Decapod Documentation

Release 0.1.2

Sergey Arkhipov

Apr 06, 2017

Contents

Note: After release 0.1 Shrimp was renamed into Decapod. This guide follows new naming policy but package names are still *shrimp_**.

CHAPTER 1

Introduction

Decapod is a tool that simplifies the deployment and lifecycle management of Ceph. Using Decapod, you can deploy clusters with best known practices, add new nodes to a cluster, remove them, and purge a cluster, if required. Decapod provides a simple API to manage cluster configurations. Also, you can use the Decapod web UI to easily manage your clusters.

Decapod uses Ansible with the ceph-ansible community project to deliver the best user experience. For tasks, you can use plugins that encapsulate the appropriate settings. Also, you can customize the configuration before execution, if required.

Decapod provides the following functionality:

- Deploying Ceph on remote nodes
- Adding and removing Ceph roles on machine (for example, deploying an OSD or removing a monitor)
- Purging a cluster
- Upgrading and updating clusters
- Managing partitions on disk devices for Ceph

However, Decapod does not cover:

- Providing a server for PXE
- Managing DHCP
- Managing networks by all means
- Managing host OS packages
- Deploying OS
- Managing partitions on disks that are not related to Ceph

See also:

- Ceph
- Ansible

- ceph-ansible community project
- Decapod API reference

CHAPTER 2

Install and configure Decapod

This section describes how to install and configure Decapod and includes the following topics:

Prerequisites

You can build Decapod on any commodity node that has Linux or OS X. However, prior to installing Decapod, verify that your software configurations meet the following requirements:

1. Machine should have `pip setuptools>=26` Python packages (does not matter, Python2 or Python3 is used)
2. Machine should have `npm>=3` (required to build frontend)
3. Machine should have `git`, `make` and `docker-engine` installed.
4. Machine should have an access to external network

Build images

Debian / Ubuntu 14.04 installation

First, you need to install build dependencies:

```
$ sudo apt-get update
$ sudo apt-get install -y curl git python python-pip python-setuptools make
```

Next, you need to install node>=4.x and npm. Unfortunately, trusty and debian stable have obsolete versions in their repositories, so follow this guide: https://github.com/nodesource/distributions#debian-and-ubuntu-based-distributions

Also, you need to update setuptools to aversion >=26 (previous versions have critical bugs in building wheels with big sets of package data)

```
$ sudo pip install 'setuptools>=26'
```

Ubuntu 16.04 installation

Please follow the same guide as for trusty installation, but you do not need to install external node: valid packages are in the official repository now.

```
$ sudo apt update
$ sudo apt install -y nodejs nodejs-legacy npm
```

Updating of setuptools is also required.

```
$ sudo pip install 'setuptools>=26'
```

Docker-engine installation

To install docker-engine, please follow the official instructions. Also, please pay attention to the DNS configuration.

Cloning of source code repository

```
$ git clone --recurse-submodules \
  https://github.com/Mirantis/ceph-lcm.git decapod
$ cd decapod
```

Inside repository, please check available versions with git tag. To select specific version, please do `git checkout {tag} && git submodule update --init --recursive`.

Building a development version

There is little difference between production and development build. The difference is only in SSH private keys, SSL certificate and configuration file. In development variant, they are pregenerated and placed in `containerization/files` directory of the source code.

To build development images, just execute the following command:

```
$ make build_containers_dev
```

Actually, there is not big difference between production and development version. Basically, target `build_containers_dev` is a sequence of 2 targets: `copy_example_keys` and `build_containers`. Target `copy_example_keys` copies hardcoded files, mentioned in *Building a production version* into correct places.

Since these files are placed in VCS, user has to replace them with private ones on container build.

Building a production version

To build a production version, you need to have your own configuration file, SSH private key for Ansible and SSL certificate for web frontend. Please check the next section for details. After you place required files in the top level directory of the source code repository, execute the following command:

```
$ make build_containers
```

As a summary, to build production containers, you need to have the following in the top level directory of your source code repository:

ansible_ssh_keyfile.pem SSH private key which should be used by Ansible to connect to Ceph nodes.

ssl.key Private key for SSL/TLS certificate which should be used by web UI.

ssl.crt Signed certificate for SSL/TLS which should be used by web UI.

ssl-dhparam.pem Diffie-Hellman ephemeral parameters for SSL/TLS. This enables perfect-forward secrecy for secured connection.

config.yaml Configuration file for Decapod.

mongodb.pem SSL/TLS pair of certificate and key, concatenated in one file. Required to use secured connection by MongoDB.

SSH private keys

> **Warning:** Secrecy of the key is on you. Please keep it private.

Decapod uses Ansible to configure remote machines, Ansible uses SSH to connect to remote machines. Therefore, it is required to propagate SSH private key to Decapod. If you don't have a prepared SSH private key, you may generate a new one using the following guide: https://confluence.atlassian.com/bitbucketserver/creating-ssh-keys-776639788.html

After you generate a new one, copy it to the top level of the source code repository. It has to have name: `ansible_ssh_keyfile.pem`. The format of the file is PEM[1].

SSL certificate

> **Warning:** Secrecy of the key if on you. Please keep it private. Please do not use self-signed certificates for production installation.

SSL certificate should have 3 parts: private key for certificate, signed certificate and Diffie-Hellman ephemeral parameters.

If you have no such certificates, you may generate new ones using the following instructions:

- https://www.digitalocean.com/community/tutorials/openssl-essentials-working-with-ssl-certificates-private-keys-and-csrs
- https://raymii.org/s/tutorials/Strong_SSL_Security_On_nginx.html#Forward_Secrecy_&_Diffie_Hellman_Ephemeral_Parameters

All SSL keys should be in PEM[1] format.

Please put SSL files in the top level of your source code repository:

- *Private key* should be placed as `ssl.key`;
- *Signed certificate* should be placed as `ssl.crt`;
- *Diffie-Hellman parameters* should be placed as `ssl-dhparam.pem`.

[1] https://tools.ietf.org/html/rfc1421

Configuration

Configuration for Decapod is done in YAML[2] format. Please check the example in `containerization/files/devconfigs/config.yaml`.

MongoDB secured connection

To allow SSL/TLS for MongoDB connection, you have to have generated private key and certificate. Mongo allows to use unified PEM file which contains both items. To get information on generation of such file, please refer official documentation: https://docs.mongodb.com/manual/tutorial/configure-ssl/#pem-file

To allow SSL/TLS on client side, please be sure that config file has `?ssl=true` parameter in URI. For example, `mongodb://database:27017/db` won't use secured connection, but `mongodb://database:27017/db?ssl=true` will.

MongoDB authorization/authentication

Note: By default, containers will have no information about users and their passwords.

To use db authentication, please follow the official guide or a community checklist:

- https://docs.mongodb.com/manual/core/security-users/
- https://gist.github.com/leommoore/f977860d22dfb2860fc2
- https://hub.docker.com/_/mongo/

After you have a MongoDB running with the required authentication, please make sure that user/password pair is set in config file. URI should look like `mongodb://user:password@database:27017/db?ssl=true`.

Move images

In the future, it will be possible to run decapod services on different machines, but this guide assumes that you have only one machine with docker and docker-compose. There may be one build machine and another production one. If you have such a diversity, please use Docker registry to manage Decapod images or dump/load them manually. To do so, build images and execute the following commands:

```
$ make dump_images
$ rsync -a output/images/ <remote_machine>:images/
$ scp docker-compose.yml <remote_machine>:docker-compose.yml
$ ssh <remote_machine>
$ cd images
$ for i in $(\ls -1 *.bz2); do docker load -i "$i"; done;
$ cd ..
$ docker-compose up
```

This will dump docker images, copy them to remote host and load. After that it will be possible to run docker-compose.

[2] http://www.yaml.org/spec/1.2/spec.html

Run containers with Decapod

When you have your docker images built, it is time to run Decapod. To do that, you do not need to have source code at all, images are enough. Minimal requirements to run Decapod in containers are installed docker-engine and docker-compose.

To install docker-engine, please check instructions for build machine. To install docker-compose, please check the official guide: https://docs.docker.com/compose/install/

Important: docker-compose should be version 1.6 or later. Please make sure, that you are running a proper one:

```
$ pip install 'docker-compose>=1.6'
```

The next thing you need, is configuration for docker-compose. It is placed in the top level of the source code repository as docker-compose.yml. Place it in any place you like. After that, run it as follows

```
$ docker-compose up
```

This will start the service. Docker-compose will use 2 socket binds: 0.0.0.0:9999 and 0.0.0.0:10000. Port 9999 is HTTP endpoint, 10000 - HTTPS. Assuming, that your Decapod production machine uses IP 10.10.0.2, you may access UI with http://10.10.0.2:9999 or https://10.10.0.2:10000.

CHAPTER 3

Data models

Decapod is used to deploy and manage Ceph clusters. All the management functionality is distributed using plugins, called playbooks. Each playbook requires configuration. This section describes the Decapod data models and entities, workflows, and terms used in other sections.

The section contains the following topics:

User model

A user is an entity that contains common information about the Decapod user. It has a login, email, password, full name, and a role. The user model is used for authentication and authorization purposes.

When creating a user model in the system, Decapod sends the new password to the user email. It is possible to reset the password and set a new one.

A user created without a role can do a bare minimum with the system because even listing the entities requires permissions. Authorization is performed by assigning a role to the user. A user may have only one role in Decapod.

See also:

- *Role model*

Role model

A role has two properties: name and permissions. Consider the role as a named set of permissions. Decapod has two types of permissions:

- API permissions allow using different API endpoints and, therefore, a set of actions available for usage. For example, to view the list of users, you need to have the `view_user` permission. To modify the information about a user, you also require the `edit_user` permission.

Note: Some API endpoints require several permissions. For example, user editing requires both `view_user` and `edit_user` permissions.

- Playbook permissions define a list of playbooks that a user can execute. For example, a user with any role can execute service playbooks to safely update a host package or add new OSDs. But a user requires special permissions to execute destructive playbooks, such as purging a cluster or removing OSD hosts.

Server model

The server model defines a server used for Ceph purposes. Servers are detected during the server discovery process. Each server has a name (FQDN by default), IP, FQDN, state, cluster ID, and facts. A user is only allowed to modify the server name, other attributes are updated automatically on the server discovery. The facts property is a set of facts collected by Ansible and returned as is. By default, Ansible collects only its own facts, ignoring Ohai and Facter.

Note: We do not recommend that you manually create a new server using the API. Servers must be discovered by the discovery protocol.

Server discovery is an automatic process of discovering new servers in Decapod. During this process, Decapod works passively.

Important: A node operating system deployment is not in the scope of Decapod. The server discovery is performed using `cloud-init`, so the only requirement for the node OS is to support `cloud-init`.

The cloud-init package is required to create a user for Ansible, set the deployment SSH public key for the user's authorized keys, and update the `/etc/rc.local` script. Then, the `/etc/rc.local` script registers the host in Decapod.

See also:

- Ohai
- Facter
- The cloud-init documentation

Cluster model

A cluster defines a separate Ceph cluster. It has a default name that you can edit only explicitly. You can delete only the cluster that has no servers in it.

An explicit cluster model is required because it defines a name of FSID for Ceph. By default, the name of the model is used as a name of the Ceph cluster and its ID as FSID.

The cluster model configuration is a simple mapping of roles to the list of servers. You cannot manage this configuration explicitly. Instead, you can use playbooks. For example, when executing the playbook for adding a new OSD host, this host will be added to the list of servers for role `osds`. If you remove Rados Gateways from the clusters using an appropriate playbook, these servers will be deleted from the list.

Several models are required to deploy a cluster. Basically, cluster deployment contains the following steps:

1. Creating an empty cluster model. This model is a holder for the cluster configuration. Also, it defines the Ceph FSID and name.
2. Creating a playbook configuration model for the `deploy_cluster` playbook. This will allow you to deploy the cluster.

   Note: Cluster deployment is an idempotent operation and you may execute it several times.

3. Executing that playbook configuration by creating a new execution. If required, examine the execution steps or logs.

See also:

- *Playbook configuration*
- *Playbook execution*

Decapod playbooks

Decapod uses plugins to deliver the Ceph management functionality. A plugin is a Python package that contains Ansible playbooks, a configuration file, and the Python code itself.

This section describes the Decapod playbooks and contains the following topics:

Playbook configuration

In most cases, Ansible playbooks are generic and have the capability to inject values: not only the hosts where a playbook has to be executed but also some arbitrary parameters, for example, Ceph FSID. These parameters are injected into the Ansible playbooks using the `--extra-vars` option or by setting them in inventory. A playbook configuration defines the name of the playbook and its parameters. For simplicity, parameters are split into two sections:

- The `global_vars` section contains the global variables for a playbook. Each parameter in the `global_vars` section is defined for all hosts. However, the `inventory` section redefines any parameters.
- The `inventory` section is used as the Ansible inventory. Mostly, this will be a real inventory. You can change the section to exclude sensitive information, for example. But in most cases, the `inventory` parameters are used as is.

Note: Parameters from the `global_vars` section will be passed as the `--extra-vars` parameters. For details, see the Ansible official documentation.

Basically, configuring a playbook includes:

1. Placing the contents of `global_vars` into `./inventoryfile`.
2. Executing the following command:

```
$ ansible-playbook -i ./inventoryfile --extra-vars "inventory_section|to_json"␣
↪playbook.yaml
```

Decapod generates the best possible configuration for a given set of *Server model* models. After that, modify it as required.

Note: Decapod uses the server IP as a host. This IP is the IP of the machine visible to Decapod and does not belong to any network other than the one used by Decapod to SSH on the machine.

Creating a playbook configuration supports optional hints. Hints are the answers to simple questions understandable by plugins. With hints, you can generate more precise configuration. For example, if you set the `dmcrypt` hint for a cluster deployment, Decapod will generate the configuration with dmcrypted OSDs.

To see the available hints, use the `GET /v1/playbook` API endpoint or see *Playbook plugins*.

See also:

- *Playbook execution*

Playbook execution

The execution model defines the execution of a playbook configuration. You can run each playbook configuration several times, and this model defines a single execution. As a result, you receive the execution status (completed, failed, and others) and the execution log. The execution log can be shown as:

- Execution steps, which are the parsed steps of the execution.
- Raw log, which is a pure Ansible log of the whole execution as is, taken from `stdout` and `stderr`.

Each execution step has timestamps (started, finished), ID of the server that issued the event, role and task name of the event, status of the task, and detailed information on the error, if any.

See also:

- *Playbook configuration*

CHAPTER 4

Manage users and roles

This section descibes how to manage users and roles in Decapod through the web UI and contains the following topics:

Manage users

To add a new user:

1. Log in to the Decapod web UI.
2. Navigate to *USERS MANAGEMENT*.
3. Click the *USERS* tab.
4. Click *CREATE NEW USER* and type the required data.
5. Click *SAVE*. A new user has been created.

> **Note:** The password is sent to the user email. This password can be changed.

After saving the changes, you will see that the *CHANGELOG* is updated. This *CHANGELOG* tracks all the results and it is possible to view the details about a user modifications. This is related not only to the user management. Decapod stores all changes and you can always obtain the entire log.

Clicking *DELETE USER* does not delete the user but archives it instead. You can still access the user through the Decapod CLI if you know the user ID.

Manage roles

The following table describes how to create, edit, and delete roles through the Decapod web UI.

Task	Steps
Create a new role	1. In the Decapod web UI. Navigate to *USERS MANAGEMENT*. 2. Click the *ROLES* tab. 3. Click *CREATE NEW ROLE*. 4. Type the role name and select the required permissions. 5. Click *SAVE CHANGES*.
Edit a role	1. In the Decapod web UI, navigate to *USERS MANAGEMENT*. 2. Click the *ROLES* tab. 3. Click the pen icon near the required role name and edit the role as required.
Delete a role	1. In the Decapod web UI, navigate to *USERS MANAGEMENT*. 2. Click the *ROLES* tab. 3. Click the trash can icon near the required role name. **Note:** This will not completely delete the role but will archive it instead. You can access the role through the Decapod CLI if you know the role ID.
Assign a role to a user	1. In the Decapod web UI, navigate to *USERS MANAGEMENT*. 2. Click the *USERS* tab. 3. Expand the required user. 4. Select the required role in the *ROLE* section. 5. Click *SAVE*.

See also:

- *Role model*

CHAPTER 5

Deploy a cluster

This section describes the cluster deployment workflow using the Decapod web UI.

The section contains the following topics:

Create a cluster

To create a cluster:

1. Log in to the Decapod web UI.
2. Navigate to *CLUSTER*.
3. Click *CREATE NEW CLUSTER*.
4. Type the cluster name and click *SAVE*.

A new cluster is empty and contains no servers. Discover servers as described in *Discover a server*.

View servers

Verify that you have discovered the required servers as described in *Discover a server*.

To view the discovered servers:

1. Log in to the Decapod web UI.
2. Navigate to *SERVERS*. The *SERVERS* page lists the servers accessible by Decapod.
3. Expand the required server to view its details.

Create a playbook configuration

To create a playbook configuration:

1. Log in to the Decapod web UI.
2. Navigate to *PLAYBOOK CONFIGURATION*.
3. Click *CREATE NEW CONFIGURATION*.
4. Type the configuration name and select a cluster, then click *NEXT*.
5. Select the required playbook and click *NEXT*.

   The table lists the plugins available for execution. Some playbooks require an explicit list of servers. For example, to purge a cluster, Decapod will use the servers in this cluster and you do not need to specify them manually.
6. In the *SELECT SERVERS* window, select all servers and click *SAVE CHANGES*. Once the new playbook configuration is created, you will see the *PLAYBOOK CONFIGURATION* window.
7. Edit the playbook configuration, if required, and click *SAVE CHANGES*.

Execute a playbook configuration

To execute a playbook configuration:

1. In the Decapod web UI, navigate to *CONFIGURATIONS*.
2. Click *EXECUTE* to execute the required configuration. Once the execution starts, its state changes to *STARTED* on the *EXECUTIONS* page.
3. To view the execution process, click *LOGS*.
4. Once the execution is finished, its status will change to *COMPLETED*. To download the entire execution log, click *DOWNLOAD*.

CHAPTER 6

Use the Decapod CLI

This section contains the following topics:

Install the Decapod CLI

To install the Decapod CLI on a local machine, install two packages:

- `decapodlib`, the RPC client library to access the Decapod API
- `decapod-cli`, the CLI wrapper for the library

To install the Decapod CLI:

1. At the top level of the source code repository, run the following command to build the packages and place them to the `output/eggs` directory:

```
$ make build_eggs
```

2. Install the packages:

```
$ pip install output/eggs/decapodlib*.whl output/eggs/decapod_cli*.whl
```

3. Run **`decapod`** to verify the installation.

See also:

- *Access the Decapod CLI*

Access the Decapod CLI

To access Decapod, you need to know its URL (http://10.10.0.2:9999 or https://10.10.0.2:10000), your username and password (`root`/`root` for a development installation).

To access Decapod using CLI:

1. Set your credentials directly to the Decapod CLI or use the environment variables:

```
export DECAPOD_URL=http://10.10.0.2:9999
export DECAPOD_LOGIN=root
export DECAPOD_PASSWORD=root
```

   Save this to a file and source when required.

2. Verify that it works:

```
$ decapod -u http://10.10.0.2:9999 -l root -p root user get-all
```

   If you used environment variables, run:

```
$ decapod user get-all
```

Cluster deployment workflow

This section describes the cluster deployment workflow and contains the following topics:

Create a cluster

To create a cluster:

1. Verify that you can log in to the Decapod using CLI.
2. To create a cluster, run:

```
$ decapod cluster create <CUSTER_NAME>
```

   Example:

```
$ decapod cluster create ceph
{
    "data": {
        "configuration": {},
        "name": "ceph"
    },
    "id": "f2621e71-76a3-4e1a-8b11-fa4ffa4a6958",
    "initiator_id": "7e47d3ff-3b2e-42b5-93a2-9bd2601500d7",
    "model": "cluster",
    "time_deleted": 0,
    "time_updated": 1479902503,
    "version": 1
}
```

   As a result, a new cluster with the name `ceph` and ID `f2621e71-76a3-4e1a-8b11-fa4ffa4a6958` has been created. This ID is required for creating the playbook configuration.

3. Proceed to *Discover a server*.

Discover a server

To discover a server:

1. Generate the user-data configuration for `cloud-init`. For details, see *Generate user data*.

   The `cloud-init` execution generates the content of `/etc/rc.local`. The first and next reboots will call the Decapod API for server registering. Such registration is an idempotent operation. The execution of the Decapod API (POST /v1/server) creates a task for the controller server on facts discovery. The controller executes this task and collects facts from the remote host. A new server model is created or the information on the existing one is updated.

2. With this configuration, deploy an operating system on a Ceph node. For an example of such deployment, see: *Deploy an operating system on a Ceph node*, official cloud-init documentation, or use kernel parameters.

As a result, the server should be listed in Decapod. The server discovery takes time because of `cloud-init`. Therefore, the server may appear in five minutes after deployment. Once the server appears in Decapod, the tool can use it.

See also:

- *Server model*

Create a playbook configuration

To create a playbook configuration:

1. List the existing playbooks:

```
$ decapod playbook get-all
{
    "items": [
        {
            "description": "Adding new OSD to the cluster.\n\nThis plugin adds␣
↪OSD to the existing cluster.",
            "id": "add_osd",
            "name": "Add OSD to Ceph cluster",
            "required_server_list": true
        },
        {
            "description": "Ceph cluster deployment playbook.\n\nThis plugin␣
↪deploys Ceph cluster into a set of servers. After sucessful\ndeployment,␣
↪cluster model will be updated.",
            "id": "cluster_deploy",
            "name": "Deploy Ceph cluster",
            "required_server_list": true
        },
        {
            "description": "Example plugin for playbook.\n\nThis plugin deploys␣
↪simple hello world service on remote machine If\nremote machine host is
↪'hostname', \
            then http://hostname:8085 will\nrespond with '{\"result\": \"ok\"}'␣
↪JSON.",
            "id": "hello_world",
            "name": "Hello World",
            "required_server_list": false
        },
        {
            "description": "Purge whole Ceph cluster.\n\nThis plugin purges whole␣
↪Ceph cluster. It removes packages, all data,\nreformat Ceph devices.",
            "id": "purge_cluster",
            "name": "Purge cluster",
            "required_server_list": false
```

```
        },
        {
            "description": "Remove OSD host from cluster.",
            "id": "remove_osd",
            "name": "Remove OSD host from Ceph cluster",
            "required_server_list": true
        }
    ]
}
```

This will list the available playbooks in details. The `name` and `description` are the human-readable items to display in the Decapod UI.

2. Note the ID of the Ceph cluster deployment playbook. It is `cluster_deploy` in the example above.
3. The cluster deployment playbook requires a list of servers to operate with (field `required_server_list` is `true`). To list the available servers:

```
$ decapod server get-all
```

Note: The output of this command can be quite long. Therefore, we recommend that you use a tool for listing. One of the best tools available to work with JSON in CLI is jq.

4. Obtain the required server IDs:
   - Extract the IDs manually
   - Use compact listing:

```
$ decapod server get-all --compact
"machine_id","version","fqdn","username","default_ip","interface=mac=ipv4=ipv6
↪","..."
"015fd324-4437-4f28-9f4b-7e3a90bdc30f","1","chief-gull.maas","ansible","10.10.
↪0.9","ens3=52:54:00:29:14:22=10.10.0.9=fe80::5054:ff:fe29:1422"
"7e791f07-845e-4d70-bff1-c6fad6bfd7b3","1","exotic-swift.maas","ansible","10.
↪10.0.11","ens3=52:54:00:05:b0:54=10.10.0.11=fe80::5054:ff:fe05:b054"
"70753205-3e0e-499d-b019-bd6294cfbe0f","1","helped-pig.maas","ansible","10.10.
↪0.12","ens3=52:54:00:01:7c:1e=10.10.0.12=fe80::5054:ff:fe01:7c1e"
"40b96868-205e-48a2-b8f6-3e3fcfbc41ef","1","joint-feline.maas","ansible","10.
↪10.0.10","ens3=52:54:00:4a:c3:6d=10.10.0.10=fe80::5054:ff:fe4a:c36d"
"8dd33842-fee6-4ec7-a1e5-54bf6ae24710","1","polite-rat.maas","ansible","10.10.
↪0.8","ens3=52:54:00:d4:da:29=10.10.0.8=fe80::5054:ff:fed4:da29"
```

     Where `machine_id` is the server ID.

   - Use the `jq` tool mentioned above:

```
$ decapod server get-all | jq -rc '.[]|.id'
015fd324-4437-4f28-9f4b-7e3a90bdc30f
7e791f07-845e-4d70-bff1-c6fad6bfd7b3
70753205-3e0e-499d-b019-bd6294cfbe0f
40b96868-205e-48a2-b8f6-3e3fcfbc41ef
8dd33842-fee6-4ec7-a1e5-54bf6ae24710
```

Note: We recommend using the `jq` tool as the compact representation shows only a limited amount of information. Using `jq` allows you to extract any certain data.

5. At this step you should have all the required data to create a playbook configuration:
   - The cluster name (can be any)
   - The playbook name
   - The cluster ID
   - The server IDs
6. Create a playbook configuration using the following command:

```
$ decapod playbook-configuration create <NAME> <PLAYBOOK> <CLUSTER_ID> [SERVER_
↪IDS]...
```

Example:

```
$ decapod playbook-configuration create deploy cluster_deploy f2621e71-76a3-4e1a-
↪8b11-fa4ffa4a6958 015fd324-4437-4f28-9f4b-7e3a90bdc30f \
7e791f07-845e-4d70-bff1-c6fad6bfd7b3 70753205-3e0e-499d-b019-bd6294cfbe0f␣
↪40b96868-205e-48a2-b8f6-3e3fcfbc41ef 8dd33842-fee6-4ec7-a1e5-54bf6ae24710
{
    "data": {
        "cluster_id": "f2621e71-76a3-4e1a-8b11-fa4ffa4a6958",
        "configuration": {
            "global_vars": {
                "ceph_facts_template": "/usr/local/lib/python3.5/dist-packages/
↪decapod_common/facts/ceph_facts_module.py.j2",
                "ceph_stable": true,
                "ceph_stable_distro_source": "jewel-xenial",
                "ceph_stable_release": "jewel",
                "ceph_stable_repo": "http://eu.mirror.fuel-infra.org/shrimp/ceph/
↪apt",
                "cluster": "ceph",
                "cluster_network": "10.10.0.0/24",
                "copy_admin_key": true,
                "fsid": "f2621e71-76a3-4e1a-8b11-fa4ffa4a6958",
                "journal_collocation": true,
                "journal_size": 100,
                "max_open_files": 131072,
                "nfs_file_gw": false,
                "nfs_obj_gw": false,
                "os_tuning_params": [
                    {
                        "name": "fs.file-max",
                        "value": 26234859
                    },
                    {
                        "name": "kernel.pid_max",
                        "value": 4194303
                    }
                ],
                "public_network": "10.10.0.0/24"
            },
            "inventory": {
                "_meta": {
                    "hostvars": {
                        "10.10.0.10": {
                            "ansible_user": "ansible",
                            "devices": [
```

```
                        "/dev/vdc",
                        "/dev/vde",
                        "/dev/vdd",
                        "/dev/vdb"
                    ],
                    "monitor_interface": "ens3"
                },
                "10.10.0.11": {
                    "ansible_user": "ansible",
                    "devices": [
                        "/dev/vdc",
                        "/dev/vde",
                        "/dev/vdd",
                        "/dev/vdb"
                    ],
                    "monitor_interface": "ens3"
                },
                "10.10.0.12": {
                    "ansible_user": "ansible",
                    "devices": [
                        "/dev/vdc",
                        "/dev/vde",
                        "/dev/vdd",
                        "/dev/vdb"
                    ],
                    "monitor_interface": "ens3"
                },
                "10.10.0.8": {
                    "ansible_user": "ansible",
                    "devices": [
                        "/dev/vdc",
                        "/dev/vde",
                        "/dev/vdd",
                        "/dev/vdb"
                    ],
                    "monitor_interface": "ens3"
                },
                "10.10.0.9": {
                    "ansible_user": "ansible",
                    "devices": [
                        "/dev/vdc",
                        "/dev/vde",
                        "/dev/vdd",
                        "/dev/vdb"
                    ],
                    "monitor_interface": "ens3"
                }
            }
        },
        "clients": [],
        "iscsi_gw": [],
        "mdss": [],
        "mons": [
            "10.10.0.9"
        ],
        "nfss": [],
        "osds": [
            "10.10.0.10",
```

```
                    "10.10.0.12",
                    "10.10.0.11",
                    "10.10.0.8"
                ],
                "rbd_mirrors": [],
                "restapis": [
                    "10.10.0.9"
                ],
                "rgws": []
            }
        },
        "name": "deploy",
        "playbook_id": "cluster_deploy"
    },
    "id": "fd499a1e-866e-4808-9b89-5f582c6bd29e",
    "initiator_id": "7e47d3ff-3b2e-42b5-93a2-9bd2601500d7",
    "model": "playbook_configuration",
    "time_deleted": 0,
    "time_updated": 1479906402,
    "version": 1
}
```


Where the playbook configuration ID is `fd499a1e-866e-4808-9b89-5f582c6bd29e`.

Update a playbook configuration

You may need to update a playbook configuration, for example, to use another host for the monitor.

To do so, update the playbook model using one of the following ways:

- Edit the playbook and send to `stdin` of the **`decapod playbook-configuration update fd499a1e-866e-4808-9b89-5f582c6bd29e`** command where `fd499a1e-866e-4808-9b89-5f582c6bd29e` is the playbook configuration ID.
- Run an external editor with the `--model-editor` option. Using this option, the Decapod CLI downloads the model and sends its data field to the editor. After you save and close the editor, the updated model is sent to the Decapod API. To use this model, verify that your editor is set using the **`env | grep EDITOR`** command.
- Dump JSON with modifications and inject into the `--model` option.

Important: Avoid updating fields outside of the `data` field (that is why the `--model-editor` option shows only the `data` field). Sending the whole model back to the Decapod API allows keeping consistent behavior of the Decapod API.

To update a playbook configuration:

1. Run the **`decapod playbook-configuration update`** command with the `--model-editor` flag.

   Example:

```
$ decapod playbook-configuration update fd499a1e-866e-4808-9b89-5f582c6bd29e --
↪model-editor
{
    "data": {
        "cluster_id": "f2621e71-76a3-4e1a-8b11-fa4ffa4a6958",
        "configuration": {
            "global_vars": {
```

```
                "ceph_facts_template": "/usr/local/lib/python3.5/dist-packages/
↪decapod_common/facts/ceph_facts_module.py.j2",
                "ceph_stable": true,
                "ceph_stable_distro_source": "jewel-xenial",
                "ceph_stable_release": "jewel",
                "ceph_stable_repo": "http://eu.mirror.fuel-infra.org/shrimp/ceph/
↪apt",
                "cluster": "ceph",
                "cluster_network": "10.10.0.0/24",
                "copy_admin_key": true,
                "fsid": "f2621e71-76a3-4e1a-8b11-fa4ffa4a6958",
                "journal_collocation": true,
                "journal_size": 100,
                "max_open_files": 131072,
                "nfs_file_gw": false,
                "nfs_obj_gw": false,
                "os_tuning_params": [
                   {
                        "name": "fs.file-max",
                        "value": 26234859
                    },
                    {
                        "name": "kernel.pid_max",
                        "value": 4194303
                    }
                ],
                "public_network": "10.10.0.0/24"
            },
            "inventory": {
                "_meta": {
                    "hostvars": {
                        "10.10.0.10": {
                            "ansible_user": "ansible",
                            "devices": [
                                "/dev/vdc",
                                "/dev/vde",
                                "/dev/vdd",
                                "/dev/vdb"
                            ],
                            "monitor_interface": "ens3"
                        },
                        "10.10.0.11": {
                            "ansible_user": "ansible",
                            "devices": [
                                "/dev/vdc",
                                "/dev/vde",
                                "/dev/vdd",
                                "/dev/vdb"
                            ],
                            "monitor_interface": "ens3"
                        },
                        "10.10.0.12": {
                            "ansible_user": "ansible",
                            "devices": [
                                "/dev/vdc",
                                "/dev/vde",
                                "/dev/vdd",
                                "/dev/vdb"
```

```
                            ],
                            "monitor_interface": "ens3"
                        },
                        "10.10.0.8": {
                            "ansible_user": "ansible",
                            "devices": [
                                "/dev/vdc",
                                "/dev/vde",
                                "/dev/vdd",
                                "/dev/vdb"
                            ],
                            "monitor_interface": "ens3"
                        },
                        "10.10.0.9": {
                            "ansible_user": "ansible",
                            "devices": [
                                "/dev/vdc",
                                "/dev/vde",
                                "/dev/vdd",
                                "/dev/vdb"
                            ],
                            "monitor_interface": "ens3"
                        }
                    }
                },
                "clients": [],
                "iscsi_gw": [],
                "mdss": [],
                "mons": [
                    "10.10.0.8"
                ],
                "nfss": [],
                "osds": [
                    "10.10.0.10",
                    "10.10.0.12",
                    "10.10.0.11",
                    "10.10.0.9"
                ],
                "rbd_mirrors": [],
                "restapis": [
                    "10.10.0.8"
                ],
                "rgws": []
            }
        },
        "name": "deploy",
        "playbook_id": "cluster_deploy"
    },
    "id": "fd499a1e-866e-4808-9b89-5f582c6bd29e",
    "initiator_id": "7e47d3ff-3b2e-42b5-93a2-9bd2601500d7",
    "model": "playbook_configuration",
    "time_deleted": 0,
    "time_updated": 1479907354,
    "version": 2
}
```


The example above shows replacing `10.10.0.9` in `mons/restapis` and adding it to the OSD list, and also placing the `10.10.0.8` from OSDs to `mons/restapis`. As a result, the playbook configuration ID is

`fd499a1e-866e-4808-9b89-5f582c6bd29e` and the version is `2`.

2. Save your changes and exit the editor. Proceed to *Execute a playbook configuration*.

Execute a playbook configuration

To execute a playbook configuration:

1. Run **`decapod execution create`** with the playbook configuration ID and version.

   Example:

```
$ decapod execution create fd499a1e-866e-4808-9b89-5f582c6bd29e 2
{
    "data": {
        "playbook_configuration": {
            "id": "fd499a1e-866e-4808-9b89-5f582c6bd29e",
            "version": 2
        },
        "state": "created"
    },
    "id": "f2fbb668-6c89-42d2-9251-21e0b79ae973",
    "initiator_id": null,
    "model": "execution",
    "time_deleted": 0,
    "time_updated": 1479908503,
    "version": 1
}
```

   Once done, the playbook configuration is in the `created` state. It takes some time for the execution to start.

2. To verify that the execution has started, use the **`decapod execution get`** command with the execution ID.

   Example:

```
 $ decapod execution get f2fbb668-6c89-42d2-9251-21e0b79ae973
 {
     "data": {
         "playbook_configuration": {
             "id": "fd499a1e-866e-4808-9b89-5f582c6bd29e",
            "version": 2
        },
        "state": "started"
    },
    "id": "f2fbb668-6c89-42d2-9251-21e0b79ae973",
    "initiator_id": null,
    "model": "execution",
    "time_deleted": 0,
    "time_updated": 1479908503,
    "version": 2
}
```

   Once completed, the execution state will turn to `completed`.

Additionally, you can perform the following actions:

- Track the execution steps using the **`decapod execution steps`** command with the execution ID.

  Example:

```
$ decapod execution steps f2fbb668-6c89-42d2-9251-21e0b79ae973
[
    {
        "data": {
            "error": {},
            "execution_id": "f2fbb668-6c89-42d2-9251-21e0b79ae973",
            "name": "add custom repo",
            "result": "skipped",
            "role": "ceph.ceph-common",
            "server_id": "8dd33842-fee6-4ec7-a1e5-54bf6ae24710",
            "time_finished": 1479908609,
            "time_started": 1479908609
        },
        "id": "58359d01b3670f0089d9330b",
        "initiator_id": "f2fbb668-6c89-42d2-9251-21e0b79ae973",
        "model": "execution_step",
        "time_deleted": 0,
        "time_updated": 1479908609,
        "version": 1
    },
    {
        "data": {
            "error": {},
            "execution_id": "f2fbb668-6c89-42d2-9251-21e0b79ae973",
            "name": "add gluster nfs ganesha repo",
            "result": "skipped",
            "role": "ceph.ceph-common",
            "server_id": "8dd33842-fee6-4ec7-a1e5-54bf6ae24710",
            "time_finished": 1479908609,
            "time_started": 1479908609
        },
        "id": "58359d01b3670f0089d9330c",
        "initiator_id": "f2fbb668-6c89-42d2-9251-21e0b79ae973",
        "model": "execution_step",
        "time_deleted": 0,
        "time_updated": 1479908609,
        "version": 1
    }
]
```

- View the execution history using the **`decapod execution get-version-all`** command with the execution ID.

  Example:

```
$ decapod execution get-version-all f2fbb668-6c89-42d2-9251-21e0b79ae973
[
    {
        "data": {
            "playbook_configuration": {
                "id": "fd499a1e-866e-4808-9b89-5f582c6bd29e",
                "version": 2
            },
            "state": "completed"
        },
        "id": "f2fbb668-6c89-42d2-9251-21e0b79ae973",
        "initiator_id": null,
        "model": "execution",
```

```
        "time_deleted": 0,
        "time_updated": 1479909342,
        "version": 3
    },
    {
        "data": {
            "playbook_configuration": {
                "id": "fd499a1e-866e-4808-9b89-5f582c6bd29e",
                "version": 2
            },
            "state": "started"
        },
        "id": "f2fbb668-6c89-42d2-9251-21e0b79ae973",
        "initiator_id": null,
        "model": "execution",
        "time_deleted": 0,
        "time_updated": 1479908503,
        "version": 2
    },
    {
        "data": {
            "playbook_configuration": {
                "id": "fd499a1e-866e-4808-9b89-5f582c6bd29e",
                "version": 2
            },
            "state": "created"
        },
        "id": "f2fbb668-6c89-42d2-9251-21e0b79ae973",
        "initiator_id": null,
        "model": "execution",
        "time_deleted": 0,
        "time_updated": 1479908503,
        "version": 1
    }
]
```


- Once the execution is done, view the entire execution log using the **`decapod execution log`** command with the execution ID.

  Example:

```
$ decapod execution log f2fbb668-6c89-42d2-9251-21e0b79ae973
Using /etc/ansible/ansible.cfg as config file
statically included: /usr/local/lib/python2.7/dist-packages/decapod_ansible/ceph-
↪ansible/roles/ceph.ceph-common/tasks/./checks/check_system.yml
statically included: /usr/local/lib/python2.7/dist-packages/decapod_ansible/ceph-
↪ansible/roles/ceph.ceph-common/tasks/./checks/check_mandatory_vars.yml
statically included: /usr/local/lib/python2.7/dist-packages/decapod_ansible/ceph-
↪ansible/roles/ceph.ceph-common/tasks/./release.yml
statically included: /usr/local/lib/python2.7/dist-packages/decapod_ansible/ceph-
↪ansible/roles/ceph.ceph-common/tasks/facts.yml
statically included: /usr/local/lib/python2.7/dist-packages/decapod_ansible/ceph-
↪ansible/roles/ceph-mon/tasks/deploy_monitors.yml
statically included: /usr/local/lib/python2.7/dist-packages/decapod_ansible/ceph-
↪ansible/roles/ceph-mon/tasks/start_monitor.yml
statically included: /usr/local/lib/python2.7/dist-packages/decapod_ansible/ceph-
↪ansible/roles/ceph-mon/tasks/ceph_keys.yml
statically included: /usr/local/lib/python2.7/dist-packages/decapod_ansible/ceph-
↪ansible/roles/ceph-mon/tasks/openstack_config.yml
```

```
statically included: /usr/local/lib/python2.7/dist-packages/decapod_ansible/ceph-
↪ansible/roles/ceph-mon/tasks/create_mds_filesystems.yml
statically included: /usr/local/lib/python2.7/dist-packages/decapod_ansible/ceph-
↪ansible/roles/ceph-mon/tasks/secure_cluster.yml
statically included: /usr/local/lib/python2.7/dist-packages/decapod_ansible/ceph-
↪ansible/roles/ceph-mon/tasks/./docker/main.yml
statically included: /usr/local/lib/python2.7/dist-packages/decapod_ansible/ceph-
↪ansible/roles/ceph-mon/tasks/docker/checks.yml
statically included: /usr/local/lib/python2.7/dist-packages/decapod_ansible/ceph-
↪ansible/roles/ceph-mon/tasks/docker/pre_requisite.yml
statically included: /usr/local/lib/python2.7/dist-packages/decapod_ansible/ceph-
↪ansible/roles/ceph-mon/tasks/docker/dirs_permissions.yml
statically included: /usr/local/lib/python2.7/dist-packages/decapod_ansible/ceph-
↪ansible/roles/ceph-mon/tasks/docker/create_configs.yml
statically included: /usr/local/lib/python2.7/dist-packages/decapod_ansible/ceph-
↪ansible/roles/ceph-mon/tasks/docker/fetch_configs.yml
statically included: /usr/local/lib/python2.7/dist-packages/decapod_ansible/ceph-
↪ansible/roles/ceph-mon/tasks/docker/selinux.yml
statically included: /usr/local/lib/python2.7/dist-packages/decapod_ansible/ceph-
↪ansible/roles/ceph-mon/tasks/docker/start_docker_monitor.yml
statically included: /usr/local/lib/python2.7/dist-packages/decapod_ansible/ceph-
↪ansible/roles/ceph-mon/tasks/docker/copy_configs.yml
statically included: /usr/local/lib/python2.7/dist-packages/decapod_ansible/ceph-
↪ansible/roles/ceph-mon/tasks/calamari.yml
statically included: /usr/local/lib/python2.7/dist-packages/decapod_ansible/ceph-
↪ansible/roles/ceph-agent/tasks/pre_requisite.yml
statically included: /usr/local/lib/python2.7/dist-packages/decapod_ansible/ceph-
↪ansible/roles/ceph-agent/tasks/start_agent.yml
statically included: /usr/local/lib/python2.7/dist-packages/decapod_ansible/ceph-
↪ansible/roles/ceph.ceph-common/tasks/./checks/check_system.yml
statically included: /usr/local/lib/python2.7/dist-packages/decapod_ansible/ceph-
↪ansible/roles/ceph.ceph-common/tasks/./checks/check_mandatory_vars.yml
statically included: /usr/local/lib/python2.7/dist-packages/decapod_ansible/ceph-
↪ansible/roles/ceph.ceph-common/tasks/./release.yml
statically included: /usr/local/lib/python2.7/dist-packages/decapod_ansible/ceph-
↪ansible/roles/ceph.ceph-common/tasks/facts.yml
statically included: /usr/local/lib/python2.7/dist-packages/decapod_ansible/ceph-
↪ansible/roles/ceph.ceph-common/tasks/./checks/check_system.yml

...

TASK [ceph-restapi : run the ceph rest api docker image] ***********************
task path: /usr/local/lib/python2.7/dist-packages/decapod_ansible/ceph-ansible/
↪roles/ceph-restapi/tasks/docker/start_docker_restapi.yml:2
skipping: [10.10.0.8] => {"changed": false, "skip_reason": "Conditional check␣
↪failed", "skipped": true}

PLAY [rbdmirrors] **************************************************************
skipping: no hosts matched

PLAY [clients] *****************************************************************
skipping: no hosts matched

PLAY [iscsigws] ****************************************************************
skipping: no hosts matched

PLAY RECAP *********************************************************************
10.10.0.10                 : ok=61   changed=12   unreachable=0    failed=0
```

```
10.10.0.11                 : ok=60   changed=12   unreachable=0    failed=0
10.10.0.12                 : ok=60   changed=12   unreachable=0    failed=0
10.10.0.8                  : ok=90   changed=19   unreachable=0    failed=0
10.10.0.9                  : ok=60   changed=12   unreachable=0    failed=0
```

CHAPTER 7

Deploy an operating system on a Ceph node

Warning: Decapod does not perform bare metal provisioning, OS deployment, and network setup. Perform these operations by external means.

The OS must support `cloud-init`. Also, it must be possible to run your own user data. For the available datasources for `cloud-init`, see Datasources. Alternatively, you can set user data using the kernel command line. For bare metal provisioning, try MAAS. This section covers the MAAS installation and OS deployment with this tool.

The section contains the following topics:

Generate user data for cloud-init

This section contains the following topics:

Prerequisites

Prior to generating the user data for `cloud-init`, complete the following steps:

1. Verify that your Decapod installation is up and running.
2. Obtain the server discovery token. Decapod uses automatic server discovery and `cloud-init` is required only for that. To access the Decapod API, servers will access it using an authentication token with limited capabilities (posting to the server discovery API endpoint). The server discovery token is set in the `api.server_discovery_token` section of the `config.yaml` file. Keep this token private. To obtain the token:

```
$ grep server_discovery_token config.yaml
  server_discovery_token: "7f080dab-d803-4339-9c69-e647f7d6e200"
```

3. Generate an SSH public key. To generate the SSH public key from a private one, run:

```
$ ssh-keygen -y -f ansible_ssh_keyfile.pem > ansible_ssh_keyfile.pem.pub
```

Note: The `ansible_ssh_keyfile.pem` file should have the `0600` permissions:

```
$ chmod 0600 ansible_ssh_keyfile.pem
```

Generate user data

Verify that you have completed the steps described in *Prerequisites*.

To generate user data:

Run the following command:

```
$ decapod -u http://10.10.0.2:9999 cloud-config \
  7f080dab-d803-4339-9c69-e647f7d6e200 ansible_ssh_keyfile.pem.pub
```

Where the URL is the public URL of the Decapod machine with a correct port. The servers will send an HTTP request for server discovery using this URL. As a result, you will obtain a YAML-like user data.

MAAS requires user-data config to be BASE64 encoded. You can do that with:

```
$ decapod -u http://10.10.0.2:9999 cloud-config \
      7f080dab-d803-4339-9c69-e647f7d6e200 ansible_ssh_keyfile.pem.pub \
  | base64 -w 0
```

`-w` in **`base64`** means line wrapping. We need to disable it.

Deploy OS using MAAS

MAAS deployment is not part of this product. Therefore, we cannot guarantee its robustness. To provision your Ceph nodes manually, skip this section.

The section contains the following topics:

Prerequisites

MAAS installation has the following requirements:

- MAAS has its own DHCP server. To avoid collisions, disable the default one.
- If you plan to run MAAS in a virtual network with libvirt, create a new network with disabled DHCP, but enabled NAT.

Install MAAS

To install MAAS:

To install MAAS, follow the steps described in:

1. Installing a single node MAAS.

2. Importing the boot images.
3. Logging in.

Deploy an OS using MAAS

To deploy an operating system using MAAS:

1. Encode the user data to `base64` and send it to MAAS:

```
$ decapod -u http://10.10.0.2:9999 cloud-config \
  7f080dab-d803-4339-9c69-e647f7d6e200 ansible_ssh_keyfile.pem.pub \
  | base64 -w 0 > user_data.txt
```

2. Deploy an OS using the required MAAS version.

Note: MAAS 2.0 has non-backward-compatible API changes.

- MAAS 2.0:
  (a) Obtain `system_id` of the machine to deploy:

```
$ maas mymaas nodes read
```

  (b) Deploy the OS:

```
$ maas mymaas machine deploy {system_id} user_data={base64-encoded of␣
↪user-data}
```

  Where `mymaas` is the profile name of the MAAS command line.

- MAAS prior to 2.0:
  (a) Obtain `system_id` of the machine to deploy:

```
$ maas prof nodes list
```

  (b) Deploy the OS:

```
$ maas mymaas node start {system_id} user_data={base64-encoded of user-
↪data} distro_series={distro series. Eg. trusty}
```

  Where `mymaas` is the profile name of the MAAS command line.

CHAPTER 8

Supported Ceph packages

Mirantis provides its own Ceph packages with a set of patches that are not included in the community yet but are crucial for customers and internal needs. The supported LTS release of Ceph is Jewel. And the only supported distribution is 16.04 Xenial Xerus.

Mirantis keeps the patches as minimal and non-intrusive as possible and tracks the community releases as close as reasonable. To publish an urgent fix, intermediate releases can be issued. The packages are available from the following APT repository

```
deb http://mirror.fuel-infra.org/decapod/ceph/jewel-xenial jewel-xenial main
```

The following table lists packages provided for upgrades only:

Ceph release	Ubuntu release	APT repository
Jewel	14.04	`deb http://mirror.fuel-infra.org/decapod/ceph/jewel-trusty jewel-trusty main`
Hammer (0.94.x)	14.04	`deb http://mirror.fuel-infra.org/decapod/ceph/hammer-trusty hammer-trusty main`
Hammer (0.94.x)	12.04	`deb http://mirror.fuel-infra.org/decapod/ceph/hammer-precise hammer-precise main`
Firefly	14.04	`deb http://mirror.fuel-infra.org/decapod/ceph/firefly-trusty firefly-trusty main`
Firefly	12.04	`deb http://mirror.fuel-infra.org/decapod/ceph/ firefly-precise firefly-precise main`

Important: Packages for old LTS releases and Jewel for Ubuntu 14.04 are intended for upgrade purposes only and are *not* maintained other than fixing bugs hindering the upgrade to Jewel and Ubuntu 16.04.

CHAPTER 9

Playbook plugins

Decapod performs Ceph management through plugins. These plugins support different tasks, such as cluster deployment, adding and removing of OSDs, and so on. This section describes the available playbook plugins and the main options these plugins support.

The section contains the following topics:

Deploy Ceph cluster

The *Deploy Ceph cluster* playbook plugin allows you to deploy an initial Ceph cluster. The plugin supports all the capabilities and roles of `ceph-ansible`.

Note: The majority of configuration options described in this section match the `ceph-ansible` settings. For a list of supported parameters, see official list.

The section contains the following topics:

Overview

The following table shows the general information about the *Deploy Ceph cluster* plugin:

Property	Value
ID	cluster_deploy
Name	Deploy Ceph Cluster
Required Server List	Yes

The *Deploy Ceph cluster* plugin is tightly coupled with `ceph-ansible` versions. The following table shows the mapping between the plugin version and the corresponding version of `ceph-ansible`.

Plugin version	`ceph-ansible` version
>=0.1,<0.2	v1.0.8

Parameters and roles

The *Deploy Ceph cluster* plugin has the following parameters:

ceph_facts_template The path to custom Ceph facts template. Decapod deploys the custom facts module on the nodes that collect the Ceph-related facts. Usually, you do not need to configure this parameter.

ceph_stable Set to `true` if it is required to install Ceph from the stable repository.

ceph_stable_repo, ceph_stable_release, ceph_stable_distro_source The options define the repository where to obtain Ceph. In case of Ubuntu Xenial, you will get the following repository string:

```
deb {{ ceph_stable_repo }} {{ ceph_stable_distro_source }} main
```

cluster Defines the cluster name.

Important: Some tools require the `ceph` cluster name only. The default name allows executing the **ceph** utility without an explicit cluster name and with the `--cluster` option.

---

cluster_network Defines the cluster network.

copy_admin_key Copies the admin key to all nodes. This is required if you want to run the **ceph** utility from any cluster node. Keep this option as `true`. Otherwise, it may break some playbooks that maintain the lifecycle after deployment.

fsid The unique identifier for your object store. Since you can run multiple clusters on the same hardware, you must specify the unique ID of the object store when bootstrapping a monitor.

journal_collocation Defines if the OSD will place its journal on the same disk with the data. It is `false` by default.

If you want to have separate disks for journals (SSDs) and data (rotationals), set this to `false`. Also, set `raw_multi_journal` to `true` and list journal disks as `raw_journal_devices`.

raw_multi_journal This option is the opposite to `journal_collocation`.

Note: The `raw_multi_journal` and `journal_collocation` options must have different values. For example, if `journal_collocation` is set to `true`, set `raw_multi_journal` to `false`.

---

dmcrypt_journal_collocation This option has the same meaning as `journal_collocation` but both journal and data disks are encrypted by `dmcrypt`.

dmcrypt_dedicated_journal This option has the same meaning as `journal_collocation` set to `false`. If `dmcrypt_dedicated_journal` is set to `true`, the journal and data will be placed on different disks and encrypted with `dmcrypt`.

journal_size OSD journal size in megabytes.

max_open_files Sets the number of open files to have on a node.

nfs_file_gw Set to `true` to enable file access through NFS. Requires an MDS role.

nfs_obj_gw Set to `true` to enable object access through NFS. Requires an RGW role.

os_tuning_params Different kernels parameters. This is the list of dicts where `name` is the name of the parameter and `value` is the value.

public_network Defines the public network.

monitor_interface The option defines the NIC on the host that is connected to the public network.

devices Defines the disks where to place the OSD data. If collocation is enabled, then journal devices, `raw_journal_devices`, are not used.

raw_journal_devices Defines the disks where to place the journals for OSDs. If collocation is enabled, this option is not used.

`ceph-ansible` supports two deployment modes: with journal collocation and on separate drives, and also with `dmcrypt` and without. Therefore, there are four possible combinations.

The following table lists the possible combinations:

Collocation	Dmcrypt	journal_collocation	raw_multi_journal	dmcrypt_journal_collocation	dmcrypt_dedicated_journal	Data devices option name	Journal devices option name
true	true	false	true	false	false	devices	–
true	false	true	false	false	false	devices	–
false	true	false	false	false	true	devices	raw_journal_devices
false	false	false	true	false	false	devices	raw_journal_devices

Consider the different meaning of `devices` and `raw_journal_devices` in different modes: if no collocation is defined, then `devices` means disks with data. Journals are placed on `raw_journal_devices` disks. Otherwise, define `devices` only. In this case, the journal will be placed on the same device as the data.

Configuration example

The following is an example of the *Deploy Ceph cluster* plugin configuration:

```
{
  "global_vars": {
    "ceph_facts_template": "/usr/local/lib/python3.5/dist-packages/decapod_common/
↪facts/ceph_facts_module.py.j2",
    "ceph_stable": true,
    "ceph_stable_distro_source": "jewel-xenial",
    "ceph_stable_release": "jewel",
    "ceph_stable_repo": "http://eu.mirror.fuel-infra.org/shrimp/ceph/apt",
    "cluster": "ceph",
    "cluster_network": "10.10.0.0/24",
    "copy_admin_key": true,
    "dmcrypt_dedicated_journal": true,
    "dmcrypt_journal_collocation": false,
    "fsid": "e0b82a0d-b669-4787-8f4d-84f6733e45cd",
    "journal_collocation": false,
    "journal_size": 512,
    "max_open_files": 131072,
    "nfs_file_gw": false,
    "nfs_obj_gw": false,
    "os_tuning_params": [
      {
        "name": "kernel.pid_max",
        "value": 4194303
      },
      {
        "name": "fs.file-max",
        "value": 26234859
      }
    ],
    "public_network": "10.10.0.0/24",
```

```
    "raw_multi_journal": false
  },
  "inventory": {
    "_meta": {
      "hostvars": {
        "10.10.0.10": {
          "ansible_user": "ansible",
          "devices": [
            "/dev/vde",
            "/dev/vdb"
          ],
          "monitor_interface": "ens3",
          "raw_journal_devices": [
            "/dev/vdd",
            "/dev/vdc"
          ]
        },
        "10.10.0.11": {
          "ansible_user": "ansible",
          "devices": [
            "/dev/vde",
            "/dev/vdb"
          ],
          "monitor_interface": "ens3",
          "raw_journal_devices": [
            "/dev/vdd",
            "/dev/vdc"
          ]
        },
        "10.10.0.12": {
          "ansible_user": "ansible",
          "devices": [
            "/dev/vde",
            "/dev/vdb"
          ],
          "monitor_interface": "ens3",
          "raw_journal_devices": [
            "/dev/vdd",
            "/dev/vdc"
          ]
        },
        "10.10.0.8": {
          "ansible_user": "ansible",
          "devices": [
            "/dev/vde",
            "/dev/vdb"
          ],
          "monitor_interface": "ens3",
          "raw_journal_devices": [
            "/dev/vdd",
            "/dev/vdc"
          ]
        },
        "10.10.0.9": {
          "ansible_user": "ansible",
          "devices": [
            "/dev/vde",
            "/dev/vdb"
```

```
          ],
          "monitor_interface": "ens3",
          "raw_journal_devices": [
            "/dev/vdd",
            "/dev/vdc"
          ]
        }
      }
    },
    "clients": [],
    "iscsi_gw": [],
    "mdss": [],
    "mons": [
      "10.10.0.9"
    ],
    "nfss": [],
    "osds": [
      "10.10.0.10",
      "10.10.0.12",
      "10.10.0.11",
      "10.10.0.8"
    ],
    "rbd_mirrors": [],
    "restapis": [
      "10.10.0.9"
    ],
    "rgws": []
  }
}
```


Add OSD host

The *Add OSD host* playbook plugin allows you to add a new host with OSDs to a cluster. The plugin supports all the capabilities and roles of `ceph-ansible`.

Note: The majority of configuration options described in this section match the `ceph-ansible` settings. For a list of supported parameters, see official list.

The section contains the following topics:

Overview

The following table shows the general information about the *Add OSD host* plugin:

Property	Value
ID	add_osd
Name	Add OSD Host
Required Server List	Yes

The *Add OSD host* plugin is tightly coupled with `ceph-ansible` versions. The following table shows the mapping between the plugin version and the corresponding version of `ceph-ansible`.

Plugin version	`ceph-ansible` version
>=0.1,<0.2	v1.0.8

Parameters and roles

The *Add OSD host* plugin parameters are mostly the same as the ones for the *Deploy Ceph cluster* plugin. However, the plugin has the following roles:

`mons` Defines the nodes to deploy monitors.

`osds` Defines the nodes to deploy OSDs.

Configuration example

The following is an example of the *Add OSD host* plugin configuration:

```
{
    "data": {
        "cluster_id": "1597a71f-6619-4db6-9cda-a153f4f19097",
        "configuration": {
            "global_vars": {
                "ceph_facts_template": "/usr/local/lib/python3.5/dist-packages/shrimp_
↪common/facts/ceph_facts_module.py.j2",
                "ceph_stable": true,
                "ceph_stable_distro_source": "jewel-xenial",
                "ceph_stable_release": "jewel",
                "ceph_stable_repo": "http://eu.mirror.fuel-infra.org/shrimp/ceph/apt",
                "cluster": "ceph",
                "cluster_network": "10.10.0.0/24",
                "copy_admin_key": true,
                "fsid": "1597a71f-6619-4db6-9cda-a153f4f19097",
                "journal_collocation": true,
                "journal_size": 100,
                "max_open_files": 131072,
                "nfs_file_gw": false,
                "nfs_obj_gw": false,
                "os_tuning_params": [
                    {
                        "name": "kernel.pid_max",
                        "value": 4194303
                    },
                    {
                        "name": "fs.file-max",
                        "value": 26234859
                    }
                ],
                "public_network": "10.10.0.0/24"
            },
            "inventory": {
                "_meta": {
                    "hostvars": {
                        "10.10.0.2": {
                            "ansible_user": "ansible",
                            "devices": [
                                "/dev/vdb"
                            ],
                            "monitor_interface": "ens3"
```

```
                        },
                        "10.10.0.3": {
                            "ansible_user": "ansible",
                            "devices": [
                                "/dev/vdb"
                            ],
                            "monitor_interface": "ens3"
                        }
                    }
                },
                "mons": [
                    "10.10.0.2"
                ],
                "osds": [
                    "10.10.0.3",
                ],
            }
        },
        "name": "add_osd_name",
        "playbook_id": "add_osd"
    },
    "id": "fd76cea9-3efa-4432-854c-fee30ca79ddb",
    "initiator_id": "9d010f3f-2ec0-4079-ae8c-f46415e2530c",
    "model": "playbook_configuration",
    "time_deleted": 0,
    "time_updated": 1478174220,
    "version": 2
}
```


Remove OSD host

The *Remove OSD host* playbook plugin allows you to remove a host with OSDs from a cluster.

The section contains the following topics:

Overview

The following table shows the general information about the *Remove OSD host* plugin:

Property	Value
ID	remove_osd
Name	Remove OSD Host
Required Server List	Yes

Configuration example

The following is an example of the *Remove OSD host* plugin configuration:


```
{
  "global_vars": {
    "cluster": "ceph"
  },
  "inventory": {
```

```
    "_meta": {
      "hostvars": {
        "10.10.0.12": {
          "ansible_user": "ansible"
        },
        "10.10.0.9": {
          "ansible_user": "ansible"
        }
      }
    },
    "mons": [
      "10.10.0.9"
    ],
    "osds": [
      "10.10.0.12"
    ]
  }
}
```

This playbook has the simplest possible configuration. You only need to define the monitors and the OSD to remove.

Purge cluster

The *Purge cluster* playbook plugin allows you to remove a host with OSDs from a cluster.

The section contains the following topics:

Overview

The following table shows the general information about the *Purge cluster* plugin:

Property	Value
ID	purge_cluster
Name	Purge Cluster
Required Server List	No

Parameters and roles

The *Purge cluster* plugin has the following parameter:

`cluster` Defines the name of the cluster.

Important: Some tools require the `ceph` cluster name only. The default name allows executing the **`ceph`** utility without an explicit cluster name and with the `--cluster` option.

Configuration example

The following is an example of the *Purge cluster* plugin configuration:

```
{
  "global_vars": {
    "cluster": "ceph"
  },
  "inventory": {
    "_meta": {
      "hostvars": {
        "10.10.0.10": {
          "ansible_user": "ansible"
        },
        "10.10.0.11": {
          "ansible_user": "ansible"
        },
        "10.10.0.12": {
          "ansible_user": "ansible"
        },
        "10.10.0.8": {
          "ansible_user": "ansible"
        },
        "10.10.0.9": {
          "ansible_user": "ansible"
        }
      }
    },
    "mons": [
      "10.10.0.9"
    ],
    "osds": [
      "10.10.0.10",
      "10.10.0.12",
      "10.10.0.11",
      "10.10.0.8"
    ],
    "restapis": [
      "10.10.0.9"
    ]
  }
}
```

This playbook has the simplest possible configuration. You only need to define the nodes and their roles.

Index

D