id 100 --ram 1024 --disk 10 --vcpus 1 m1.magnum
openstack flavor create --id 200 --ram 512 --disk 10 --vcpus 1 s1.magnum

. /opt/stack/devstack/openrc demo demo
openstack keypair create --public-key ~/.ssh/id_rsa.pub  default
```

You may need to explicitly upgrade required packages if youve installed them before and their versions become too old:

```
UPPER_CONSTRAINTS=/opt/stack/requirements/upper-constraints.txt
sudo pip install -c $UPPER_CONSTRAINTS -U -r test-requirements.txt
```

Outside of devstack

If you are not using devstack, youll need to create the configuration files. The /etc/tempest.conf configuration file is documented here

https://docs.openstack.org/tempest/latest/configuration.html#tempest-configuration

Heres a reasonable sample of tempest.conf settings you might need:

```
[auth]
use_dynamic_credentials=False
test_accounts_file=/tmp/etc/magnum/accounts.yaml
admin_username=admin
admin_password=password
admin_project_name=admin

[identity]
disable_ssl_certificate_validation=True
uri=https://identity.example.com/v2.0
auth_version=v2
region=EAST

[identity-feature-enabled]
api_v2 = true
api_v3 = false
trust = false

[oslo_concurrency]
lock_path = /tmp/
```

(continues on next page)

(continued from previous page)

```
[magnum]
image_id=22222222-2222-2222-2222-222222222222
nic_id=11111111-1111-1111-1111-111111111111
keypair_id=default
flavor_id=small
magnum_url=https://magnum.example.com/v1

[debug]
trace_requests=true
```

A sample functional_creds.conf can be found in the root of this project named functional_creds.conf.sample

When you run tox, be sure to specify the location of your tempest.conf using TEMPEST_CONFIG_DIR:

```
export TEMPEST_CONFIG_DIR=/tmp/etc/magnum/
tox -e functional-api
```

Execution

Magnum has different functional tests for each COE and for the API. All the environments are detailed in Magnums tox.ini:

```
cat tox.ini | grep functional- | awk -F: '{print $2}' | sed s/]//
```

To run a particular subset of tests, specify that group as a tox environment. For example, here is how you would run all of the kubernetes tests:

```
tox -e functional-k8s
```

To run a specific test or group of tests, specify the test path as a positional argument:

```
tox -e functional-k8s -- magnum.tests.functional.k8s.v1.test_k8s_python_
↪client.TestBayModelResource
```

To avoid creating multiple clusters simultaneously, you can execute the tests with concurrency 1:

```
tox -e functional-swarm -- --concurrency 1
```

Developer Troubleshooting Guide

This guide is intended to provide information on how to resolve common problems encountered when developing code for magnum.

Troubleshooting MySQL

When creating alembic migrations, developers might encounter the `Multiple head revisions are present for given argument 'head'` error.

This can occur when two migrations revise the same head. For example, the developer creates a migration locally but another migration has already been accepted and merged into master that revises the same head:

```
$ alembic heads
12345 (your local head)
67890 (new master head)
```

In order to fix this, the developer should update the down_revision of their local migration to point to the head of the new migration in master:

```
# revision identifiers, used by Alembic.
revision = '12345'
down_revision = '67890'
```

Now the newest local migration should be head:

```
$ alembic heads
12345 (your local head)
```

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

Release Notes

What is reno ?

Magnum uses reno for providing release notes in-tree. That means that a patch can include a *reno file* or a series can have a follow-on change containing that file explaining what the impact is.

A *reno file* is a YAML file written in the releasenotes/notes tree which is generated using the reno tool this way:

```
$ tox -e venv -- reno new <name-your-file>
```

where usually `<name-your-file>` can be `bp-<blueprint_name>` for a blueprint or `bug-XXXXXX` for a bugfix.

Refer to the reno documentation for the full list of sections.

When a release note is needed

A release note is required anytime a reno section is needed. Below are some examples for each section. Any sections that would be blank should be left out of the note file entirely. If no section is needed, then you know you dont need to provide a release note :-)

- **upgrade**
  - The patch has an UpgradeImpact tag
  - A DB change needs some deployer modification (like a migration)
  - A configuration option change (deprecation, removal or modified default)
  - some specific changes that have a DocImpact tag but require further action from an deployer perspective
  - any patch that requires an action from the deployer in general
- **security**
  - If the patch fixes a known vulnerability
- **features**
  - If the patch has an APIImpact tag
- **critical**
  - Bugfixes categorized as Critical in Launchpad *impacting users*
- **fixes**
  - No clear definition of such bugfixes. Hairy long-standing bugs with high importance that have been fixed are good candidates though.

Three sections are left intentionally unexplained (`prelude`, `issues` and `other`). Those are targeted to be filled in close to the release time for providing details about the soon-ish release. Dont use them unless you know exactly what you are doing.

API Microversions

Background

Magnum uses a framework we call API Microversions for allowing changes to the API while preserving backward compatibility. The basic idea is that a user has to explicitly ask for their request to be treated with a particular version of the API. So breaking changes can be added to the API without breaking users who dont specifically ask for it. This is done with an HTTP header `OpenStack-API-Version` which has as its value a string containing the name of the service, `container-infra`, and a monotonically increasing semantic version number starting from `1.1`. The full form of the header takes the form:

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

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

When do I need a new Microversion?

A microversion is needed when the contract to the user is changed. The user contract covers many kinds of information such as:

- the Request
  - the list of resource urls which exist on the server

    Example: adding a new clusters/{ID}/foo which didnt exist in a previous version of the code
  - the list of query parameters that are valid on urls

    Example: adding a new parameter `is_yellow` clusters/{ID}?is_yellow=True
  - the list of query parameter values for non free form fields

    Example: parameter filter_by takes a small set of constants/enums A, B, C. Adding support for new enum D.
  - new headers accepted on a request
  - the list of attributes and data structures accepted.

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

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

    Example: adding a new allowed `status` to clusters/{ID}
  - the list of status codes allowed for a particular request

    Example: an API previously could return 200, 400, 403, 404 and the change would make the API now also be allowed to return 409.

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

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

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

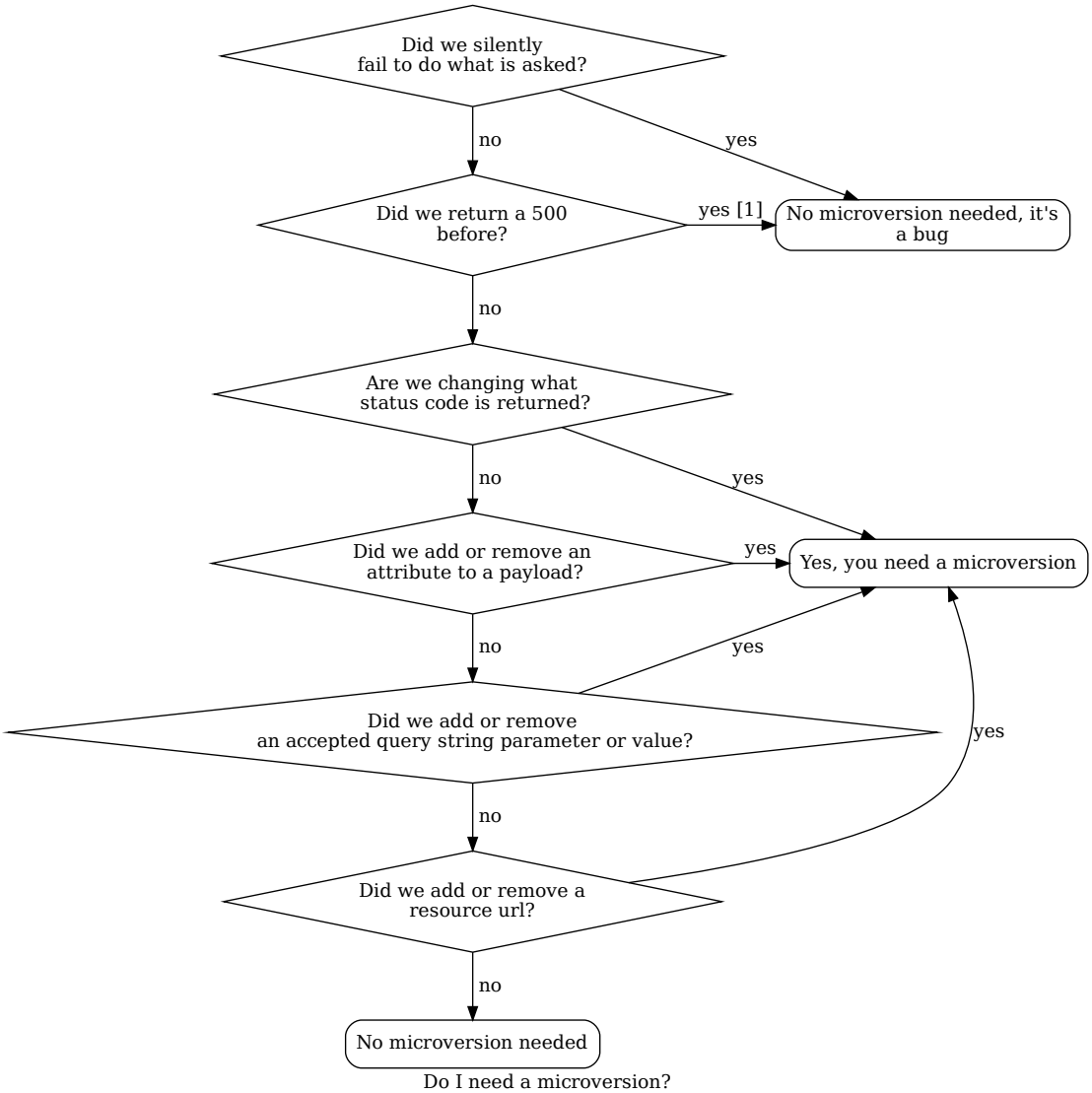
[2] The exception to not needing a microversion when returning a previously unspecified error code is the 400, 403, 404 and 415 cases. This is considered OK to return even if previously unspecified in the code since its implied given keystone authentication can fail with a 403 and API validation can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

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

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

- new headers returned on a response

The following flow chart attempts to walk through the process of do we need a microversion.



Footnotes

[1] When fixing 500 errors that previously caused stack traces, try to map the new error into the existing set of errors that API call could previously return (400 if nothing else is appropriate). Changing the set of allowed status codes from a request is changing the contract, and should be part of a microversion (except in[7]).

The reason why we are so strict on contract is that wed like application writers to be able to know, for sure, what the contract is at every microversion in Magnum. If they do not, they will need to write conditional code in their application to handle ambiguities.

When in doubt, consider application authors. If it would work with no client side changes on both Magnum versions, you probably dont need a microversion. If, on the other hand, there is any ambiguity, a microversion is probably needed.

When a microversion is not needed

A microversion is not needed in the following situation:

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

In Code

In `magnum/api/controllers/base.py` we define an `@api_version` decorator which is intended to be used on top-level Controller methods. It is not appropriate for lower-level methods. Some examples:

Adding a new API method

In the controller class:

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

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

Removing an API method

In the controller class:

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

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

Changing a methods behavior

In the controller class:

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

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

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

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

When not using decorators

When you dont want to use the `@api_version` decorator on a method or you want to change behavior within a method (say it leads to simpler or simply a lot less code) you can directly test for the requested version with a method as long as you have access to the api request object (commonly accessed with `pecan.request`). Every API method has an versions object attached to the request object and that can be used to modify behavior based on its value:

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

    req_version = pecan.request.headers.get(Version.string)
    req1_min = versions.Version("1.1")
    req1_max = versions.Version("1.5")
    req2_min = versions.Version("1.6")
    req2_max = versions.Version("1.10")

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

    <common code>
```

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

Other necessary changes

If you are adding a patch which adds a new microversion, it is necessary to add changes to other places which describe your change:

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

Allocating a microversion

If you are adding a patch which adds a new microversion, it is necessary to allocate the next microversion number. Except under extremely unusual circumstances and this would have been mentioned in the magnum spec for the change, the minor number of `CURRENT_MAX_VER` will be incremented. This will also be the new microversion number for the API change.

It is possible that multiple microversion patches would be proposed in parallel and the microversions would conflict between patches. This will cause a merge conflict. We dont reserve a microversion for each patch in advance as we dont know the final merge order. Developers may need over time to rebase their patch calculating a new version number as above based on the updated value of `CURRENT_MAX_VER`.

Versioned Objects

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

Object Version Testing

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

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

Object Change Example

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

```
tox -e py37 magnum.tests.unit.objects.test_objects
```

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

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

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

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

```
@base.MagnumObjectRegistry.register
class Cluster(base.MagnumPersistentObject, base.MagnumObject,
              base.MagnumObjectDictCompat):
    # Version 1.0: Initial version
    # Version 1.1: Added 'foo' field
    VERSION = '1.1'
```

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

```
testtools.matchers._impl.MismatchError: !=:
reference = {'Cluster': '1.0-35edde13ad178e9419e7ea8b6d580bcd'}
actual    = {'Cluster': '1.1-22b40e8eed0414561ca921906b189820'}
```

I can now copy the new fingerprint needed (1.1-22b40e8eed0414561ca921906b189820), to the object_data map within magnum/tests/unit/objects/test_objects.py:

```
object_data = {
    'Cluster': '1.1-22b40e8eed0414561ca921906b189820',
    'ClusterTemplate': '1.0-06863f04ab4b98307e3d1b736d3137bf',
    'Certificate': '1.0-69b579203c6d726be7878c606626e438',
    'MyObj': '1.0-b43567e512438205e32f4e95ca616697',
    'X509KeyPair': '1.0-fd008eba0fbc390e0e5da247bba4eedd',
    'MagnumService': '1.0-d4b8c0f3a234aec35d273196e18f7ed1',
}
```

Running the unit tests now shows no failure.

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

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

REST API Version History

This documents the changes made to the REST API with every microversion change. The description for each version should be a verbose one which has enough information to be suitable for use in user documentation.

1.1

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

A user can specify a header in the API request:

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

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

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

1.2

Support for async cluster (previously known as bay) operations

Before v1.2 all magnum bay operations were synchronous and as a result API requests were blocked until response from HEAT service is received. With this change cluster-create/bay-create, cluster-update/bay-update and cluster-delete/bay-delete calls will be asynchronous.

1.3

Rollback cluster (previously known as bay) on update failure

User can enable rollback on bay update failure by specifying microversion 1.3 in header({OpenStack-API-Version: container-infra 1.3}) and passing rollback=True when issuing cluster/bay update request. For example:- - http://XXX/v1/clusters/XXX/?rollback=True or - http://XXX/v1/bays/XXX/?rollback=True

1.4

Add stats API

An admin user can get total number of clusters and nodes for a specified tenant or for all the tenants and also a non-admin user can get self stats. For example:- - http://XXX/v1/stats or - http://XXX/v1/stats?project_id=<project-id> or - http://XXX/v1/stats?project_id=<project-id>&type=<stats-type>

1.5

Support for cluster CA certificate rotation

This gives admins a way to revoke access to an existing cluster once a user has been granted access.

1.6

Add quotas API

An admin user can set/update/delete/list quotas for the given tenant. A non-admin user can get self quota information.

1.7

Add resize API

1.8

Add upgrade API

1.9

Add nodegroup API

Allow create/update/delete/list of default-worker and additional nodegroups.

1.10

Allow nodegroups with 0 nodes

Allow the cluster to be created with node_count = 0 as well as to update existing nodegroups to have 0 nodes.

Magnum Development Policies

Contents

Magnum is made possible by a wide base of contributors from numerous countries and time zones around the world. We work as a team in accordance with the Guiding Principles of the OpenStack Community. We all want to be valued members of a successful team on an inspiring mission. Code contributions are merged into our code base through a democratic voting process. Anyone may vote on patches submitted by our contributors, and everyone is encouraged to make actionable and helpful suggestions for how patches can be improved prior to merging. We strive to strike a sensible balance between the speed of

our work, and the quality of each contribution. This document describes the correct balance in accordance with the prevailing wishes of our team.

This document is an extension of the OpenStack Governance that explicitly converts our tribal knowledge into a codified record. If any conflict is discovered between the OpenStack governance, and this document, the OpenStack documents shall prevail.

Team Responsibilities

Responsibilities for Everyone

Everyone in our community is expected to know and comply with the OpenStack Community Code of Conduct. We all need to work together to maintain a thriving team that enjoys working together to solve challenges.

Responsibilities for Contributors

When making contributions to any Magnum code repository, contributors shall expect their work to be peer reviewed. See *Merge Criteria* for details about how reviewed code is approved for merge.

Expect reviewers to vote against merging a patch, along with actionable suggestions for improvement prior to merging the code. Understand that such a vote is normal, and is essential to our quality process.

If you receive votes against your review submission, please revise your work in accordance with any requests, or leave comments indicating why you believe the work should be further considered without revision.

If you leave your review without further comments or revision for an extended period, you should mark your patch as *Abandoned*, or it may be marked as *Abandoned* by another team member as a courtesy to you. A patch with no revisions for multiple weeks should be abandoned, or changed to work in progress (WIP) with the *workflow-1* flag. We want all code in the review queue to be actionable by reviewers. Note that an *Abandoned* status shall be considered temporary, and that your patch may be restored and revised if and when you are ready to continue working on it. Note that a core reviewer may un-abandon a patch to allow subsequent revisions by you or another contributor, as needed.

When making revisions to patches, please acknowledge and confirm each previous review comment as Done or with an explanation for why the comment was not addressed in your subsequent revision.

Summary of Contributor Responsibilities

- Includes the *Everyone* responsibilities, plus:
- Recognize that revisions are a normal part of our review process.
- Make revisions to your patches to address reviewer comments.
- Mark each inline comment as *Done* once it has been addressed.
- Indicate why any requests have not been acted upon.
- Set *workflow-1* until a patch is ready for merge consideration.
- Consider patches without requested revisions as abandoned after a few weeks.

Responsibilities for Reviewers

Each reviewer is responsible for upholding the quality of our code. By making constructive and actionable requests for revisions to patches, together we make better software. When making requests for revisions, each reviewer shall carefully consider our aim to merge contributions in a timely manner, while improving them. **Contributions do not need to be perfect in order to be merged.** You may make comments with a 0 vote to call out stylistic preferences that will not result in a material change to the software if/when resolved.

If a patch improves our code but has been through enough revisions that delaying it further is worse than including it now in imperfect form, you may file a tech-debt bug ticket against the code, and vote to merge the imperfect patch.

When a reviewer requests a revision to a patch, he or she is expected to review the subsequent revision to verify the change addressed the concern.

Summary of Reviewer Responsibilities

- Includes the Everyone responsibilities, plus:
- Uphold quality of our code.
- Provide helpful and constructive requests for patch revisions.
- Carefully balance need to keep moving while improving contributions.
- Submit tech-debt bugs to merge imperfect code with known problems.
- Review your requested revisions to verify them.

Responsibilities for Core Reviewers

Core reviewers have all the responsibilities mentioned above, as well as a responsibility to judge the readiness of a patch for merge, and to set the *workflow+1* flag to order a patch to be merged once at least one other core reviewers has issued a +2 vote. See: *Merge Criteria*.

Reviewers who use the -2 vote shall:

1. Explain what scenarios can/will lift the -2 or downgrade it to a -1 (non-sticky), or explain this is unmergable for reason <X>. Non-negotiable reasons such as breaks API contract, or introduces fundamental security issues are acceptable.
2. Recognize that a -2 needs more justification than a -1 does. Both require actionable notes, but a -2 comment shall outline the reason for the sticky vote rather than a -1.
3. Closely monitor comments and revisions to that review so the vote is promptly downgraded or removed once addressed by the contributor.

All core reviewers shall be responsible for setting a positive and welcoming tone toward other reviewers and contributors.

Summary of Core Reviewer Responsibilities

- Includes the Reviewer responsibilities, plus:
- Judge readiness of patches for merge.
- Approve patches for merge when requirements are met.
- Set a positive and welcoming tone toward other reviewers and contributors.

PTL Responsibilities

In accordance with our Project Team Guide for PTLs our PTL carries all the responsibilities referenced above plus:

- Select and target blueprints for each release cycle.
- Determine Team Consensus. Resolve disagreements among our team.
- May delegate his/her responsibilities to others.
- **Add and remove core reviewers in accordance with his/her judgement.**
  - Note that in accordance with the Project Team Guide, selection or removal of core reviewers is not a democratic process.
  - Our PTL shall maintain a core reviewer group that works well together as a team. Our PTL will seek advice from our community when making such changes, but ultimately decides.
  - Clearly communicate additions to the developer mailing list.

Our Development Philosophy

Overview

- Continuous iterative improvements.
- Small contributions preferred.
- Perfect is the enemy of good.
- We need a compass, not a master plan.

Discussion

We believe in making continuous iterative improvements to our software. Making several small improvements is preferred over making fewer large changes. Contributions of about perhaps 400 lines of change or less are considered ideal because they are easier to review. This makes them more efficient from a review perspective than larger contributions are, because they get reviewed more quickly, and are faster to revise than larger works. We also encourage unrelated changes to be contributed in separate patches to make reasoning about each one simpler.

Although we should strive for perfection in our work, we must recognize that what matters more than absolute perfection is that our software is consistently improving over time. When contributions are

slowed down by too many revisions, we should decide to merge code even when it is imperfect, as long as we have systematically tracked the weaknesses so we can revisit them with subsequent revision efforts.

Rule of Thumb

Our rule of thumb shall be the answer to two simple questions:

1. Is this patch making Magnum better?
2. Will this patch cause instability, or prevent others from using Magnum effectively?

If the answers respectively are *yes* and *no*, and our objections can be effectively addressed in a follow-up patch, then we should decide to merge code with tech-debt bug tickets to systematically track our desired improvements.

How We Make Decisions

Team Consensus

On the Magnum team, we rely on Team Consensus to make key decisions. Team Consensus is the harmonious and peaceful agreement of the majority of our participating team. That means that we seek a clear indication of agreement of those engaged in discussion of a topic. Consensus shall not be confused with the concept of Unanimous Consent where all participants are in full agreement. Our decisions do not require Unanimous Consent. We may still have a team consensus even if we have a small number of team members who disagree with the majority viewpoint. We must recognize that we will not always agree on every key decision. Whats more important than our individual position on an argument is that the interests of our team are met.

We shall take reasonable efforts to address all opposition by fairly considering it before making a decision. Although Unanimous Consent is not required to make a key decision, we shall not overlook legitimate questions or concerns. Once each such concern has been addressed, we may advance to making a determination of Team Consensus.

Some code level changes are controversial in nature. If this happens, and a core reviewer judges the minority viewpoint to be reasonably considered, he or she may conclude we have Team Consensus and approve the patch for merge using the normal voting guidelines. We shall allow reasonable time for discussion and socialization when controversial decisions are considered.

If any contributor disagrees with a merged patch, and believes our decision should be reconsidered, (s)he may consult our *Reverting Patches* guidelines.

No Deadlocks

We shall not accept any philosophy of agree to disagree. This form of deadlock is not decision making, but the absence of it. Instead, we shall proceed to decision making in a timely fashion once all input has been fairly considered. We shall accept when a decision does not go our way.

Handling Disagreement

When we disagree, we shall first consult the OpenStack Community Code of Conduct for guidance. In accordance with our code of conduct, our disagreements shall be handled with patience, respect, and fair consideration for those who dont share the same point of view. When we do not agree, we take care to ask why. We strive to understand the reasons we disagree, and seek opportunities to reach a compromise.

Our PTL is responsible for determining Team Consensus when it can not be reached otherwise. In extreme cases, it may be possible to appeal a PTL decision to the OpenStack TC.

Open Design Process

One of the four open principles embraced by the OpenStack community is Open Design. We collaborate openly to design new features and capabilities, as well as planning major improvements to our software. We use multiple venues to conduct our design, including:

- Written specifications
- Blueprints
- Bug tickets
- PTG meetings
- Summit meetings
- IRC meetings
- Mailing list discussions
- Review comments
- IRC channel discussion

The above list is ordered by formality level. Notes and/or minutes from meetings shall be recorded in etherpad documents so they can be accessed by participants not present in the meetings. Meetings shall be open, and shall not intentionally exclude any stakeholders.

Specifications

The most formal venue for open design are written specifications. These are RST format documents that are proposed in the magnum-specs code repository by release cycle name. The repository holds a template for the format of the document, as required by our PTL for each release cycle.

Specifications are intended to be a high level description of a major feature or capability, expressed in a way to demonstrate that the feature has been well contemplated, and is acceptable by Team Consensus. Using specifications allows us to change direction without requiring code rework because input can be considered before code has been written.

Specifications do not require specific implementation details. They shall describe the implementation in enough detail to give reviewers a high level sense of what to expect, with examples to make new concepts clear. We do not require specifications that detail every aspect of the implementation. We recognize that it is more effective to express implementations with patches than conveying them in the abstract. If a proposed patch set for an implementation is not acceptable, we can address such concerns using review comments on those patches. If a reviewer has an alternate idea for implementation, they are welcome to develop another patch in WIP or completed form to demonstrate an alternative approach

for consideration. This option for submitting an alternative review is available for alternate specification ideas that reach beyond the scope of a simple review comment. Offering reviewers multiple choices for contributions is welcome, and is not considered wasteful.

Implementations of features do not require merged specifications. However, major features or refactoring should be expressed in a specification so reviewers will know what to expect prior to considering code for review. Contributors are welcome to start implementation before the specifications are merged, but should be ready to revise the implementation as needed to conform with changes in the merged specification.

Reviews

A review is a patch set that includes a proposal for inclusion in our code base. We follow the process outlined in the Code Review section of the OpenStack Developers Guide. The following workflow states may by applied to each review:

State	Meaning	Detail
workflow -1	Work in progress	This patch is submitted for team input, but should not yet be considered for merge. May be set by a core reviewer as a courtesy. It can be set after workflow+1 but prior to merge in order to prevent a gate breaking merge.
workflow 0	Ready for reviews	This patch should be considered for merge.
workflow+1	Approved	This patch has received at least two +2 votes, and is approved for merge. Also known as a +A vote.

The following votes may be applied to a review:

Vote	Meaning
-2	**Do Not Merge** • WARNING: Use extreme caution applying this vote, because contributors perceive this action as hostile unless it is accompanied with a genuine offer to help remedy a critical concern collaboratively. • This vote is a veto that indicates a critical problem with the contribution. It is sticky, meaning it must be removed by the individual who added it, even if further revisions are made. • All -2 votes shall be accompanied with a polite comment that clearly states what can be changed by the contributor to result in reversal or downgrade of the vote to a -1. • **Core reviewers may use this vote:** – To indicate a critical problem to address, such as a security vulnerability that other core reviewers may be unable to recognize. – To indicate a decision that the patch is not consistent with the direction of the project, subsequent to conference with the PTL about the matter. • **The PTL may use this vote:** – To indicate a decision that the patch is not consistent with the direction of the project. – While coordinating a release to prevent incompatible changes from merging before the release is tagged. – To address a critical concern with the contribution. • **Example uses of this vote that are not considered ap** – To ensure more reviews before merge. – To block competing patches. – In cases when you lack the time to follow up closely afterward. • To avoid a -2 vote on your contribution, discuss your plans with the development team prior to writing code, and post a WIP (*workflow-1*) patch while you are working on it, and ask for input before you submit it for

Merge Criteria

We want code to merge relatively quickly in order to keep a rapid pace of innovation. Rather than asking reviewers to wait a prescribed arbitrary time before merging patches, we instead use a simple *2 +2s* policy for approving new code for merge. The following criteria apply when judging readiness to merge a patch:

1. All contributions shall be peer reviewed and approved with a +2 vote by at least two core reviewers prior to being merged. Exceptions known as *Fast Merge* commits may bypass peer review as allowed by this policy.
2. The approving reviewer shall verify that all open questions and concerns have been adequately addressed prior to voting +A by adding the workflow+1 to merge a patch. This judgement verifies that *Team Consensus* has been reached.

Note: We discourage any *workflow+1* vote on patches that only have two +2 votes from cores from the same affiliation. This guideline applies when reviewer diversity allows for it.

See *Reverting Patches* for details about how to remedy mistakes when code is merged too quickly.

Reverting Patches

Moving quickly with our *Merge Criteria* means that sometimes we might make mistakes. If we do, we may revert problematic patches. The following options may be applied:

1. Any contributor may revert a change by submitting a patch to undo the objection and include a reference to the original patch in the commit message. The commit message shall include clear rationale for considering the revert. Normal voting rules apply.
2. Any contributor may re-implement a feature using an alternate approach at any time, even after a previous implementation has merged. Normal voting rules apply.
3. If a core reviewer wishes to revert a change (s)he may use the options described above, or may apply the *Fast Revert* policy.

Fast Merge

Sometimes we need to merge code quickly by bypassing the peer review process when justified. Allowed exceptions include:

- **PTL (Project Team Lead) Intervention / Core intervention**
  - Emergency un-break gate.
  - VMT embargoed patch submitted to Gerrit.
- Automatic proposals (e.g. requirements updates).
- PTL / Core discretion (with comment) that a patch already received a +2 but minor (typo/rebase) fixes were addressed by another core reviewer and the *correcting* reviewer has opted to carry forward the other +2. The *correcting* reviewer shall not be the original patch submitter.

We recognize that mistakes may happen when changes are merged quickly. When concerns with any *Fast Merge* surface, our *Fast Revert* policy may be applied.

Fast Revert

This policy was adapted from novas Reverts for Retrospective Vetos policy in 2017. Sometimes our simple *2 +2s* approval policy will result in errors when we move quickly. These errors might be a bug that was missed, or equally importantly, it might be that other cores feel that there is a need for further discussion on the implementation of a given piece of code.

Rather than an enforced time-based solution - for example, a patch could not be merged until it has been up for review for 3 days - we have chosen an honor-based system of *Team Consensus* where core reviewers do not approve controversial patches until proposals are sufficiently socialized and everyone has a chance to raise any concerns.

Recognizing that mistakes can happen, we also have a policy where contentious patches which were quickly approved may be reverted so that the discussion around the proposal may continue as if the patch had never been merged in the first place. In such a situation, the procedure is:

1. The commit to be reverted must not have been released.
2. The core team member who has a -2 worthy objection may propose a revert, stating the specific concerns that they feel need addressing.
3. Any subsequent patches depending on the to-be-reverted patch shall be reverted also, as needed.
4. Other core team members shall quickly approve the revert. No detailed debate is needed at this point. A -2 vote on a revert is strongly discouraged, because it effectively blocks the right of cores approving the revert from -2 voting on the original patch.
5. The original patch submitter may re-submit the change, with a reference to the original patch and the revert.
6. The original reviewers of the patch shall restore their votes and attempt to summarize their previous reasons for their votes.
7. The patch shall not be re-approved until the concerns of the opponents are fairly considered. A mailing list discussion or design spec may be the best way to achieve this.

This policy shall not be used in situations where *Team Consensus* was fairly reached over a reasonable period of time. A *Fast Revert* applies only to new concerns that were not part of the *Team Consensus* determination when the patch was merged.

See also: *Team Consensus*.

Continuous Improvement

If any part of this document is not clear, or if you have suggestions for how to improve it, please contact our PTL for help.

CHAPTER

SIX

ADMIN GUIDE

6.1 Administrators Guide

6.1.1 Installation & Operations

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

Using Proxies in magnum if running under firewall

If you are running magnum behind a firewall then you may need a proxy for using services like docker, kubernetes and mesos. Use these steps when your firewall will not allow you to use those services without a proxy.

NOTE: This feature has only been tested with the supported cluster type and associated image: Kubernetes and Swarm use the Fedora Atomic image, and Mesos uses the Ubuntu image.

Proxy Parameters to define before use

1. http-proxy

Address of a proxy that will receive all HTTP requests and relay them. The format is a URL including a port number. For example: http://10.11.12.13:8000 or http://abcproxy.com:8000

2. https-proxy

Address of a proxy that will receive all HTTPS requests and relay them. The format is a URL including a port number. For example: https://10.11.12.13:8000 or https://abcproxy.com:8000

3. no-proxy

A comma separated list of IP addresses or hostnames that should bypass your proxy, and make connections directly.

NOTE: You may not express networks/subnets. It only accepts names and ip addresses. Bad example: 192.168.0.0/28.

Steps to configure proxies.

You can specify all three proxy parameters while creating ClusterTemplate of any coe type. All of proxy parameters are optional.

```
$ openstack coe cluster template create k8s-cluster-template \
                    --image fedora-atomic-latest \
                    --keypair testkey \
                    --external-network public \
                    --dns-nameserver 8.8.8.8 \
                    --flavor m1.small \
                    --coe kubernetes \
                    --http-proxy <http://abc-proxy.com:8080> \
                    --https-proxy <https://abc-proxy.com:8080> \
                    --no-proxy <172.24.4.4,172.24.4.9,172.24.4.8>
```

```
$ openstack coe cluster template create swarm-cluster-template \
                    --image fedora-atomic-latest \
                    --keypair testkey \
                    --external-network public \
                    --dns-nameserver 8.8.8.8 \
                    --flavor m1.small \
                    --coe swarm \
                    --http-proxy <http://abc-proxy.com:8080> \
                    --https-proxy <https://abc-proxy.com:8080> \
                    --no-proxy <172.24.4.4,172.24.4.9,172.24.4.8>
```

```
$ openstack coe cluster template create mesos-cluster-template \
                    --image ubuntu-mesos \
                    --keypair testkey \
                    --external-network public \
                    --dns-nameserver 8.8.8.8 \
                    --flavor m1.small \
                    --coe mesos \
                    --http-proxy <http://abc-proxy.com:8080> \
                    --https-proxy <https://abc-proxy.com:8080> \
                    --no-proxy <172.24.4.4,172.24.4.9,172.24.4.8>
```

Guru Meditation Reports

Magnum contains a mechanism whereby developers and system administrators can generate a report about the state of a running Magnum executable. This report is called a *Guru Meditation Report* (*GMR* for short).

Generating a GMR

A *GMR* can be generated by sending the *USR2* signal to any Magnum process with support (see below). The *GMR* will then be outputted as standard error for that particular process.

For example, suppose that `magnum-api` has process id `8675`, and was run with `2>/var/log/magnum/magnum-api-err.log`. Then, `kill -USR2 8675` will trigger the Guru Meditation report to be printed to `/var/log/magnum/magnum-api-err.log`.

Structure of a GMR

The *GMR* is designed to be extensible; any particular executable may add its own sections. However, the base *GMR* consists of several sections:

Package Shows information about the package to which this process belongs, including version information.

Threads Shows stack traces and thread ids for each of the threads within this process.

Green Threads Shows stack traces for each of the green threads within this process (green threads dont have thread ids).

Configuration Lists all the configuration options currently accessible via the CONF object for the current process.

Adding Support for GMRs to New Executables

Adding support for a *GMR* to a given executable is fairly easy.

First import the module:

```
from oslo_reports import guru_meditation_report as gmr
from magnum import version
```

Then, register any additional sections (optional):

```
TextGuruMeditation.register_section('Some Special Section',
                                    some_section_generator)
```

Finally (under main), before running the main loop of the executable (usually `service.server(server)` or something similar), register the *GMR* hook:

```
TextGuruMeditation.setup_autorun(version)
```

Extending the GMR

As mentioned above, additional sections can be added to the GMR for a particular executable. For more information, see the inline documentation under `oslo.reports`

Magnum Troubleshooting Guide

This guide is intended for users who use Magnum to deploy and manage clusters of hosts for a Container Orchestration Engine. It describes common failure conditions and techniques for troubleshooting. To help the users quickly identify the relevant information, the guide is organized as a list of failure symptoms: each has some suggestions with pointers to the details for troubleshooting.

A separate section *for developers* describes useful techniques such as debugging unit tests and gate tests.

Failure symptoms

My cluster-create takes a really long time If you are using devstack on a small VM, cluster-create will take a long time and may eventually fail because of insufficient resources. Another possible reason is that a process on one of the nodes is hung and heat is still waiting on the signal. In this case, it will eventually fail with a timeout, but since heat has a long default timeout, you can look at the *heat stacks* and check the WaitConditionHandle resources.

My cluster-create fails with error: Failed to create trustee XXX in domain XXX Check the *trustee for cluster*

Kubernetes cluster-create fails Check the *heat stacks*, log into the master nodes and check the *Kubernetes services* and *etcd service*.

Swarm cluster-create fails Check the *heat stacks*, log into the master nodes and check the *Swarm services* and *etcd service*.

Mesos cluster-create fails Check the *heat stacks*, log into the master nodes and check the *Mesos services*.

I get the error Timed out waiting for a reply when deploying a pod Verify the *Kubernetes services* and *etcd service* are running on the master nodes.

I deploy pods on Kubernetes cluster but the status stays Pending The pod status is Pending while the Docker image is being downloaded, so if the status does not change for a long time, log into the minion node and check for *Cluster internet access*.

I deploy pods and services on Kubernetes cluster but the app is not working The pods and services are running and the status looks correct, but if the app is performing communication between pods through services, verify *Kubernetes networking*.

Swarm cluster is created successfully but I cannot deploy containers Check the *Swarm services* and *etcd service* on the master nodes.

Mesos cluster is created successfully but I cannot deploy containers on Marathon Check the *Mesos services* on the master node.

I get a Protocol violation error when deploying a container For Kubernetes, check the *Kubernetes services* to verify that kube-apiserver is running to accept the request. Check *TLS* and *Barbican service*.

My cluster-create fails with a resource error on docker_volume Check for available volume space on Cinder and the *request volume size* in the heat template. Run nova volume-list to check the volume status.

Troubleshooting details

Heat stacks

To be filled in

A cluster is deployed by a set of heat stacks: one top level stack and several nested stack. The stack names are prefixed with the cluster name and the nested stack names contain descriptive internal names like *kube_masters*, *kube_minions*.

To list the status of all the stacks for a cluster:

> heat stack-list -n | grep *cluster-name*

If the cluster has failed, then one or more of the heat stacks would have failed. From the stack list above, look for the stacks that failed, then look for the particular resource(s) that failed in the failed stack by:

> heat resource-list *failed-stack-name* | grep FAILED

The resource_type of the failed resource should point to the OpenStack service, e.g. OS::Cinder::Volume. Check for more details on the failure by:

> heat resource-show *failed-stack-name failed-resource-name*

The resource_status_reason may give an indication on the failure, although in some cases it may only say Unknown.

If the failed resource is OS::Heat::WaitConditionHandle, this indicates that one of the services that are being started on the node is hung. Log into the node where the failure occurred and check the respective *Kubernetes services*, *Swarm services* or *Mesos services*. If the failure is in other scripts, look for them as *Heat software resource scripts*.

Trustee for cluster

When a user creates a cluster, Magnum will dynamically create a service account for the cluster. The service account will be used by the cluster to access the OpenStack services (i.e. Neutron, Swift, etc.). A trust relationship will be created between the user who created the cluster (the trustor) and the service account created for the cluster (the trustee). For details, please refer to Create a trustee user for each bay.

If Magnum fails to create the trustee, check the magnum config file (usually in /etc/magnum/magnum.conf). Make sure trustee_* and www_authenticate_uri are set and their values are correct:

> [keystone_authtoken] www_authenticate_uri = http://controller:5000/v3
>
> [trust] trustee_domain_admin_password = XXX trustee_domain_admin_id = XXX trustee_domain_id = XXX

If the trust group is missing, you might need to create the trustee domain and the domain admin:

```
. /opt/stack/devstack/accrc/admin/admin
export OS_IDENTITY_API_VERSION=3
unset OS_AUTH_TYPE
openstack domain create magnum
openstack user create trustee_domain_admin --password secret \
    --domain magnum
openstack role add --user=trustee_domain_admin --user-domain magnum \
    --domain magnum admin

. /opt/stack/devstack/functions
export MAGNUM_CONF=/etc/magnum/magnum.conf
iniset $MAGNUM_CONF trust trustee_domain_id \
    $(openstack domain show magnum | awk '/ id /{print $4}')
iniset $MAGNUM_CONF trust trustee_domain_admin_id \
    $(openstack user show trustee_domain_admin | awk '/ id /{print $4}')
iniset $MAGNUM_CONF trust trustee_domain_admin_password secret
```

Then, restart magnum-api and magnum-cond to pick up the new configuration. If the problem still exists, you might want to manually verify your domain admin credential to ensure it has the right privilege. To do that, run the script below with the credentials replaced (you must use the IDs where specified). If it fails, that means the credential you provided is invalid.

```
from keystoneauth1.identity import v3 as ka_v3
from keystoneauth1 import session as ka_session
from keystoneclient.v3 import client as kc_v3

auth = ka_v3.Password(
    auth_url=YOUR_AUTH_URI,
    user_id=YOUR_TRUSTEE_DOMAIN_ADMIN_ID,
    domain_id=YOUR_TRUSTEE_DOMAIN_ID,
    password=YOUR_TRUSTEE_DOMAIN_ADMIN_PASSWORD)

session = ka_session.Session(auth=auth)
domain_admin_client = kc_v3.Client(session=session)
user = domain_admin_client.users.create(
    name='anyname',
    password='anypass')
```

TLS

In production deployments, operators run the OpenStack APIs using ssl certificates and in private clouds it is common to use self-signed or certificates signed from CAs that they are usually not included in the systems default CA-bundles. Magnum clusters with TLS enabled have their own CA but they need to make requests to the OpenStack APIs for several reasons. Eg Get the cluster CA and sign node certificates (Keystone, Magnum), signal the Heat API for stack completion, create resources (volumes, load balancers) or get information for each node (Cinder, Neutron, Nova). In these cases, the cluster nodes need the CA used for to run the APIs.

To pass the OpenStack CA bundle to the nodes you can set the CA using the *openstack_ca_file* option in the *drivers* section of Magnums configuration file (usually */etc/magnum/magnum.conf*). The default

drivers in magnum install this CA in the system and set it in all the places it might be needed (eg when configuring the kubernetes cloud provider or for the heat-agents.)

The cluster nodes will validate the Certificate Authority by default when making requests to the OpenStack APIs (Keystone, Magnum, Heat). If you need to disable CA validation, the configuration parameter verify_ca can be set to False. More information on CA Validation.

Barbican service

To be filled in

Cluster internet access

The nodes for Kubernetes, Swarm and Mesos are connected to a private Neutron network, so to provide access to the external internet, a router connects the private network to a public network. With devstack, the default public network is public, but this can be replaced by the parameter external-network in the ClusterTemplate. The public network with devstack is actually not a real external network, so it is in turn routed to the network interface of the host for devstack. This is configured in the file local.conf with the variable PUBLIC_INTERFACE, for example:

```
PUBLIC_INTERFACE=eth1
```

If the route to the external internet is not set up properly, the ectd discovery would fail (if using public discovery) and container images cannot be downloaded, among other failures.

First, check for connectivity to the external internet by pinging an external IP (the IP shown here is an example; use an IP that works in your case):

```
ping 8.8.8.8
```

If the ping fails, there is no route to the external internet. Check the following:

- Is PUBLIC_INTERFACE in devstack/local.conf the correct network interface? Does this interface have a route to the external internet?
- If external-network is specified in the ClusterTemplate, does this network have a route to the external internet?
- Is your devstack environment behind a firewall? This can be the case for some enterprises or countries. In this case, consider using a *proxy server*.
- Is the traffic blocked by the security group? Check the rules of security group.
- Is your host NATing your internal network correctly? Check your host iptables.
- Use *tcpdump* for networking troubleshooting. You can run *tcpdump* on the interface *docker0, flannel0* and *eth0* on the node and then run *ping* to see the path of the message from the container.

If ping is successful, check that DNS is working:

```
wget google.com
```

If DNS works, you should get back a few lines of HTML text.

If the name lookup fails, check the following:

- Is the DNS entry correct in the subnet? Try neutron subnet-show <subnet-id> for the private subnet and check dns_nameservers. The IP should be either the default public DNS 8.8.8.8 or the value specified by dns-nameserver in the ClusterTemplate.
- If you are using your own DNS server by specifying dns-nameserver in the ClusterTemplate, is it reachable and working?
- More help on DNS troubleshooting.

Kubernetes networking

The networking between pods is different and separate from the neutron network set up for the cluster. Kubernetes presents a flat network space for the pods and services and uses different network drivers to provide this network model.

It is possible for the pods to come up correctly and be able to connect to the external internet, but they cannot reach each other. In this case, the app in the pods may not be working as expected. For example, if you are trying the redis example, the key:value may not be replicated correctly. In this case, use the following steps to verify the inter-pods networking and pinpoint problems.

Since the steps are specific to the network drivers, refer to the particular driver being used for the cluster.

Using Flannel as network driver

Flannel is the default network driver for Kubernetes clusters. Flannel is an overlay network that runs on top of the neutron network. It works by encapsulating the messages between pods and forwarding them to the correct node that hosts the target pod.

First check the connectivity at the node level. Log into two different minion nodes, e.g. node A and node B, run a docker container on each node, attach to the container and find the IP.

For example, on node A:

```
sudo docker run -it alpine
# ip -f inet -o a | grep eth0 | awk '{print $4}'
10.100.54.2/24
```

Similarly, on node B:

```
sudo docker run -it alpine
# ip -f inet -o a | grep eth0 | awk '{print $4}'
10.100.49.3/24
```

Check that the containers can see each other by pinging from one to another.

On node A:

```
# ping 10.100.49.3
PING 10.100.49.3 (10.100.49.3): 56 data bytes
64 bytes from 10.100.49.3: seq=0 ttl=60 time=1.868 ms
64 bytes from 10.100.49.3: seq=1 ttl=60 time=1.108 ms
```

Similarly, on node B:

```
# ping 10.100.54.2
PING 10.100.54.2 (10.100.54.2): 56 data bytes
64 bytes from 10.100.54.2: seq=0 ttl=60 time=2.678 ms
64 bytes from 10.100.54.2: seq=1 ttl=60 time=1.240 ms
```

If the ping is not successful, check the following:

- Is neutron working properly? Try pinging between the VMs.
- Are the docker0 and flannel0 interfaces configured correctly on the nodes? Log into each node and find the Flannel CIDR by:

  ```
  cat /run/flannel/subnet.env | grep FLANNEL_SUBNET
  FLANNEL_SUBNET=10.100.54.1/24
  ```

  Then check the interfaces by:

  ```
  ifconfig flannel0
  ifconfig docker0
  ```

  The correct configuration should assign flannel0 with the 0 address in the subnet, like *10.100.54.0*, and docker0 with the 1 address, like *10.100.54.1*.
- Verify the IPs assigned to the nodes as found above are in the correct Flannel subnet. If this is not correct, the docker daemon is not configured correctly with the parameter *bip*. Check the systemd service for docker.
- Is Flannel running properly? check the *Running Flannel*.
- Ping and try tcpdump on each network interface along the path between two nodes to see how far the message is able to travel. The message path should be as follows:
  1. Source node: docker0
  2. Source node: flannel0
  3. Source node: eth0
  4. Target node: eth0
  5. Target node: flannel0
  6. Target node: docker0

If ping works, this means the flannel overlay network is functioning correctly.

The containers created by Kubernetes for pods will be on the same IP subnet as the containers created directly in Docker as above, so they will have the same connectivity. However, the pods still may not be able to reach each other because normally they connect through some Kubernetes services rather than directly. The services are supported by the kube-proxy and rules inserted into the iptables, therefore their networking paths have some extra hops and there may be problems here.

To check the connectivity at the Kubernetes pod level, log into the master node and create two pods and a service for one of the pods. You can use the examples provided in the directory */etc/kubernetes/examples/* for the first pod and service. This will start up an nginx container and a Kubernetes service to expose the endpoint. Create another manifest for a second pod to test the endpoint:

```
cat > alpine.yaml << END
apiVersion: v1
kind: Pod
metadata:
  name: alpine
spec:
  containers:
  - name: alpine
    image: alpine
    args:
    - sleep
    - "1000000"
END

kubectl create -f /etc/kubernetes/examples/pod-nginx-with-label.yaml
kubectl create -f /etc/kubernetes/examples/service.yaml
kubectl create -f alpine.yaml
```

Get the endpoint for the nginx-service, which should route message to the pod nginx:

```
kubectl describe service nginx-service | grep -e IP: -e Port:
IP:                     10.254.21.158
Port:                   <unnamed>       8000/TCP
```

Note the IP and port to use for checking below. Log into the node where the *alpine* pod is running. You can find the hosting node by running this command on the master node:

```
kubectl get pods -o wide  | grep alpine | awk '{print $6}'
k8-gzvjwcooto-0-gsrxhmyjupbi-kube-minion-br73i6ans2b4
```

To get the IP of the node, query Nova on devstack:

```
nova list
```

On this hosting node, attach to the *alpine* container:

```
export DOCKER_ID=`sudo docker ps | grep k8s_alpine | awk '{print $1}'`
sudo docker exec -it $DOCKER_ID sh
```

From the *alpine* pod, you can try to reach the nginx pod through the nginx service using the IP and Port found above:

```
wget 10.254.21.158:8000
```

If the connection is successful, you should receive the file *index.html* from nginx.

If the connection is not successful, you will get an error message like::xs

> wget: cant connect to remote host (10.100.54.9): No route to host

In this case, check the following:

- Is kube-proxy running on the nodes? It runs as a container on each node. check by logging in the minion nodes and run:

```
sudo docker ps | grep k8s_kube-proxy
```

- Check the log from kube-proxy by running on the minion nodes:

```
export PROXY=`sudo docker ps | grep "hyperkube proxy" | awk '{print $1}'`
sudo docker logs $PROXY
```

- Try additional service debugging. To see whats going during provisioning:

```
kubectl get events
```

To get information on a service in question:

```
kubectl describe services <service_name>
```

etcd service

The etcd service is used by many other components for key/value pair management, therefore if it fails to start, these other components will not be running correctly either. Check that etcd is running on the master nodes by:

```
sudo service etcd status -l
```

If it is running correctly, you should see that the service is successfully deployed:

```
Active: active (running) since ....
```

The log message should show the service being published:

```
etcdserver: published {Name:10.0.0.5 ClientURLs:[http://10.0.0.5:2379]} to␣
↪cluster 3451e4c04ec92893
```

In some cases, the service may show as *active* but may still be stuck in discovery mode and not fully operational. The log message may show something like:

```
discovery: waiting for other nodes: error connecting to https://discovery.
↪etcd.io, retrying in 8m32s
```

If this condition persists, check for *Cluster internet access*.

If the daemon is not running, the status will show the service as failed, something like:

```
Active: failed (Result: timeout)
```

In this case, try restarting etcd by:

```
sudo service etcd start
```

If etcd continues to fail, check the following:

- Check the log for etcd:

```
sudo journalctl -u etcd
```

- etcd requires discovery, and the default discovery method is the public discovery service provided by etcd.io; therefore, a common cause of failure is that this public discovery service is not reachable. Check by running on the master nodes:

```
. /etc/sysconfig/heat-params
curl $ETCD_DISCOVERY_URL
```

You should receive something like:

```
{"action":"get",
 "node":{"key":"/_etcd/registry/00a6b00064174c92411b0f09ad5466c6",
         "dir":true,
         "nodes":[
           {"key":"/_etcd/registry/00a6b00064174c92411b0f09ad5466c6/
↪7d8a68781a20c0a5",
            "value":"10.0.0.5=http://10.0.0.5:2380",
            "modifiedIndex":978239406,
            "createdIndex":978239406}],
         "modifiedIndex":978237118,
         "createdIndex":978237118}
}
```

The list of master IP is provided by Magnum during cluster deployment, therefore it should match the current IP of the master nodes. If the public discovery service is not reachable, check the *Cluster internet access*.

Running Flannel

When deploying a COE, Flannel is available as a network driver for certain COE type. Magnum currently supports Flannel for a Kubernetes or Swarm cluster.

Flannel provides a flat network space for the containers in the cluster: they are allocated IP in this network space and they will have connectivity to each other. Therefore, if Flannel fails, some containers will not be able to access services from other containers in the cluster. This can be confirmed by running *ping* or *curl* from one container to another.

The Flannel daemon is run as a systemd service on each node of the cluster. To check Flannel, run on each node:

```
sudo service flanneld status
```

If the daemon is running, you should see that the service is successfully deployed:

```
Active: active (running) since ....
```

If the daemon is not running, the status will show the service as failed, something like:

```
Active: failed (Result: timeout) ....
```

or:

```
Active: inactive (dead) ....
```

Flannel daemon may also be running but not functioning correctly. Check the following:

- Check the log for Flannel:

  ```
  sudo journalctl -u flanneld
  ```

- Since Flannel relies on etcd, a common cause for failure is that the etcd service is not running on the master nodes. Check the *etcd service*. If the etcd service failed, once it has been restored successfully, the Flannel service can be restarted by:

  ```
  sudo service flanneld restart
  ```

- Magnum writes the configuration for Flannel in a local file on each master node. Check for this file on the master nodes by:

  ```
  cat /etc/sysconfig/flannel-network.json
  ```

  The content should be something like:

  ```
  {
    "Network": "10.100.0.0/16",
    "Subnetlen": 24,
    "Backend": {
      "Type": "udp"
    }
  }
  ```

  where the values for the parameters must match the corresponding parameters from the ClusterTemplate.

  Magnum also loads this configuration into etcd, therefore, verify the configuration in etcd by running *etcdctl* on the master nodes:

  ```
  . /etc/sysconfig/flanneld
  etcdctl get $FLANNEL_ETCD_KEY/config
  ```

- Each node is allocated a segment of the network space. Check for this segment on each node by:

  ```
  grep FLANNEL_SUBNET /run/flannel/subnet.env
  ```

  The containers on this node should be assigned an IP in this range. The nodes negotiate for their segment through etcd, and you can use *etcdctl* on the master node to query the network segment associated with each node:

  ```
  . /etc/sysconfig/flanneld
  for s in `etcdctl ls $FLANNEL_ETCD_KEY/subnets`
  do
  echo $s
  etcdctl get $s
  done
  ```

(continues on next page)

(continued from previous page)

```
/atomic.io/network/subnets/10.100.14.0-24
{"PublicIP":"10.0.0.5"}
/atomic.io/network/subnets/10.100.61.0-24
{"PublicIP":"10.0.0.6"}
/atomic.io/network/subnets/10.100.92.0-24
{"PublicIP":"10.0.0.7"}
```

Alternatively, you can read the full record in ectd by:

```
curl http://<master_node_ip>:2379/v2/keys/coreos.com/network/subnets
```

You should receive a JSON snippet that describes all the segments allocated.

- This network segment is passed to Docker via the parameter *bip*. If this is not configured correctly, Docker would not assign the correct IP in the Flannel network segment to the container. Check by:

```
cat /run/flannel/docker
ps -aux | grep docker
```

- Check the interface for Flannel:

```
ifconfig flannel0
```

The IP should be the first address in the Flannel subnet for this node.

- Flannel has several different backend implementations and they have specific requirements. The *udp* backend is the most general and have no requirement on the network. The *vxlan* backend requires vxlan support in the kernel, so ensure that the image used does provide vxlan support. The *host-gw* backend requires that all the hosts are on the same L2 network. This is currently met by the private Neutron subnet created by Magnum; however, if other network topology is used instead, ensure that this requirement is met if *host-gw* is used.

Current known limitation: the image fedora-21-atomic-5.qcow2 has Flannel version 0.5.0. This version has known bugs that prevent the backend vxland and host-gw to work correctly. Only the backend udp works for this image. Version 0.5.3 and later should work correctly. The image fedora-21-atomic-7.qcow2 has Flannel version 0.5.5.

Kubernetes services

To be filled in

(How to introspect k8s when heat works and k8s does not)

Additional Kubernetes troubleshooting section is available in the Monitoring, Logging, and Debugging section.

Swarm services

To be filled in

(How to check on a swarm cluster: see membership information, view master, agent containers)

Mesos services

To be filled in

Barbican issues

To be filled in

Docker CLI

To be filled in

Request volume size

To be filled in

Heat software resource scripts

To be filled in

For Developers

This section is intended to help with issues that developers may run into in the course of their development adventures in Magnum.

Troubleshooting in Gate

Simulating gate tests

1. Boot a VM
2. Provision this VM like so:

```
apt-get update \
&& apt-get upgrade \ # Kernel upgrade, as recommended by README, select␣
↪to keep existing grub config
&& apt-get install git tmux vim \
&& git clone https://git.openstack.org/openstack-infra/system-config \
&& system-config/install_puppet.sh && system-config/install_modules.sh \
```

(continues on next page)

(continued from previous page)

```
&& puppet apply \
--modulepath=/root/system-config/modules:/etc/puppet/modules \
-e "class { openstack_project::single_use_slave: install_users => false,
ssh_key => \"$( cat .ssh/authorized_keys | awk '{print $2}' )\" }" \
&& echo "jenkins ALL=(ALL) NOPASSWD:ALL" >> /etc/sudoers \
&& cat ~/.ssh/authorized_keys >> /home/jenkins/.ssh/authorized_keys
```

3. Compare `~/.ssh/authorized_keys` and `/home/jenkins/.ssh/authorized_keys`. Your original public SSH key should now be in `/home/jenkins/.ssh/authorized_keys`. If its not, explicitly copy it (this can happen if you spin up a using `--key-name <name>`, for example).
4. Assuming all is well up to this point, now its time to `reboot` into the latest kernel
5. Once youre done booting into the new kernel, log back in as `jenkins` user to continue with setting up the simulation.
6. Now its time to set up the workspace:

```
export REPO_URL=https://git.openstack.org
export WORKSPACE=/home/jenkins/workspace/testing
export ZUUL_URL=/home/jenkins/workspace-cache2
export ZUUL_REF=HEAD
export ZUUL_BRANCH=master
export ZUUL_PROJECT=openstack/magnum
mkdir -p $WORKSPACE
git clone $REPO_URL/$ZUUL_PROJECT $ZUUL_URL/$ZUUL_PROJECT \
&& cd $ZUUL_URL/$ZUUL_PROJECT \
&& git checkout remotes/origin/$ZUUL_BRANCH
```

7. At this point, you may be wanting to test a specific change. If so, you can pull down the changes in `$ZUUL_URL/$ZUUL_PROJECT` directory:

```
cd $ZUUL_URL/$ZUUL_PROJECT \
&& git fetch https://review.openstack.org/openstack/magnum refs/changes/
↪83/247083/12 && git checkout FETCH_HEAD
```

8. Now youre ready to pull down the `devstack-gate` scripts that will let you run the gate job on your own VM:

```
cd $WORKSPACE \
&& git clone --depth 1 $REPO_URL/openstack-infra/devstack-gate
```

9. And now you can kick off the job using the following script (the `devstack-gate` documentation suggests just copying from the job which can be found in the project-config repository), naturally it should be executable (`chmod u+x <filename>`):

```
#!/bin/bash -xe
cat > clonemap.yaml << EOF
clonemap:
  - name: openstack-infra/devstack-gate
    dest: devstack-gate
EOF
```

(continues on next page)

(continued from previous page)

```
/usr/zuul-env/bin/zuul-cloner -m clonemap.yaml --cache-dir /opt/git \
    https://git.openstack.org \
    openstack-infra/devstack-gate
export PYTHONUNBUFFERED=true
export DEVSTACK_GATE_TIMEOUT=240 # bump this if you see timeout issues. ␣
↪Default is 120
export DEVSTACK_GATE_TEMPEST=0
export DEVSTACK_GATE_NEUTRON=1
# Enable tempest for tempest plugin
export ENABLED_SERVICES=tempest
export BRANCH_OVERRIDE="default"
if [ "$BRANCH_OVERRIDE" != "default" ] ; then
    export OVERRIDE_ZUUL_BRANCH=$BRANCH_OVERRIDE
fi
export PROJECTS="openstack/magnum $PROJECTS"
export PROJECTS="openstack/python-magnumclient $PROJECTS"
export PROJECTS="openstack/barbican $PROJECTS"
export DEVSTACK_LOCAL_CONFIG="enable_plugin magnum https://git.openstack.
↪org/openstack/magnum"
export DEVSTACK_LOCAL_CONFIG+=$'\n'"enable_plugin ceilometer https://git.
↪openstack.org/openstack/ceilometer"
# Keep localrc to be able to set some vars in post_test_hook
export KEEP_LOCALRC=1
function gate_hook {
     cd /opt/stack/new/magnum/
    ./magnum/tests/contrib/gate_hook.sh api # change this to swarm to run␣
↪swarm functional tests or k8s to run kubernetes functional tests
}
export -f gate_hook
function post_test_hook {
    . $BASE/new/devstack/accrc/admin/admin
    cd /opt/stack/new/magnum/
    ./magnum/tests/contrib/post_test_hook.sh api # change this to swarm␣
↪to run swarm functional tests or k8s to run kubernetes functional tests
}
export -f post_test_hook
cp devstack-gate/devstack-vm-gate-wrap.sh ./safe-devstack-vm-gate-wrap.sh
./safe-devstack-vm-gate-wrap.sh
```

6.1.2 Configuration

Following pages will be helpful in configuring specific aspects of Magnum that may or may not be suitable to every situation.

Configuration

Magnum has a number of configuration options which will be detailed here.

Magnum Config

The magnum configuration file is called `magnum.conf`.

Magnum Pipeline

The pipeline details are contained in `api-paste.ini`.

Healthcheck Middleware

This piece of middleware creates an endpoint that allows a load balancer to probe if the API endpoint should be available at the node or not.

The healthcheck middleware should be deployed as a paste application application. Which is located in your `api-paste.ini` under a section called `[app:healthcheck]`. It should look like this:

```
[app:healthcheck]
paste.app_factory = oslo_middleware:Healthcheck.app_factory
backends = disable_by_file
disable_by_file_path = /etc/magnum/healthcheck_disable
```

The main pipeline using this application should look something like this also defined in the `api-paste.ini`:

```
[composite:main]
paste.composite_factory = magnum.api:root_app_factory
/: api
/healthcheck: healthcheck
```

If you wish to disable a middleware without taking it out of the pipeline, you can create a file under the file path defined by `disable_by_file_path` ie. `/etc/magnum/healthcheck_disable`.

For more information see oslo.middleware.

CHAPTER

SEVEN

CLI GUIDE

7.1 Magnum CLI Documentation

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

7.1.1 magnum-status

CLI interface for Magnum status commands

Synopsis

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

Description

magnum-status is a tool that provides routines for checking the status of a Magnum deployment.

Options

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

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

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

```
magnum-status
```

Categories are:

- `upgrade`

Detailed descriptions are below:

You can also run with a category argument such as `upgrade` to see a list of all commands in that category:

```
magnum-status upgrade
```

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

Upgrade

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

Table 1: **Sample Output**

Upgrade Check Results
Check: Sample Check Result: Success Details: Sample detail

Return Codes

Return code	Description
0	All upgrade readiness checks passed successfully and there is nothing to do.
1	At least one check encountered an issue and requires further investigation. This is considered a warning but the upgrade may be OK.
2	There was an upgrade status check failure that needs to be investigated. This should be considered something that stops an upgrade.
255	An unexpected error occurred.

History of Checks

8.0.0 (Stein)

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

CHAPTER

EIGHT

SAMPLE CONFIGURATIONS AND POLICIES

8.1 Sample Configuration and Policy File

8.1.1 Magnum Configuration Options

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

See the online version of this documentation for the full example config file.

8.1.2 Policy configuration

Configuration

> **Warning:** JSON formatted policy file is deprecated since Magnum 12.0.0 (Wallaby). This oslopolicy-convert-json-to-yaml tool will migrate your existing JSON-formatted policy file to YAML in a backward-compatible way.

The following is an overview of all available policies in Magnum. For a sample configuration file, refer to *policy.yaml*.

magnum

context_is_admin

Default `role:admin`

(no description provided)

admin_or_owner

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

(no description provided)

admin_api

Default `rule:context_is_admin`

(no description provided)

admin_or_user

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

(no description provided)

cluster_user

Default user_id:%(trustee_user_id)s

(no description provided)

deny_cluster_user

Default not domain_id:%(trustee_domain_id)s

(no description provided)

bay:create

Default rule:deny_cluster_user

Operations

- **POST** /v1/bays

Create a new bay.

bay:delete

Default rule:deny_cluster_user

Operations

- **DELETE** /v1/bays/{bay_ident}

Delete a bay.

bay:detail

Default rule:deny_cluster_user

Operations

- **GET** /v1/bays

Retrieve a list of bays with detail.

bay:get

Default rule:deny_cluster_user

Operations

- **GET** /v1/bays/{bay_ident}

Retrieve information about the given bay.

bay:get_all

Default rule:deny_cluster_user

Operations

- **GET** /v1/bays/

Retrieve a list of bays.

`bay:update`

Default `rule:deny_cluster_user`

Operations

- **PATCH** `/v1/bays/{bay_ident}`

Update an existing bay.

`baymodel:create`

Default `rule:deny_cluster_user`

Operations

- **POST** `/v1/baymodels`

Create a new baymodel.

`baymodel:delete`

Default `rule:deny_cluster_user`

Operations

- **DELETE** `/v1/baymodels/{baymodel_ident}`

Delete a baymodel.

`baymodel:detail`

Default `rule:deny_cluster_user`

Operations

- **GET** `/v1/baymodels`

Retrieve a list of baymodel with detail.

`baymodel:get`

Default `rule:deny_cluster_user`

Operations

- **GET** `/v1/baymodels/{baymodel_ident}`

Retrieve information about the given baymodel.

`baymodel:get_all`

Default `rule:deny_cluster_user`

Operations

- **GET** `/v1/baymodels`

Retrieve a list of baymodel.

`baymodel:update`

Default `rule:deny_cluster_user`

Operations

- **PATCH** /v1/baymodels/{baymodel_ident}

Update an existing baymodel.

baymodel:publish

Default rule:admin_api

Operations

- **POST** /v1/baymodels
- **PATCH** /v1/baymodels

Publish an existing baymodel.

certificate:create

Default rule:admin_or_user or rule:cluster_user

Operations

- **POST** /v1/certificates

Sign a new certificate by the CA.

certificate:get

Default rule:admin_or_user or rule:cluster_user

Operations

- **GET** /v1/certificates/{bay_uuid/cluster_uuid}

Retrieve CA information about the given bay/cluster.

certificate:rotate_ca

Default rule:admin_or_owner

Operations

- **PATCH** /v1/certificates/{bay_uuid/cluster_uuid}

Rotate the CA certificate on the given bay/cluster.

cluster:create

Default rule:deny_cluster_user

Operations

- **POST** /v1/clusters

Create a new cluster.

cluster:delete

Default rule:deny_cluster_user

Operations

- **DELETE** /v1/clusters/{cluster_ident}

Delete a cluster.

cluster:delete_all_projects

Default `rule:admin_api`

Operations

- **DELETE** `/v1/clusters/{cluster_ident}`

Delete a cluster from any project.

`cluster:detail`

Default `rule:deny_cluster_user`

Operations

- **GET** `/v1/clusters`

Retrieve a list of clusters with detail.

`cluster:detail_all_projects`

Default `rule:admin_api`

Operations

- **GET** `/v1/clusters`

Retrieve a list of clusters with detail across projects.

`cluster:get`

Default `rule:deny_cluster_user`

Operations

- **GET** `/v1/clusters/{cluster_ident}`

Retrieve information about the given cluster.

`cluster:get_one_all_projects`

Default `rule:admin_api`

Operations

- **GET** `/v1/clusters/{cluster_ident}`

Retrieve information about the given cluster across projects.

`cluster:get_all`

Default `rule:deny_cluster_user`

Operations

- **GET** `/v1/clusters/`

Retrieve a list of clusters.

`cluster:get_all_all_projects`

Default `rule:admin_api`

Operations

- **GET** `/v1/clusters/`

Retrieve a list of all clusters across projects.

`cluster:update`

Default `rule:deny_cluster_user`

Operations

- **PATCH** `/v1/clusters/{cluster_ident}`

Update an existing cluster.

`cluster:update_health_status`

Default `rule:admin_or_user or rule:cluster_user`

Operations

- **PATCH** `/v1/clusters/{cluster_ident}`

Update the health status of an existing cluster.

`cluster:update_all_projects`

Default `rule:admin_api`

Operations

- **PATCH** `/v1/clusters/{cluster_ident}`

Update an existing cluster.

`cluster:resize`

Default `rule:deny_cluster_user`

Operations

- **POST** `/v1/clusters/{cluster_ident}/actions/resize`

Resize an existing cluster.

`cluster:upgrade`

Default `rule:deny_cluster_user`

Operations

- **POST** `/v1/clusters/{cluster_ident}/actions/upgrade`

Upgrade an existing cluster.

`cluster:upgrade_all_projects`

Default `rule:admin_api`

Operations

- **POST** `/v1/clusters/{cluster_ident}/actions/upgrade`

Upgrade an existing cluster across all projects.

`clustertemplate:create`

Default `rule:deny_cluster_user`

Operations

- **POST** `/v1/clustertemplates`

Create a new cluster template.

`clustertemplate:delete`

Default `rule:admin_or_owner`

Operations

- **DELETE** `/v1/clustertemplate/{clustertemplate_ident}`

Delete a cluster template.

`clustertemplate:delete_all_projects`

Default `rule:admin_api`

Operations

- **DELETE** `/v1/clustertemplate/{clustertemplate_ident}`

Delete a cluster template from any project.

`clustertemplate:detail_all_projects`

Default `rule:admin_api`

Operations

- **GET** `/v1/clustertemplates`

Retrieve a list of cluster templates with detail across projects.

`clustertemplate:detail`

Default `rule:deny_cluster_user`

Operations

- **GET** `/v1/clustertemplates`

Retrieve a list of cluster templates with detail.

`clustertemplate:get`

Default `rule:deny_cluster_user`

Operations

- **GET** `/v1/clustertemplate/{clustertemplate_ident}`

Retrieve information about the given cluster template.

`clustertemplate:get_one_all_projects`

Default `rule:admin_api`

Operations

- **GET** `/v1/clustertemplate/{clustertemplate_ident}`

Retrieve information about the given cluster template across project.

`clustertemplate:get_all`

Default `rule:deny_cluster_user`

Operations

- **GET** /v1/clustertemplates

Retrieve a list of cluster templates.

`clustertemplate:get_all_all_projects`

Default rule:admin_api

Operations

- **GET** /v1/clustertemplates

Retrieve a list of cluster templates across projects.

`clustertemplate:update`

Default rule:admin_or_owner

Operations

- **PATCH** /v1/clustertemplate/{clustertemplate_ident}

Update an existing cluster template.

`clustertemplate:update_all_projects`

Default rule:admin_api

Operations

- **PATCH** /v1/clustertemplate/{clustertemplate_ident}

Update an existing cluster template.

`clustertemplate:publish`

Default rule:admin_api

Operations

- **POST** /v1/clustertemplates
- **PATCH** /v1/clustertemplates

Publish an existing cluster template.

`federation:create`

Default rule:deny_cluster_user

Operations

- **POST** /v1/federations

Create a new federation.

`federation:delete`

Default rule:deny_cluster_user

Operations

- **DELETE** /v1/federations/{federation_ident}

Delete a federation.

`federation:detail`

Default `rule:deny_cluster_user`

Operations

- **GET** `/v1/federations`

Retrieve a list of federations with detail.

`federation:get`

Default `rule:deny_cluster_user`

Operations

- **GET** `/v1/federations/{federation_ident}`

Retrieve information about the given federation.

`federation:get_all`

Default `rule:deny_cluster_user`

Operations

- **GET** `/v1/federations/`

Retrieve a list of federations.

`federation:update`

Default `rule:deny_cluster_user`

Operations

- **PATCH** `/v1/federations/{federation_ident}`

Update an existing federation.

`magnum-service:get_all`

Default `rule:admin_api`

Operations

- **GET** `/v1/mservices`

Retrieve a list of magnum-services.

`quota:create`

Default `rule:admin_api`

Operations

- **POST** `/v1/quotas`

Create quota.

`quota:delete`

Default `rule:admin_api`

Operations

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

Delete quota for a given project_id and resource.

`quota:get`

Default `rule:admin_or_owner`

Operations

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

Retrieve Quota information for the given project_id.

`quota:get_all`

Default `rule:admin_api`

Operations

- **GET** `/v1/quotas`

Retrieve a list of quotas.

`quota:update`

Default `rule:admin_api`

Operations

- **PATCH** `/v1/quotas/{project_id}/{resource}`

Update quota for a given project_id.

`stats:get_all`

Default `rule:admin_or_owner`

Operations

- **GET** `/v1/stats`

Retrieve magnum stats.

`nodegroup:get`

Default `rule:admin_or_owner`

Operations

- **GET** `/v1/clusters/{cluster_id}/nodegroup/{nodegroup}`

Retrieve information about the given nodegroup.

`nodegroup:get_all`

Default `rule:admin_or_owner`

Operations

- **GET** `/v1/clusters/{cluster_id}/nodegroups/`

Retrieve a list of nodegroups that belong to a cluster.

`nodegroup:get_all_all_projects`

Default `rule:admin_api`

Operations

- **GET** `/v1/clusters/{cluster_id}/nodegroups/`

Retrieve a list of nodegroups across projects.

`nodegroup:get_one_all_projects`

Default `rule:admin_api`

Operations

- **GET** `/v1/clusters/{cluster_id}/nodegroups/{nodegroup}`

Retrieve infornation for a given nodegroup.

`nodegroup:create`

Default `rule:admin_or_owner`

Operations

- **POST** `/v1/clusters/{cluster_id}/nodegroups/`

Create a new nodegroup.

`nodegroup:delete`

Default `rule:admin_or_owner`

Operations

- **DELETE** `/v1/clusters/{cluster_id}/nodegroups/{nodegroup}`

Delete a nodegroup.

`nodegroup:update`

Default `rule:admin_or_owner`

Operations

- **PATCH** `/v1/clusters/{cluster_id}/nodegroups/{nodegroup}`

Update an existing nodegroup.

8.1.3 Sample configuration files

Configuration files can alter how Magnum behaves at runtime and by default are located in `/etc/magnum/`. Links to sample configuration files can be found below:

policy.yaml

Warning: JSON formatted policy file is deprecated since Magnum 12.0.0 (Wallaby). This oslopolicy-convert-json-to-yaml tool will migrate your existing JSON-formatted policy file to YAML in a backward-compatible way.

Use the `policy.yaml` file to define additional access controls that apply to the Container Infrastructure Management service:

```
#"context_is_admin": "role:admin"

#"admin_or_owner": "is_admin:True or project_id:%(project_id)s"

#"admin_api": "rule:context_is_admin"

#"admin_or_user": "is_admin:True or user_id:%(user_id)s"

#"cluster_user": "user_id:%(trustee_user_id)s"

#"deny_cluster_user": "not domain_id:%(trustee_domain_id)s"

# Create a new bay.
# POST  /v1/bays
#"bay:create": "rule:deny_cluster_user"

# Delete a bay.
# DELETE  /v1/bays/{bay_ident}
#"bay:delete": "rule:deny_cluster_user"

# Retrieve a list of bays with detail.
# GET  /v1/bays
#"bay:detail": "rule:deny_cluster_user"

# Retrieve information about the given bay.
# GET  /v1/bays/{bay_ident}
#"bay:get": "rule:deny_cluster_user"

# Retrieve a list of bays.
# GET  /v1/bays/
#"bay:get_all": "rule:deny_cluster_user"

# Update an existing bay.
# PATCH  /v1/bays/{bay_ident}
#"bay:update": "rule:deny_cluster_user"

# Create a new baymodel.
# POST  /v1/baymodels
#"baymodel:create": "rule:deny_cluster_user"

# Delete a baymodel.
# DELETE  /v1/baymodels/{baymodel_ident}
#"baymodel:delete": "rule:deny_cluster_user"

# Retrieve a list of baymodel with detail.
# GET  /v1/baymodels
#"baymodel:detail": "rule:deny_cluster_user"

# Retrieve information about the given baymodel.
# GET  /v1/baymodels/{baymodel_ident}
```

(continues on next page)

(continued from previous page)

```
#"baymodel:get": "rule:deny_cluster_user"

# Retrieve a list of baymodel.
# GET  /v1/baymodels
#"baymodel:get_all": "rule:deny_cluster_user"

# Update an existing baymodel.
# PATCH  /v1/baymodels/{baymodel_ident}
#"baymodel:update": "rule:deny_cluster_user"

# Publish an existing baymodel.
# POST  /v1/baymodels
# PATCH  /v1/baymodels
#"baymodel:publish": "rule:admin_api"

# Sign a new certificate by the CA.
# POST  /v1/certificates
#"certificate:create": "rule:admin_or_user or rule:cluster_user"

# Retrieve CA information about the given bay/cluster.
# GET  /v1/certificates/{bay_uuid/cluster_uuid}
#"certificate:get": "rule:admin_or_user or rule:cluster_user"

# Rotate the CA certificate on the given bay/cluster.
# PATCH  /v1/certificates/{bay_uuid/cluster_uuid}
#"certificate:rotate_ca": "rule:admin_or_owner"

# Create a new cluster.
# POST  /v1/clusters
#"cluster:create": "rule:deny_cluster_user"

# Delete a cluster.
# DELETE  /v1/clusters/{cluster_ident}
#"cluster:delete": "rule:deny_cluster_user"

# Delete a cluster from any project.
# DELETE  /v1/clusters/{cluster_ident}
#"cluster:delete_all_projects": "rule:admin_api"

# Retrieve a list of clusters with detail.
# GET  /v1/clusters
#"cluster:detail": "rule:deny_cluster_user"

# Retrieve a list of clusters with detail across projects.
# GET  /v1/clusters
#"cluster:detail_all_projects": "rule:admin_api"

# Retrieve information about the given cluster.
# GET  /v1/clusters/{cluster_ident}
```

(continues on next page)

(continued from previous page)

```
#"cluster:get": "rule:deny_cluster_user"

# Retrieve information about the given cluster across projects.
# GET  /v1/clusters/{cluster_ident}
#"cluster:get_one_all_projects": "rule:admin_api"

# Retrieve a list of clusters.
# GET  /v1/clusters/
#"cluster:get_all": "rule:deny_cluster_user"

# Retrieve a list of all clusters across projects.
# GET  /v1/clusters/
#"cluster:get_all_all_projects": "rule:admin_api"

# Update an existing cluster.
# PATCH  /v1/clusters/{cluster_ident}
#"cluster:update": "rule:deny_cluster_user"

# Update the health status of an existing cluster.
# PATCH  /v1/clusters/{cluster_ident}
#"cluster:update_health_status": "rule:admin_or_user or rule:cluster_user"

# Update an existing cluster.
# PATCH  /v1/clusters/{cluster_ident}
#"cluster:update_all_projects": "rule:admin_api"

# Resize an existing cluster.
# POST  /v1/clusters/{cluster_ident}/actions/resize
#"cluster:resize": "rule:deny_cluster_user"

# Upgrade an existing cluster.
# POST  /v1/clusters/{cluster_ident}/actions/upgrade
#"cluster:upgrade": "rule:deny_cluster_user"

# Upgrade an existing cluster across all projects.
# POST  /v1/clusters/{cluster_ident}/actions/upgrade
#"cluster:upgrade_all_projects": "rule:admin_api"

# Create a new cluster template.
# POST  /v1/clustertemplates
#"clustertemplate:create": "rule:deny_cluster_user"

# Delete a cluster template.
# DELETE  /v1/clustertemplate/{clustertemplate_ident}
#"clustertemplate:delete": "rule:admin_or_owner"

# Delete a cluster template from any project.
# DELETE  /v1/clustertemplate/{clustertemplate_ident}
#"clustertemplate:delete_all_projects": "rule:admin_api"
```

(continues on next page)

(continued from previous page)

```
# Retrieve a list of cluster templates with detail across projects.
# GET  /v1/clustertemplates
#"clustertemplate:detail_all_projects": "rule:admin_api"

# Retrieve a list of cluster templates with detail.
# GET  /v1/clustertemplates
#"clustertemplate:detail": "rule:deny_cluster_user"

# Retrieve information about the given cluster template.
# GET  /v1/clustertemplate/{clustertemplate_ident}
#"clustertemplate:get": "rule:deny_cluster_user"

# Retrieve information about the given cluster template across
# project.
# GET  /v1/clustertemplate/{clustertemplate_ident}
#"clustertemplate:get_one_all_projects": "rule:admin_api"

# Retrieve a list of cluster templates.
# GET  /v1/clustertemplates
#"clustertemplate:get_all": "rule:deny_cluster_user"

# Retrieve a list of cluster templates across projects.
# GET  /v1/clustertemplates
#"clustertemplate:get_all_all_projects": "rule:admin_api"

# Update an existing cluster template.
# PATCH  /v1/clustertemplate/{clustertemplate_ident}
#"clustertemplate:update": "rule:admin_or_owner"

# Update an existing cluster template.
# PATCH  /v1/clustertemplate/{clustertemplate_ident}
#"clustertemplate:update_all_projects": "rule:admin_api"

# Publish an existing cluster template.
# POST  /v1/clustertemplates
# PATCH  /v1/clustertemplates
#"clustertemplate:publish": "rule:admin_api"

# Create a new federation.
# POST  /v1/federations
#"federation:create": "rule:deny_cluster_user"

# Delete a federation.
# DELETE  /v1/federations/{federation_ident}
#"federation:delete": "rule:deny_cluster_user"

# Retrieve a list of federations with detail.
# GET  /v1/federations
```

(continues on next page)

(continued from previous page)

```
#"federation:detail": "rule:deny_cluster_user"

# Retrieve information about the given federation.
# GET  /v1/federations/{federation_ident}
#"federation:get": "rule:deny_cluster_user"

# Retrieve a list of federations.
# GET  /v1/federations/
#"federation:get_all": "rule:deny_cluster_user"

# Update an existing federation.
# PATCH  /v1/federations/{federation_ident}
#"federation:update": "rule:deny_cluster_user"

# Retrieve a list of magnum-services.
# GET  /v1/mservices
#"magnum-service:get_all": "rule:admin_api"

# Create quota.
# POST  /v1/quotas
#"quota:create": "rule:admin_api"

# Delete quota for a given project_id and resource.
# DELETE  /v1/quotas/{project_id}/{resource}
#"quota:delete": "rule:admin_api"

# Retrieve Quota information for the given project_id.
# GET  /v1/quotas/{project_id}/{resource}
#"quota:get": "rule:admin_or_owner"

# Retrieve a list of quotas.
# GET  /v1/quotas
#"quota:get_all": "rule:admin_api"

# Update quota for a given project_id.
# PATCH  /v1/quotas/{project_id}/{resource}
#"quota:update": "rule:admin_api"

# Retrieve magnum stats.
# GET  /v1/stats
#"stats:get_all": "rule:admin_or_owner"

# Retrieve information about the given nodegroup.
# GET  /v1/clusters/{cluster_id}/nodegroup/{nodegroup}
#"nodegroup:get": "rule:admin_or_owner"

# Retrieve a list of nodegroups that belong to a cluster.
# GET  /v1/clusters/{cluster_id}/nodegroups/
#"nodegroup:get_all": "rule:admin_or_owner"
```

(continues on next page)

(continued from previous page)

```
# Retrieve a list of nodegroups across projects.
# GET  /v1/clusters/{cluster_id}/nodegroups/
#"nodegroup:get_all_all_projects": "rule:admin_api"

# Retrieve information for a given nodegroup.
# GET  /v1/clusters/{cluster_id}/nodegroups/{nodegroup}
#"nodegroup:get_one_all_projects": "rule:admin_api"

# Create a new nodegroup.
# POST  /v1/clusters/{cluster_id}/nodegroups/
#"nodegroup:create": "rule:admin_or_owner"

# Delete a nodegroup.
# DELETE  /v1/clusters/{cluster_id}/nodegroups/{nodegroup}
#"nodegroup:delete": "rule:admin_or_owner"

# Update an existing nodegroup.
# PATCH  /v1/clusters/{cluster_id}/nodegroups/{nodegroup}
#"nodegroup:update": "rule:admin_or_owner"
```

CHAPTER

NINE

WORK IN PROGRESS