Decapod Documentation

Release 1.2.0.dev1

Sergey Arkhipov

Jul 11, 2017

Contents

Decapod is a tool that simplifies the deployment and lifecycle management of Ceph clusters. This section guides you through the process of installation and configuration of Decapod.

CHAPTER 1

Deployment Guide

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. Install `git` and `make`.
2. Install Docker Engine as described in Install Docker Engine. Pay attention to the DNS configuration.
3. Install Docker Compose version 1.6 or later as described in Install Docker Compose.
4. Verify that your machine has access to the external network.

Install Decapod

The installation procedure contains the following steps:

1. Building the Decapod images.
2. Moving the Docker images to the target node.
3. Configuring Docker Compose.
4. Running the Docker containers.
5. Running migrations. If you run Decapod for the first time or upgrade from a previous version, apply migrations. This operation is idempotent and you may execute it safely at any time. Decapod does not reapply already applied migrations. On the first boot, migrations are required to obtain the root user. Otherwise, Decapod starts with an empty database and, therefore, without the capability to perform any operations.

Before you install Decapod, verify that you have completed the tasks described in *Prerequisites*.

To install Decapod:

1. Clone the source code repository:

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

2. In the repository, check the available versions using Git `tag`. To select a specific version:

```
git checkout {tag} && git submodule update --init --recursive
```

3. Build the Decapod images.

   (a) Copy the repository files to the top level directory:

   ```
   make copy_example_keys
   ```

   Note: The `copy_example_keys` target allows the build process to override the default Ubuntu and Debian repositories.

   (b) Build the images:

   ```
   $ make build_containers
   ```

Important: If you have no access to the private repository to fetch base images, use the community repository:

```
$ make docker_registry_use_dockerhub
```

To switch back:

```
$ make docker_registry_use_internal_ci
```

4. Move the Docker images to the target node.

Note: In this release, only one machine with Docker and Docker compose is supported. There may be one build machine and another production one. If you have such a diversity, use the Docker registry to manage Decapod images or dump/load them manually.

Use the following commands to dump Docker images, copy to a remote host, and load them:

```
$ 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
```

5. Configure Docker Compose as described in *Configure Docker Compose* and *Configuration files* subsection in the *Manage Ceph clusters using Decapod* section of *MCP Operations Guide*.

6. Run Docker Compose:

```
$ docker-compose up
```

To daemonize the process:

```
$ docker-compose up -d
```

To stop the detached process:

```
$ docker-compose down
```

For details, see Overview of the Docker Compose CLI.

7. If you run Decapod for the first time or upgrade from a previous version, run migrations.

   Example

```
$ docker-compose exec admin decapod-admin migration apply
2017-02-06 11:11:48 [DEBUG   ] (        lock.py:118 ): Lock applying_migrations␣
↪was acquire by locker 76eef103-0878-42c2-9727-b9e83e52db47
2017-02-06 11:11:48 [DEBUG   ] (        lock.py:183 ): Prolong thread for locker␣
↪applying_migrations of lock 76eef103-0878-42c2-9727-b9e83e52db47 has been␣
↪started. Thread MongoLock prolonger 76eef103-0878-42c2-9727-b9e83e52db47 for␣
↪applying_migrations, ident 140167584413440
2017-02-06 11:11:48 [INFO    ] (   migration.py:123 ): Run migration 0000_index_
↪models.py
2017-02-06 11:11:48 [INFO    ] (   migration.py:198 ): Run /usr/local/lib/python3.
↪5/dist-packages/decapod_admin/migration_scripts/0000_index_models.py. Pid 49
2017-02-06 11:11:53 [DEBUG   ] (        lock.py:164 ): Lock applying_migrations␣
↪was proloned by locker 76eef103-0878-42c2-9727-b9e83e52db47.
2017-02-06 11:11:56 [INFO    ] (   migration.py:203 ): /usr/local/lib/python3.5/
↪dist-packages/decapod_admin/migration_scripts/0000_index_models.py has been␣
↪finished. Exit code 0
2017-02-06 11:11:56 [INFO    ] (   migration.py:277 ): Save result of 0000_index_
↪models.py migration (result MigrationState.ok)
2017-02-06 11:11:56 [INFO    ] (   migration.py:123 ): Run migration 0001_insert_
↪default_role.py
2017-02-06 11:11:56 [INFO    ] (   migration.py:198 ): Run /usr/local/lib/python3.
↪5/dist-packages/decapod_admin/migration_scripts/0001_insert_default_role.py.␣
↪Pid 56
2017-02-06 11:11:58 [INFO    ] (   migration.py:203 ): /usr/local/lib/python3.5/
↪dist-packages/decapod_admin/migration_scripts/0001_insert_default_role.py has␣
↪been finished. Exit code 0
2017-02-06 11:11:58 [INFO    ] (   migration.py:277 ): Save result of 0001_insert_
↪default_role.py migration (result MigrationState.ok)
2017-02-06 11:11:58 [INFO    ] (   migration.py:123 ): Run migration 0002_insert_
↪default_user.py
2017-02-06 11:11:58 [INFO    ] (   migration.py:198 ): Run /usr/local/lib/python3.
↪5/dist-packages/decapod_admin/migration_scripts/0002_insert_default_user.py.␣
↪Pid 64
2017-02-06 11:11:58 [DEBUG   ] (        lock.py:164 ): Lock applying_migrations␣
↪was proloned by locker 76eef103-0878-42c2-9727-b9e83e52db47.
2017-02-06 11:11:59 [INFO    ] (   migration.py:203 ): /usr/local/lib/python3.5/
↪dist-packages/decapod_admin/migration_scripts/0002_insert_default_user.py has␣
↪been finished. Exit code 0
2017-02-06 11:11:59 [INFO    ] (   migration.py:277 ): Save result of 0002_insert_
↪default_user.py migration (result MigrationState.ok)
2017-02-06 11:11:59 [INFO    ] (   migration.py:123 ): Run migration 0003_native_
↪ttl_index.py
2017-02-06 11:11:59 [INFO    ] (   migration.py:198 ): Run /usr/local/lib/python3.
↪5/dist-packages/decapod_admin/migration_scripts/0003_native_ttl_index.py. Pid␣
↪192
```

```
2017-02-06 11:12:00 [INFO    ] (   migration.py:203 ): /usr/local/lib/python3.5/
↪dist-packages/decapod_admin/migration_scripts/0003_native_ttl_index.py has been␣
↪finished. Exit code 0
2017-02-06 11:12:00 [INFO    ] (   migration.py:277 ): Save result of 0003_native_
↪ttl_index.py migration (result MigrationState.ok)
2017-02-06 11:12:00 [INFO    ] (   migration.py:123 ): Run migration 0004_migrate_
↪to_native_ttls.py
2017-02-06 11:12:00 [INFO    ] (   migration.py:198 ): Run /usr/local/lib/python3.
↪5/dist-packages/decapod_admin/migration_scripts/0004_migrate_to_native_ttls.py.␣
↪Pid 200
2017-02-06 11:12:02 [INFO    ] (   migration.py:203 ): /usr/local/lib/python3.5/
↪dist-packages/decapod_admin/migration_scripts/0004_migrate_to_native_ttls.py␣
↪has been finished. Exit code 0
2017-02-06 11:12:02 [INFO    ] (   migration.py:277 ): Save result of 0004_
↪migrate_to_native_ttls.py migration (result MigrationState.ok)
2017-02-06 11:12:02 [INFO    ] (   migration.py:123 ): Run migration 0005_index_
↪cluster_data.py
2017-02-06 11:12:02 [INFO    ] (   migration.py:198 ): Run /usr/local/lib/python3.
↪5/dist-packages/decapod_admin/migration_scripts/0005_index_cluster_data.py. Pid␣
↪208
2017-02-06 11:12:03 [INFO    ] (   migration.py:203 ): /usr/local/lib/python3.5/
↪dist-packages/decapod_admin/migration_scripts/0005_index_cluster_data.py has␣
↪been finished. Exit code 0
2017-02-06 11:12:03 [INFO    ] (   migration.py:277 ): Save result of 0005_index_
↪cluster_data.py migration (result MigrationState.ok)
2017-02-06 11:12:03 [INFO    ] (   migration.py:123 ): Run migration 0006_create_
↪cluster_data.py
2017-02-06 11:12:03 [INFO    ] (   migration.py:198 ): Run /usr/local/lib/python3.
↪5/dist-packages/decapod_admin/migration_scripts/0006_create_cluster_data.py.␣
↪Pid 216
2017-02-06 11:12:03 [DEBUG   ] (        lock.py:164 ): Lock applying_migrations␣
↪was proloned by locker 76eef103-0878-42c2-9727-b9e83e52db47.
2017-02-06 11:12:04 [INFO    ] (   migration.py:203 ): /usr/local/lib/python3.5/
↪dist-packages/decapod_admin/migration_scripts/0006_create_cluster_data.py has␣
↪been finished. Exit code 0
2017-02-06 11:12:04 [INFO    ] (   migration.py:277 ): Save result of 0006_create_
↪cluster_data.py migration (result MigrationState.ok)
2017-02-06 11:12:04 [INFO    ] (   migration.py:123 ): Run migration 0007_add_
↪external_id_to_user.py
2017-02-06 11:12:04 [INFO    ] (   migration.py:198 ): Run /usr/local/lib/python3.
↪5/dist-packages/decapod_admin/migration_scripts/0007_add_external_id_to_user.py.
↪ Pid 224
2017-02-06 11:12:06 [INFO    ] (   migration.py:203 ): /usr/local/lib/python3.5/
↪dist-packages/decapod_admin/migration_scripts/0007_add_external_id_to_user.py␣
↪has been finished. Exit code 0
2017-02-06 11:12:06 [INFO    ] (   migration.py:277 ): Save result of 0007_add_
↪external_id_to_user.py migration (result MigrationState.ok)
2017-02-06 11:12:06 [DEBUG   ] (        lock.py:202 ): Prolong thread for locker␣
↪applying_migrations of lock 76eef103-0878-42c2-9727-b9e83e52db47 has been␣
↪stopped. Thread MongoLock prolonger 76eef103-0878-42c2-9727-b9e83e52db47 for␣
↪applying_migrations, ident 140167584413440
2017-02-06 11:12:06 [DEBUG   ] (        lock.py:124 ): Try to release lock␣
↪applying_migrations by locker 76eef103-0878-42c2-9727-b9e83e52db47.
2017-02-06 11:12:06 [DEBUG   ] (        lock.py:140 ): Lock applying_migrations␣
↪was released by locker 76eef103-0878-42c2-9727-b9e83e52db47.
```

For a list of applied migrations, use the `list all` option.

Example

```
$ docker-compose exec admin decapod-admin migration list all
[applied]     0000_index_models.py
[applied]     0001_insert_default_role.py
[applied]     0002_insert_default_user.py
[applied]     0003_native_ttl_index.py
[applied]     0004_migrate_to_native_ttls.py
[applied]     0005_index_cluster_data.py
[applied]     0006_create_cluster_data.py
[applied]     0007_add_external_id_to_user.py
```

For details on a certain migration, use the `show` option.

Example

```
$ docker-compose exec admin decapod-admin migration show 0006_create_cluster_data.
↪py
Name:           0006_create_cluster_data.py
Result:         ok
Executed at:    Mon Feb  6 11:12:04 2017
SHA1 of script: 73eb7adeb1b4d82dd8f9bdb5aadddccbcef4a8b3

-- Stdout:
Migrate 0 clusters.

-- Stderr:
```

8. Reset the `root` user password.

   (a) Obtain the user ID:

   ```
   $ docker-compose exec admin decapod user get-all
   ```

   Example:

   ```
   $ docker-compose exec admin decapod user get-all
   [
      {
          "data": {
              "email": "noreply@example.com",
              "full_name": "Root User",
              "login": "root",
              "role_id": "4ca555d3-24fd-4554-9b4b-ca44bac45062"
          },
          "id": "e6f28a01-ee7f-4ac8-b1ee-a1a21c3eb182",
          "initiator_id": null,
          "model": "user",
          "time_deleted": 0,
          "time_updated": 1488279856,
          "version": 1
      }
   ]
   ```

   (a) Change the password:

   ```
   $ decapod-admin password-reset USER_ID
   ```

Install decapod client libraries

Decapod has 2 client libraries: 1 is *decapodlib* which is API client implemented in RPC fashion (comparable to client libraries for other web services). This library is supported by Python2 (>= 2.7 release) and Python3 (>= 3.3 release). Please find details on that library in API reference.

Another library is called *decapodcli* and it is providing command line interface for Decapod installation. *decapodcli* uses *decapodlib* internally. As *decapodlib*, *decapodcli* also supports the same Pythons: Python2 (>= 2.7) and Python3 (>= 3.3).

As for release 1.1, these libraries are not hosted on PyPI therefore you have to install them from sources.

Install *decapodlib*

1. Install prerequisites:

```
$ apt-get install --no-install-recommends git gcc libssl-dev libyaml-dev python␣
↪python-dev python-pip python-setuptools python-wheel
```

   If you would like to use Python3, then install these prerequisites:

```
$ apt-get install --no-install-recommends git gcc libssl-dev libyaml-dev python3␣
↪python3-dev python3-pip python3-setuptools python3-wheel
```

2. Copy source code:

```
$ git clone -b 1.1.0 --depth 1 https://github.com/Mirantis/ceph-lcm.git ~/decapod
Cloning into '/home/vagrant/decapod'...
remote: Counting objects: 1051, done.
remote: Compressing objects: 100% (822/822), done.
remote: Total 1051 (delta 312), reused 486 (delta 188), pack-reused 0
Receiving objects: 100% (1051/1051), 1.15 MiB | 660.00 KiB/s, done.
Resolving deltas: 100% (312/312), done.
Checking connectivity... done.
```

3. Install *decapodlib*:

```
$ pip2 install ~/decapod/decapodlib
```

   Or if you are using Python3:

```
$ pip3 install ~/decapod/decapodlib
```

4. Verify that library is installed:

```
$ python2 -c 'import decapodlib; print "OK"'
OK
```

   Or with Python3:

```
$ python3 -c 'import decapodlib; print("OK")'
OK
```

Install *decapodcli*

1. Install *decapodlib* as described in *Install decapodlib*.
2. Install CLI:

```
$ pip2 install ~/decapod/decapodlib ~/decapod/decapodcli
```

Or with Python3

```
$ pip3 install ~/decapod/decapodlib ~/decapod/decapodcli
```

Note:

Sometimes you may face with following bug:

```
Traceback (most recent call last):
  File "/usr/lib/python2.7/dist-packages/pip/req/req_install.py", line␣
↪377, in setup_py
    import setuptools  # noqa
  File "/usr/share/python-wheels/setuptools-20.7.0-py2.py3-none-any.whl/
↪setuptools/__init__.py", line 11, in <module>
  File "/usr/share/python-wheels/setuptools-20.7.0-py2.py3-none-any.whl/
↪setuptools/extern/__init__.py", line 1, in <module>
ImportError: No module named extern
```

It basically means that package *cryptography* has installed latest *setuptools*. In that case just remove it and try again.

```
$ pip2 uninstall setuptools
```

It is safe to do because pip won't touch system *setuptools*.

---

3. Verify that CLI is set up:

```
$ decapod
Usage: decapod [OPTIONS] COMMAND [ARGS]...

Decapod command line tool.

With this CLI it is possible to access all API endpoints of Decapod. To do
so, you have to provide some common configuration settings: URL, login and
password to access.

These settings are possible to setup using commandline parameter, but if
you want, you can set environment variables:

    - DECAPOD_URL             - this environment variable sets URL to
                                access.
    - DECAPOD_LOGIN           - this environment variable sets login.
    - DECAPOD_PASSWORD        - this environment variable sets password.
    - DECAPOD_TIMEOUT         - this environment variable sets timeout.
    - DECAPOD_NO_VERIFY       - this environment variable removes SSL
                                certificate verification.
    - DECAPOD_SSL_CERTIFICATE - this environment variable sets a path
                                to SSL client certificate.
    - DECAPOD_DEBUG           - this environment variable sets debug mode.
```

```
    - DECAPOD_NO_PAGER        - (deprecated) this environment variable
                                removes pager support.
    - DECAPOD_PAGER           - this environment variable add pager
                                support.

Options:
  -u, --url TEXT                  Base URL for Decapod.  [required]
  -l, --login TEXT                Login to access Decapod.
  -p, --password TEXT             Password to access Decapod.
  -t, --timeout INTEGER           Timeout to access API. No timeout by
                                  default.
  -k, --no-verify                 Do not verify SSL certificates.
  -s, --ssl-certificate FILENAME
  -d, --debug                     Run in debug mode.
  -n, --no-pager                  Do not use pager for output.
  -r, --pager                     Use pager for output.
  -f, --output-format [json]      How to format output. Currently only JSON is
                                  supported.  [default: json]
  --version                       Show the version and exit.
  -h, --help                      Show this message and exit.

Commands:
  cloud-config            Generates config for cloud-init.
  cluster                 Cluster subcommands.
  execution               Execution subcommands.
  info                    Request information about remove Decapod...
  password-reset          Password reset subcommands
  permission              Permission subcommands.
  playbook                Playbook subcommands.
  playbook-configuration  Playbook configuration subcommands.
  role                    Role subcommands.
  server                  Server subcommands.
  user                    User subcommands.
```

Configure Docker Compose

To configure Docker Compose, modify the `docker-compose.override.yml` file or set the environment variables. Use the official Docker documentation and the information below.

The Decapod Docker Compose configuration supports a number of environment variables. For a list of variables, see the .env file at the top level of the repository. The defaults are applicable for a development environment built on a local machine and have to be modified to run in production:

Environment variable	Default value	Description
DECAPOD_HTTP_PORT	9999	The port to bind the HTTP endpoint of Decapod.
DECAPOD_HTTPS_PORT	10000	The port to bind the HTTPS endpoint of Decapod.
DECAPOD_MONITORING_PORT	11000	The port to bind the endpoint to monitoring data for Decapod.
DECAPOD_REGISTRY_URL		By default, Decapod tries to access local images. To take images from a private registry, point it here.
DECAPOD_NAMESPACE		In private registries, Decapod images are not always created without a prefix, sometimes the organization name, like `mirantis`, is present. The variable sets this prefix.
DECAPOD_VERSION	latest	The Decapod version to use. This is the image tag that is set in the registry. The `latest` tag means developer version.
DECAPOD_SSH_PRIVATE_KEY	$(pwd)/containerization/files/devconfigs/ansible_ssh_keyfile.pem	A full path to the SSH private key that Ansible uses to access Ceph nodes.

Default configuration example:

```
networks: {}
services:
  api:
    image: decapod/api:latest
    links:
    - database
    restart: on-failure:5
  controller:
    image: decapod/controller:latest
    links:
    - database
    restart: on-failure:5
    volumes:
    - /vagrant/containerization/files/devconfigs/ansible_ssh_keyfile.pem:/root/.ssh/
↪id_rsa:ro
   cron:
     image: decapod/cron:latest
     links:
     - database
     restart: on-failure:3
   database:
     image: decapod/db:latest
     restart: always
     volumes_from:
     - service:database_data:rw
   database_data:
     image: decapod/db-data:latest
     volumes:
     - /data/db:rw
   frontend:
     image: decapod/frontend:latest
     links:
     - api
     - cron
     ports:
     - 10000:443
```

```
    - 9999:80
    restart: always
version: '2.0'
volumes: {}
```

For example, to set `docker-prod-virtual.docker.mirantis.net` as a registry and `mirantis/ceph` as a namespace and run version 0.2, execute **`docker compose`** with the following environment variables:

```
$ DECAPOD_REGISTRY_URL=docker-prod-virtual.docker.mirantis.net/ \
DECAPOD_NAMESPACE=mirantis/ceph/ DECAPOD_VERSION=0.2 docker-compose config
networks: {}
services:
  api:
    image: docker-prod-virtual.docker.mirantis.net/mirantis/ceph/decapod/api:0.2
    links:
    - database
    restart: on-failure:5
  controller:
    image: docker-prod-virtual.docker.mirantis.net/mirantis/ceph/decapod/controller:0.
↪2
    links:
    - database
    restart: on-failure:5
    volumes:
    - /vagrant/containerization/files/devconfigs/ansible_ssh_keyfile.pem:/root/.ssh/
↪id_rsa:ro
  cron:
    image: docker-prod-virtual.docker.mirantis.net/mirantis/ceph/decapod/cron:0.2
    links:
    - database
    restart: on-failure:3
  database:
    image: docker-prod-virtual.docker.mirantis.net/mirantis/ceph/decapod/db:0.2
    restart: always
    volumes_from:
    - service:database_data:rw
  database_data:
    image: docker-prod-virtual.docker.mirantis.net/mirantis/ceph/decapod/db-data:0.2
    volumes:
    - /data/db:rw
  frontend:
    image: docker-prod-virtual.docker.mirantis.net/mirantis/ceph/decapod/frontend:0.2
    links:
    - api
    - cron
    ports:
    - 10000:443
    - 9999:80
    restart: always
version: '2.0'
volumes: {}
```

Important: The trailing slash in `DECAPOD_REGISTRY_URL` and `DECAPOD_NAMESPACE` is required due to the limitations of the Docker Compose configuration file.

Note: Docker Compose supports reading the environment variables from the `.env` file, which should be placed in the same directory as the `docker-compose.yml` file. For more information, see the Docker documentation.

Example:

Configuration:

- The default Mirantis registry for Decapod and the latest version of Decapod
- The private SSH key for Ansible is placed in `/keys/ansible_ssh_keyfile.pem`
- The Decapod HTTP port is 80 and the HTTP port is 443

The `.env` file should look as follows:

```
DECAPOD_NAMESPACE=mirantis/ceph/
DECAPOD_REGISTRY_URL=docker-prod-virtual.docker.mirantis.net/
DECAPOD_VERSION=latest
DOCKER_HTTP_PORT=80
DOCKER_HTTPS_PORT=443
DOCKER_SSH_PRIVATE_KEY=/keys/ansible_ssh_keyfile.pem
```

Alternatively, set the environment variables explicitly:

```
$ export DECAPOD_NAMESPACE=mirantis/ceph/
$ export DECAPOD_REGISTRY_URL=docker-prod-virtual.docker.mirantis.net/
$ export DECAPOD_VERSION=latest
$ export DOCKER_HTTP_PORT=80
$ export DOCKER_HTTPS_PORT=443
$ export DOCKER_SSH_PRIVATE_KEY=/keys/ansible_ssh_keyfile.pem
$ docker-compose config
networks: {}
services:
  api:
    image: docker-prod-virtual.docker.mirantis.net/mirantis/ceph/decapod/api:latest
    links:
    - database
    restart: on-failure:5
  controller:
    image: docker-prod-virtual.docker.mirantis.net/mirantis/ceph/decapod/
↪controller:latest
    links:
    - database
    restart: on-failure:5
    volumes:
    - /keys/ansible_ssh_keyfile.pem:/root/.ssh/id_rsa:ro
  cron:
    image: docker-prod-virtual.docker.mirantis.net/mirantis/ceph/decapod/cron:latest
    links:
    - database
    restart: on-failure:3
  database:
    image: docker-prod-virtual.docker.mirantis.net/mirantis/ceph/decapod/db:latest
    restart: always
    volumes_from:
    - service:database_data:rw
  database_data:
    image: docker-prod-virtual.docker.mirantis.net/mirantis/ceph/decapod/db-
↪data:latest
```

```
    volumes:
    - /data/db:rw
  frontend:
    image: docker-prod-virtual.docker.mirantis.net/mirantis/ceph/decapod/
↪frontend:latest
    links:
    - api
    - cron
    ports:
    - 443:443
    - 80:80
    restart: always
version: '2.0'
volumes: {}
```

See also:

Configuration files in the *Manage Ceph clusters using Decapod* section of *MCP Operations Guide*

CHAPTER 2

Operational Guide

Configuration files

Decapod supports a number of configuration files you may want to propagate into a container. This section describes these files.

SSH private key

The `ansible_ssh_keyfile.pem` file is an SSH private key used by Ansible to connect to Ceph nodes. 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 do not have a prepared SSH private key, generate a new one as described in Create SSH keys.

After you generate the key, copy it to the top level of the source code repository. The file name must be `ansible_ssh_keyfile.pem` and the format of the file must be PEM.

> **Warning:** Keep the key private.

SSL certificates

The following files are the SSL certificates:

- `ssl.key` - Private key for SSL/TLS certificate. Used by web UI.
- `ssl.crt` - Signed certificate for SSL/TLS. Used by web UI.
- `ssl-dhparam.pem` - Diffie-Hellman ephemeral parameters for SSL/TLS. This enables perfect forward secrecy for secured connection.

If you do not have such certificates, generate new ones as described in OpenSSL Essentials and Forward Secrecy & Diffie Hellman Ephemeral Parameters. All SSL keys should be in the PEM format. Place the SSL files to the top level of your source code repository.

Warning: Keep the key private. Do not use self-signed certificates for a production installation.

Decapod configuration file

Decapod configuration is performed within the `config.yaml` file in YAML format. Decapod searches for the configuration file in several locations in the following order:

- `$(pwd/decapod.yaml`
- `$XDG_CONFIG_HOME/decapod/config.yaml`
- `:`$HOME/.decapod.yaml`
- `/etc/decapod/config.yaml`
- Default configuration file of the `decapod_common` package

If a configuration file was found and parsed before, other alternatives will not be used. Therefore, if you have the default configuration file in `/etc/decapod/config.yaml` then placing the configuration to `$XDG_CONFIG_HOME/decapod/config.yaml` will override the default one. For details, see XDG Base Directory Specification.

Default configuration in containerized Decapod stack is placed in `/etc/decapod/config.yaml`.

Decapod config.yaml example

The following is an example of the default Decapod configuration file for containers:

```
---
common:
  password:
    length: 10
    time_cost: 10
    memory_cost: 2048
    parallelism: 3
    hash_len: 32
    salt_len: 16
  password_reset_ttl_in_seconds: 86400  # 1 day
  email:
    enabled: false
    from: "noreply@mirantis.com"
    host: "localhost"
    port: 25
    login: ""
    password: ""

# Options here are Flask options so please check
# http://flask.pocoo.org/docs/0.11/config/#builtin-configuration-values
api:
  debug: false
  testing: false
  logger_name: "decapod.decapod_api.wsgi"
  logger_handler_policy: "never"
  json_sort_keys: faluse
  jsonify_prettyprint_regular: false
  json_as_ascii: false
  pagination_per_page: 25
```

```
  server_discovery_token: "26758c32-3421-4f3d-9603-e4b5337e7ecc"
  reset_password_url: "http://127.0.0.1/password_reset/{reset_token}/"
  token:
    ttl_in_seconds: 1800
  logging:
    propagate: true
    level: "DEBUG"
    handlers:
      - "stderr_debug"
  auth:
    type: native
    parameters: {}
    # type: keystone
    # parameters:
    #   auth_url: http://keystone:5000/v3
    #   username: admin
    #   password: nomoresecret
    #   project_domain_name: default
    #   project_name: admin
    #   user_domain_name: default

controller:
  pidfile: "/tmp/decapod-controller.pid"
  daemon: false
  ansible_config: "/etc/ansible/ansible.cfg"
  # 0 worker_threads means that we will have 2 * CPU count threads
  worker_threads: 0
  graceful_stop: 10
  logging:
    propagate: true
    level: "DEBUG"
    handlers:
      - "stderr_debug"

cron:
  clean_finished_tasks_after_seconds: 2592000  # 60 * 60 * 24 * 30; 30 days

db:
  uri: "mongodb://database:27017/decapod?ssl=true"
  connect: false
  connect_timeout: 5000  # ms, 5 seconds
  socket_timeout: 5000  # ms, 5 seconds
  pool_size: 50
  gridfs_chunk_size_in_bytes: 261120  # 255 kilobytes

plugins:
  alerts:
    enabled: []
    email:
      enabled: false
      send_to:
        - "bigboss@example.com"
      from: "errors@example.com"
  playbooks:
    disabled:
      - hello_world

# Default Python logging is used.
```

```
# https://docs.python.org/2/library/logging.config.html#dictionary-schema-details
logging:
  version: 1
  incremental: false
  disable_existing_loggers: true
  root:
    handlers: []
  filters: {}
  formatters:
    stderr_default:
      format: "%(asctime)s [%(levelname)-8s]: %(message)s"
      datefmt: "%Y-%m-%d %H:%M:%S"
    stderr_debug:
      format: "%(asctime)s [%(levelname)-8s] (%(filename)15s:%(lineno)-4d):
↪%(message)s"
      datefmt: "%Y-%m-%d %H:%M:%S"
    syslog:
      format: "%(name)s %(asctime)s [%(levelname)-8s]: %(message)s"
      datefmt: "%Y-%m-%d %H:%M:%S"
  handlers:
    stderr_debug:
      class: "logging.StreamHandler"
      formatter: "stderr_debug"
      level: "DEBUG"
    stderr_default:
      class: "logging.StreamHandler"
      formatter: "stderr_default"
      level: "DEBUG"
    syslog:
      class: "logging.handlers.SysLogHandler"
      formatter: "syslog"
      level: "DEBUG"
```

See also:

Decapod config.yaml settings description

Settings description

The following tables describe the `config.yaml` configuration file settings:

Table 2.1: Common settings

Setting	Description
`common`	Defines generic Decapod settings not related to API or controller. You can specify the following parameters: **`password`** Describes settings for Decapod key derivation function. Decapod does not store user passwords in plain text. Instead, it uses key derivation functions to calculate a cryptographic secure hash from the password. To do so, it uses the Argon2 key derivation function that is similar to the scrypt key derivation function but has a property for defense against concurrent attacks with GPUs. To change the default settings, follow the Argon2 documentation. **`password_reset_ttl_in_seconds`** Sets the TTL value in seconds for the password reset token. When resetting the password, the user gets a secret token. Consuming this token performs the actual password reset. This parameter sets the TTL of such token. The token is valid only for the specified amount of time and expires after. **`email`** Defines how to send emails from Decapod. The `from` parameter defines the email to set in the `From` field. The `enabled` parameter (boolean) enables or disables the email sending. If disabled, all other fields in this section are ignored.

The `api` section contains settings specific to the API service only. Some parameters propagate directly to Flask. For Flask settings description, see Flask documentation. The following parameters are related to Flask:

- `DEBUG`
- `TESTING`
- `LOGGER_NAME`
- `LOGGER_HANDLER_POLICY`
- `JSON_SORT_KEYS`
- `JSON_AS_ASCII`
- `JSONIFY_PRETTYPRINT_REGULAR`

If you are not sure which parameter to specify, use the default ones.

The following parameters of the `api` section are Decapod-related:

Table 2.2: Decapod-related API settings

Setting	Description
`pagination_per_page`	Sets a default count of items per page in paginated listings. If the number of items is less than `pagination_per_page`, then fewer elements would be returned.
`server_discovery_token`	Defines the server discovery token. Servers found during the server discovery must have an authentication token to access the `POST /v1/server` API endpoint. This token does not refer to any certain user and allows accessing only the mentioned API endpoint. However, Ansible will access the remote host to gather facts and verify the access.
`reset_password_url`	Defines the template of the URL that will be used for generating the email during the password reset. The email sent to the user will contain this URL. Decapod will replace `{reset_token}` to a correct password reset token.
`token`	Contains configuration for authentication tokens. The `ttl_in_seconds` is the TTL value in seconds. This parameter applies only to newly generated tokens. This section is used only if native authentication back end is enabled. For example, Keystone integration will not use this parameter because Keystone manages its own tokens.
`logging`	Defines specific parameters for logging in API. Applies the parameters specified in the `logging` setting to API only.
`auth`	Configures the authentication back end used by Decapod. If not specified, the native authentication back end with default configuration is used. The `type` parameter defines the type of the back end to use and `parameters` define the back-end configuration. for details on available authentication back ends, see *Authentication back ends*.

Table 2.3: Controller settings

Setting	Description
`controller`	Defines specific settings for the controller service. This service manages the task queue and runs Ansible for tasks. You can specify the following parameters: **`daemon`** Defines whether to run the controller service as a UNIX daemon. If you use `systemd` or Docker containers, set this to `false`. **`pidfile`** Defines the PIDFile for the daemon if the controller service is run as a daemon. **`ansible_config`** Defines the path to the default Ansible configuration to use. You can leave this parameter as is. **`worker_threads`** Defines the number of workers per controller. The controller service uses the worker pool to manage Ansible executions concurrently. The `0` value means to define this number automatically. By default, it is `2 * cpu_count`. **`graceful_stop`** Defines the graceful stop for external processes in seconds. Since the controller service executes a number of processes, it cannot be stopped immediately, the processes should be correctly finished. Initially, the controller service sends the `SIGTERM` signal to the processes and if they do not stop after the amount of time specified in `graceful_stop`, the controller service stops them with `SIGKILL`. **`logging`** Defines specific parameters for logging in the controller service. Applies the parameters specified in the `logging` setting to the controller service only.

Table 2.4: Cron settings

Setting	Description
`cron`	Defines Cron-related settings. You can specify the following parameters: **`clean_finished_tasks_after_seconds`** Defines the TTL for finished tasks. After the specified amount of time, the tasks will be purged from the database. This is related only to finished tasks that were completed or failed and is not related to not started tasks.

Table 2.5: Database settings

Setting	Description
`db`	Defines the MongoDB-related settings, for example, how to connect to the database and some specifics of the database client configuration. You can specify the following parameters: **`uri`** Defines the URI to connect to MongoDB. For information on connection URIs, see the MongoDB documentation. **`connect`** Defines whether Decapod will connect to MongoDB immediately after initialization of a client or on the first request. We suggest that you keep this parameter value `false`. **`socket_timeout`** Defines the amount of time in milliseconds the driver will wait during server monitoring when connecting a new socket to a server before concluding that the server is unavailable. **`socket_timeout`** Defines the amount of time in milliseconds the driver will wait for a response after sending an ordinary (non-monitoring) database operation before concluding that a network error has occurred. **`pool_size`** Defines the maximum allowed number of concurrent connections to each connected server. Requests to a server will be blocked if there are more connections to the requested served than defined in `pool_size`. **`gridfs_chunk_size_in_bytes`** Defines the size of file chunk (a part of the file, stored in a separate document) for GridFS. It is 255 kilobytes by default.

Table 2.6: Plugins and logs settings

Setting	Description
`plugins`	Describes what to do with plugins: disable, enable, and others. All plugins are split into 2 categories, `alerts` and `plugins`. • The `alerts` section contains a list of enabled alerts plugins responsible for issues alerting, for example, in case of a `500` error. Every parameter except `enabled` defines how to set up each alert plugin. • The `playbooks` section has only 1 parameter: `disabled` that lists the plugins that are disabled even if installed.
`logging`	Defines the configuration of Decapod logging. For more information on this setting and its options, see Python documentation.

See also:

Decapod configuration file example

Authentication back ends

Decapod supports two authentication backends: the default native and the Keystone authentication back ends.

Native authentication back end

Native authentication back end uses Decapod MongoDB to store authentication tokens. Each time a user logs in to Decapod, it creates a new authentication token and stores it in the collection. Each time a user logs out, Decapod removes the corresponding token. Also, every token has a TTL value and when it expires, MongoDB deletes the token. This is performed using the MongoDB TTL indexes.

To set up the native authentication back end, place the following snippet to the `api` section of the `config.yaml` file:

```
auth:
  type: native
  parameters: {}
```

This type of back end does not require configuration.

Keystone authentication back end

Keystone authentication back end uses Keystone for authentication. This is option has a more complex setup than the default *Native authentication back end*.

Using the Keystone authentication back end, creating or deleting a user in Decapod will not affect Keystone and Decapod will not create or remove a user from Keystone. Decapod synchronizes the user list with Keystone every 10 minutes. So if you create, delete, or disable a user in Keystone, it will be also created, deleted, or disabled in Decapod.

To set up Keystone integration:

1. Place the following snippet to the `api` section of the `config.yaml` file:

   ```
   auth:
     type: keystone
     parameters:
       auth_url: {os_auth_url}
       username: {os_username}
       password: {os_password}
       project_domain_name: {os_project_domain_name}
       project_name: {os_project_name}
       user_domain_name: {os_domain_name}
   ```

   For details on these parameters, see the OpenStack command-line options. For the whole list of options, see v3.Password.

   Important: Username and password should correspond to the user that has enough permissions to request tokens for other users and list them.

2. Perform initial synchronization using the `admin` service:

```
$ docker-compose -p myprojectname exec admin decapod-admin keystone initial -h
Usage: decapod-admin keystone initial [OPTIONS] ROLE [USER]...

  Initial Keystone sync.

  On initial sync it is possible to setup role for a user (users). If no
  usernames are given, then all users from Keystone would be synced and role
  will be applied to them.

Options:
  -h, --help  Show this message and exit.

Specify the role name (default is ``wheel``, which has the biggest number of
permissions) and user login for this role.
```

As a result, you should be able to access Decapod and set required roles for users.

Note: Newly synchronized users from Keystone have no role.

Using the `admin` service, synchronization is performed by Cron, but you can execute it manually after the initial synchronization:

```
$ docker-compose -p myprojectname exec admin decapod-admin keystone sync
```

See also:

Decapod configuration file

MongoDB certificate and key

The `mongodb.pem` file is the SSL/TLS pair of certificate and key, concatenated in one file. This is required for a secure connection to MongoDB. Generate this file as described in MongoDB documentation. To allow SSL/TLS on client side, verify that the configuration file has the `?ssl=true` parameter in URI. For example, `mongodb://database:27017/db` will not use a secure connection, but `mongodb://database:27017/db?ssl=true` will.

Note: To use database authentication, see:

- MongoDB documentation
- MongoDB security
- Docker hub

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

By default, containers contain no information about users and their passwords.

Configuration files location

The table below provides the list of configuration files and their location in containers depending on the particular Docker Compose service. After changing the configuration, place the changed file into an appropriate container.

Table 2.7: Configuration files location

Configuration file	Location
`ansible_ssh_keyfile.pem`	• Controller: `/root/.ssh/id_rsa` • Admin: `/root/.ssh/id_rsa`
`ssl.key`	• Front end: `/ssl/ssl.key`
`ssl.crt`	• Front end: `/ssl/ssl.crt`
`ssl-dhparam.pem`	• Front end: `/ssl/dhparam.pem`
`config.yaml`	• API: `/etc/decapod/config.yaml` • Controller: `/etc/decapod/config.yaml` • Admin: `/etc/decapod/config.yaml`
`mongodb.pem`	• Database: `/certs/mongodb.pem`
`mongodb-ca.crt`	• Database: `/certs/mongodb-ca.crt`

To specify custom files, use the `docker-compose.override.yml` file. For details, see Docker Compose documentation. An example of the `docker-compose.override.yml` file is placed in the top level of the repository.

Note: Provide the modified configuration for API, controller, and Cron services. There is no possibility to define it for all services in Docker Compose configuration version 2.

See also:

- PEM
- YAML

Deploy an OS on a Ceph node

Warning: Decapod does not perform bare metal provisioning, OS deployment, and network setup.

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.

Generate user data for cloud-init

Prerequisites

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

1. Verify that your Decapod installation is up and running:

   ```
   $ docker-compose -p PROJECT ps
   ```

   All containers except `decapod_database_data_1` should be in the `Up` state.

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.

Deploy OS using MAAS

Decapod does not provide MAAS deployment. This section describes one of the Ceph node OS deployment options that you may consider. To provision your Ceph nodes manually, skip this section.

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.

Note: If you do not want or cannot use server discovery, refer to Ansible playbooks to prepare a machine based on generated user data.

Manage users and roles

This section descibes how to manage users and roles in Decapod through the web UI.

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

Using the Decapod web UI you can create, edit, and delete roles as well as assign a role to a user.

To 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*.

To 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.

To 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.

To 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:

Ceph cluster deployed by Decapod in *MCP Reference Architecture*

Deploy a cluster

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

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*.

Playbook plugins

Decapod uses plugins to manage Ceph. These plugins support various tasks, such as cluster deployment, adding and removing of OSDs, and others. This section describes the available playbook plugins and the main options these plugins support.

Add RBD and CLI clients to the host

This plugin adds required packages and config files to the node to allow end user to execute **`ceph`** tool. Also, it installs **`rbd`** and **`rados`** tools and configures them.

Overview

The following table shows the general information about the *Add RBD and CLI clients to the host* plugin:

Property	Value
ID	`add_client`
Name	Add RBD and CLI clients to the host
Required Server List	Yes

This 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
>=1.1,<1.2	v2.2.4

Parameters and roles

For consistency, Decapod checks the Ceph version it is going to deploy. If a Ceph cluster has inconsistent versions, the deployment stops and you must fix the versions withing the cluster. If the Ceph version you are going to deploy is newer that the deployed ones, the process will also stop and you must update the cluster packages first.

The following parameters are responsble for such checks:

ceph_version_verify A boolean setting that checks that strict mode is enabled. If set to `false`, no verification described above is performed.

ceph_version_verify_packagename The name of the package to check. It is not required to configure this setting.

Configuration example

The following is an example of the *Add RBD and CLI clients to the host* plugin configuration:


```
{
  "global_vars": {
    "ceph_nfs_access_type": "RW",
    "ceph_nfs_ceph_access_type": "RW",
    "ceph_nfs_ceph_export_id": 20134,
    "ceph_nfs_ceph_protocols": "3,4",
    "ceph_nfs_ceph_pseudo_path": "/cephobject",
    "ceph_nfs_export_id": 20134,
    "ceph_nfs_log_file": "/var/log/ganesha.log",
    "ceph_nfs_protocols": "3,4",
    "ceph_nfs_pseudo_path": "/cephfile",
    "ceph_nfs_rgw_access_type": "RW",
    "ceph_nfs_rgw_export_id": 20134,
    "ceph_nfs_rgw_protocols": "3,4",
    "ceph_nfs_rgw_pseudo_path": "/ceph",
    "ceph_nfs_rgw_user": "cephnfs",
    "ceph_restapi_port": 5000,
    "ceph_stable": true,
    "ceph_stable_distro_source": "jewel-xenial",
    "ceph_stable_release": "jewel",
    "ceph_stable_release_uca": "jewel-xenial",
    "ceph_stable_repo": "http://mirror.fuel-infra.org/decapod/ceph/jewel-xenial",
    "ceph_stable_repo_key": "AF94F6A6A254F5F0",
    "ceph_stable_repo_keyserver": "hkp://keyserver.ubuntu.com:80",
    "ceph_version_verify": true,
    "ceph_version_verify_packagename": "ceph-common",
    "cluster": "ceph",
    "cluster_network": "10.0.0.0/24",
    "common_single_host_mode": true,
    "copy_admin_key": true,
    "dmcrypt_dedicated_journal": false,
    "dmcrypt_journal_collocation": false,
    "fsid": "0860ebc0-35af-465f-80e8-c394fc9af2de",
    "journal_collocation": false,
    "journal_size": 512,
    "max_open_files": 131072,
    "mds_allow_multimds": true,
    "mds_max_mds": 3,
    "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": "10.0.0.0/24",
    "radosgw_civetweb_num_threads": 50,
    "radosgw_civetweb_port": 8080,
    "radosgw_dns_s3website_name": "your.subdomain.tld",
    "radosgw_static_website": false,
    "radosgw_usage_log": false,
    "radosgw_usage_log_flush_threshold": 1024,
    "radosgw_usage_log_tick_interval": 30,
    "radosgw_usage_max_shards": 32,
    "radosgw_usage_max_user_shards": 1,
    "raw_multi_journal": true
  },
  "inventory": {
    "_meta": {
      "hostvars": {
        "10.0.0.20": {
          "ansible_user": "ansible",
          "devices": [
            "/dev/vdj",
            "/dev/vdb",
            "/dev/vdd",
            "/dev/vdh",
            "/dev/vdf"
          ],
          "monitor_address": "10.0.0.20",
          "raw_journal_devices": [
            "/dev/vdg",
            "/dev/vdi",
            "/dev/vde",
            "/dev/vdc",
            "/dev/vdk"
          ]
        },
        "10.0.0.21": {
          "ansible_user": "ansible",
          "devices": [
            "/dev/vdj",
            "/dev/vdb",
            "/dev/vdd",
            "/dev/vdh",
            "/dev/vdf"
          ],
          "monitor_address": "10.0.0.21",
          "raw_journal_devices": [
            "/dev/vdg",
            "/dev/vdi",
            "/dev/vde",
            "/dev/vdc",
```

```
            "/dev/vdk"
          ]
        },
        "10.0.0.22": {
          "ansible_user": "ansible",
          "devices": [
            "/dev/vdj",
            "/dev/vdb",
            "/dev/vdd",
            "/dev/vdh",
            "/dev/vdf"
          ],
          "monitor_address": "10.0.0.22",
          "raw_journal_devices": [
            "/dev/vdg",
            "/dev/vdi",
            "/dev/vde",
            "/dev/vdc",
            "/dev/vdk"
          ]
        },
        "10.0.0.23": {
          "ansible_user": "ansible"
        },
        "10.0.0.24": {
          "ansible_user": "ansible"
        }
      }
    },
    "already_deployed": [
      "10.0.0.21",
      "10.0.0.20",
      "10.0.0.22"
    ],
    "clients": [
      "10.0.0.24",
      "10.0.0.23"
    ],
    "mons": [
      "10.0.0.20"
    ]
  }
}
```


Add metadata server to the host

This plugin installs metadata server (MDS) to the host.

MDS stores metadata on behalf of the Ceph Filesystem (i.e., Ceph Block Devices and Ceph Object Storage do not use MDS). Ceph Metadata Servers make it feasible for POSIX file system users to execute basic commands like ls, find, etc. without placing an enormous burden on the Ceph Storage Cluster.

Overview

The following table shows the general information about the *Add metadata server to the host* plugin:

Property	Value
ID	`add_mds`
Name	Add metadata server to the host
Required Server List	Yes

This 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
>=1.1,<1.2	v2.1.9

Parameters and roles

For consistency, Decapod checks the Ceph version it is going to deploy. If a Ceph cluster has inconsistent versions, the deployment stops and you must fix the versions withing the cluster. If the Ceph version you are going to deploy is newer that the deployed ones, the process will also stop and you must update the cluster packages first.

The following parameters are responsble for such checks:

`ceph_version_verify` A boolean setting that checks that strict mode is enabled. If set to `false`, no verification described above is performed.

`ceph_version_verify_packagename` The name of the package to check. It is not required to configure this setting.

Configuration example

The following is an example of the *Add metadata server to the host* plugin configuration:

```
{
  "global_vars": {
    "ceph_nfs_access_type": "RW",
    "ceph_nfs_ceph_access_type": "RW",
    "ceph_nfs_ceph_export_id": 20134,
    "ceph_nfs_ceph_protocols": "3,4",
    "ceph_nfs_ceph_pseudo_path": "/cephobject",
    "ceph_nfs_export_id": 20134,
    "ceph_nfs_log_file": "/var/log/ganesha.log",
    "ceph_nfs_protocols": "3,4",
    "ceph_nfs_pseudo_path": "/cephfile",
    "ceph_nfs_rgw_access_type": "RW",
    "ceph_nfs_rgw_export_id": 20134,
    "ceph_nfs_rgw_protocols": "3,4",
    "ceph_nfs_rgw_pseudo_path": "/ceph",
    "ceph_nfs_rgw_user": "cephnfs",
    "ceph_restapi_port": 5000,
    "ceph_stable": true,
    "ceph_stable_distro_source": "jewel-xenial",
    "ceph_stable_release": "jewel",
    "ceph_stable_release_uca": "jewel-xenial",
    "ceph_stable_repo": "http://mirror.fuel-infra.org/decapod/ceph/jewel-xenial",
    "ceph_stable_repo_key": "AF94F6A6A254F5F0",
    "ceph_stable_repo_keyserver": "hkp://keyserver.ubuntu.com:80",
    "ceph_version_verify": true,
    "ceph_version_verify_packagename": "ceph-common",
    "cluster": "ceph",
    "cluster_network": "10.0.0.0/24",
```

```
    "common_single_host_mode": true,
    "copy_admin_key": true,
    "dmcrypt_dedicated_journal": false,
    "dmcrypt_journal_collocation": false,
    "fsid": "0860ebc0-35af-465f-80e8-c394fc9af2de",
    "journal_collocation": false,
    "journal_size": 512,
    "max_open_files": 131072,
    "mds_allow_multimds": true,
    "mds_max_mds": 3,
    "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": "10.0.0.0/24",
    "radosgw_civetweb_num_threads": 50,
    "radosgw_civetweb_port": 8080,
    "radosgw_dns_s3website_name": "your.subdomain.tld",
    "radosgw_static_website": false,
    "radosgw_usage_log": false,
    "radosgw_usage_log_flush_threshold": 1024,
    "radosgw_usage_log_tick_interval": 30,
    "radosgw_usage_max_shards": 32,
    "radosgw_usage_max_user_shards": 1,
    "raw_multi_journal": true
  },
  "inventory": {
    "_meta": {
      "hostvars": {
        "10.0.0.20": {
          "ansible_user": "ansible",
          "devices": [
            "/dev/vdj",
            "/dev/vdb",
            "/dev/vdd",
            "/dev/vdh",
            "/dev/vdf"
          ],
          "monitor_address": "10.0.0.20",
          "raw_journal_devices": [
            "/dev/vdg",
            "/dev/vdi",
            "/dev/vde",
            "/dev/vdc",
            "/dev/vdk"
          ]
        },
        "10.0.0.21": {
          "ansible_user": "ansible",
          "devices": [
```

```
            "/dev/vdj",
            "/dev/vdb",
            "/dev/vdd",
            "/dev/vdh",
            "/dev/vdf"
          ],
          "monitor_address": "10.0.0.21",
          "raw_journal_devices": [
            "/dev/vdg",
            "/dev/vdi",
            "/dev/vde",
            "/dev/vdc",
            "/dev/vdk"
          ]
        },
        "10.0.0.22": {
          "ansible_user": "ansible",
          "devices": [
            "/dev/vdj",
            "/dev/vdb",
            "/dev/vdd",
            "/dev/vdh",
            "/dev/vdf"
          ],
          "monitor_address": "10.0.0.22",
          "raw_journal_devices": [
            "/dev/vdg",
            "/dev/vdi",
            "/dev/vde",
            "/dev/vdc",
            "/dev/vdk"
          ]
        },
        "10.0.0.23": {
          "ansible_user": "ansible"
        },
        "10.0.0.24": {
          "ansible_user": "ansible"
        }
      }
    },
    "already_deployed": [
      "10.0.0.21",
      "10.0.0.20",
      "10.0.0.22"
    ],
    "mdss": [
      "10.0.0.24",
      "10.0.0.23"
    ],
    "mons": [
      "10.0.0.20"
    ]
  }
}
```

Add monitor host

The *Add monitor host* playbook plugin allows you to add a new host with monitors 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.

Overview

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

Property	Value
ID	`add_mon`
Name	Add Monitor Host
Required Server List	Yes

The *Add monitor 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
>=0.2,<0.3	v2.1.9
>=1.1,<1.2	v2.2.4

Parameters and roles

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

`mons` Defines the nodes to deploy monitors.

Note: For consistency, Decapod checks the Ceph version it is going to deploy. If a Ceph cluster has inconsistent versions, the deployment stops and you must fix the versions withing the cluster. If the Ceph version you are going to deploy is newer that the deployed ones, the process will also stop and you must update the cluster packages first.

The following parameters are responsble for such checks:

`ceph_version_verify` A boolean setting that checks that strict mode is enabled. If set to `false`, no verification described above is performed.

`ceph_version_verify_packagename` The name of the package to check. It is not required to configure this setting.

Configuration example

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

```
{
  "global_vars": {
    "ceph_nfs_access_type": "RW",
    "ceph_nfs_ceph_access_type": "RW",
    "ceph_nfs_ceph_export_id": 20134,
```

```
    "ceph_nfs_ceph_protocols": "3,4",
    "ceph_nfs_ceph_pseudo_path": "/cephobject",
    "ceph_nfs_export_id": 20134,
    "ceph_nfs_log_file": "/var/log/ganesha.log",
    "ceph_nfs_protocols": "3,4",
    "ceph_nfs_pseudo_path": "/cephfile",
    "ceph_nfs_rgw_access_type": "RW",
    "ceph_nfs_rgw_export_id": 20134,
    "ceph_nfs_rgw_protocols": "3,4",
    "ceph_nfs_rgw_pseudo_path": "/ceph",
    "ceph_nfs_rgw_user": "cephnfs",
    "ceph_restapi_port": 5000,
    "ceph_stable": true,
    "ceph_stable_distro_source": "jewel-xenial",
    "ceph_stable_release": "jewel",
    "ceph_stable_release_uca": "jewel-xenial",
    "ceph_stable_repo": "http://mirror.fuel-infra.org/decapod/ceph/jewel-xenial",
    "ceph_stable_repo_key": "AF94F6A6A254F5F0",
    "ceph_stable_repo_keyserver": "hkp://keyserver.ubuntu.com:80",
    "ceph_version_verify": true,
    "ceph_version_verify_packagename": "ceph-common",
    "cluster": "ceph",
    "cluster_network": "10.0.0.0/24",
    "common_single_host_mode": true,
    "copy_admin_key": true,
    "dmcrypt_dedicated_journal": false,
    "dmcrypt_journal_collocation": false,
    "fsid": "0860ebc0-35af-465f-80e8-c394fc9af2de",
    "journal_collocation": false,
    "journal_size": 512,
    "max_open_files": 131072,
    "mds_allow_multimds": true,
    "mds_max_mds": 3,
    "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": "10.0.0.0/24",
    "radosgw_civetweb_num_threads": 50,
    "radosgw_civetweb_port": 8080,
    "radosgw_dns_s3website_name": "your.subdomain.tld",
    "radosgw_static_website": false,
    "radosgw_usage_log": false,
    "radosgw_usage_log_flush_threshold": 1024,
    "radosgw_usage_log_tick_interval": 30,
    "radosgw_usage_max_shards": 32,
    "radosgw_usage_max_user_shards": 1,
    "raw_multi_journal": true
  },
  "inventory": {
```

```
    "_meta": {
      "hostvars": {
        "10.0.0.20": {
          "ansible_user": "ansible",
          "devices": [
            "/dev/vdj",
            "/dev/vdb",
            "/dev/vdd",
            "/dev/vdh",
            "/dev/vdf"
          ],
          "monitor_address": "10.0.0.20",
          "raw_journal_devices": [
            "/dev/vdg",
            "/dev/vdi",
            "/dev/vde",
            "/dev/vdc",
            "/dev/vdk"
          ]
        },
        "10.0.0.21": {
          "ansible_user": "ansible",
          "devices": [
            "/dev/vdj",
            "/dev/vdb",
            "/dev/vdd",
            "/dev/vdh",
            "/dev/vdf"
          ],
          "monitor_address": "10.0.0.21",
          "raw_journal_devices": [
            "/dev/vdg",
            "/dev/vdi",
            "/dev/vde",
            "/dev/vdc",
            "/dev/vdk"
          ]
        },
        "10.0.0.22": {
          "ansible_user": "ansible",
          "devices": [
            "/dev/vdj",
            "/dev/vdb",
            "/dev/vdd",
            "/dev/vdh",
            "/dev/vdf"
          ],
          "monitor_address": "10.0.0.22",
          "raw_journal_devices": [
            "/dev/vdg",
            "/dev/vdi",
            "/dev/vde",
            "/dev/vdc",
            "/dev/vdk"
          ]
        },
        "10.0.0.23": {
          "ansible_user": "ansible",
```

```
          "monitor_address": "10.0.0.23"
        },
        "10.0.0.24": {
          "ansible_user": "ansible",
          "monitor_address": "10.0.0.24"
        }
      }
    },
    "already_deployed": [
      "10.0.0.21",
      "10.0.0.20",
      "10.0.0.22"
    ],
    "mons": [
      "10.0.0.20",
      "10.0.0.23",
      "10.0.0.24"
    ],
    "oldmons": [
      "10.0.0.20"
    ]
  }
}
```


Add NFS Gateway to the host

This plugin installs NFS gateway to the host as a companion to Rados Gateway. This brings the ability to store data as object through the REST interface of Rados Gateway and retrieve them as file on a network filesystem, presently NFS.

Please check official blog post for the details.

Overview

The following table shows the general information about the *Add NFS Gateway* plugin:

Property	Value
ID	`add_nfs`
Name	Add NDS Gateway to the host
Required Server List	Yes

Important: This plugin requires node to have a role *rgws*. If you do not have it yet, you can deploy Rados Gateway implicitly using plugin hints.

This 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
>=1.1,<1.2	v2.2.4

Parameters and roles

`file_access` Defines is NFS is set up to give access to objects as files.

object_access Defines is NFS is set up to give access to objects themselves using NFS (providing RADOS abstractions, not only Rados Gateway ones).

do_not_deploy_rgw Defines if it is not required to deploy Rados Gateway to the node. Please remember that RGW is required. Use this option if RGW is already deployed and set up.

For consistency, Decapod checks the Ceph version it is going to deploy. If a Ceph cluster has inconsistent versions, the deployment stops and you must fix the versions withing the cluster. If the Ceph version you are going to deploy is newer that the deployed ones, the process will also stop and you must update the cluster packages first.

The following parameters are responsble for such checks:

ceph_version_verify A boolean setting that checks that strict mode is enabled. If set to `false`, no verification described above is performed.

ceph_version_verify_packagename The name of the package to check. It is not required to configure this setting.

Configuration example

The following is an example of the *Add NFS Gateway* plugin configuration:

```
{
  "global_vars": {
    "ceph_nfs_access_type": "RW",
    "ceph_nfs_ceph_access_type": "RW",
    "ceph_nfs_ceph_export_id": 20134,
    "ceph_nfs_ceph_protocols": "3,4",
    "ceph_nfs_ceph_pseudo_path": "/cephobject",
    "ceph_nfs_export_id": 20134,
    "ceph_nfs_log_file": "/var/log/ganesha.log",
    "ceph_nfs_protocols": "3,4",
    "ceph_nfs_pseudo_path": "/cephfile",
    "ceph_nfs_rgw_access_type": "RW",
    "ceph_nfs_rgw_export_id": 20134,
    "ceph_nfs_rgw_protocols": "3,4",
    "ceph_nfs_rgw_pseudo_path": "/ceph",
    "ceph_nfs_rgw_user": "cephnfs",
    "ceph_restapi_port": 5000,
    "ceph_stable": true,
    "ceph_stable_distro_source": "jewel-xenial",
    "ceph_stable_release": "jewel",
    "ceph_stable_release_uca": "jewel-xenial",
    "ceph_stable_repo": "http://mirror.fuel-infra.org/decapod/ceph/jewel-xenial",
    "ceph_stable_repo_key": "AF94F6A6A254F5F0",
    "ceph_stable_repo_keyserver": "hkp://keyserver.ubuntu.com:80",
    "ceph_version_verify": true,
    "ceph_version_verify_packagename": "ceph-common",
    "cluster": "ceph",
    "cluster_network": "10.0.0.0/24",
    "common_single_host_mode": true,
    "copy_admin_key": true,
    "dmcrypt_dedicated_journal": false,
    "dmcrypt_journal_collocation": false,
    "fsid": "0860ebc0-35af-465f-80e8-c394fc9af2de",
    "journal_collocation": false,
    "journal_size": 512,
    "max_open_files": 131072,
    "mds_allow_multimds": true,
```

```
    "mds_max_mds": 3,
    "nfs_file_gw": true,
    "nfs_obj_gw": true,
    "os_tuning_params": [
      {
        "name": "fs.file-max",
        "value": 26234859
      },
      {
        "name": "kernel.pid_max",
        "value": 4194303
      }
    ],
    "public_network": "10.0.0.0/24",
    "radosgw_civetweb_num_threads": 50,
    "radosgw_civetweb_port": 8080,
    "radosgw_dns_s3website_name": "your.subdomain.tld",
    "radosgw_static_website": false,
    "radosgw_usage_log": false,
    "radosgw_usage_log_flush_threshold": 1024,
    "radosgw_usage_log_tick_interval": 30,
    "radosgw_usage_max_shards": 32,
    "radosgw_usage_max_user_shards": 1,
    "raw_multi_journal": true
  },
  "inventory": {
    "_meta": {
      "hostvars": {
        "10.0.0.20": {
          "ansible_user": "ansible",
          "devices": [
            "/dev/vdj",
            "/dev/vdb",
            "/dev/vdd",
            "/dev/vdh",
            "/dev/vdf"
          ],
          "monitor_address": "10.0.0.20",
          "raw_journal_devices": [
            "/dev/vdg",
            "/dev/vdi",
            "/dev/vde",
            "/dev/vdc",
            "/dev/vdk"
          ]
        },
        "10.0.0.21": {
          "ansible_user": "ansible",
          "devices": [
            "/dev/vdj",
            "/dev/vdb",
            "/dev/vdd",
            "/dev/vdh",
            "/dev/vdf"
          ],
          "monitor_address": "10.0.0.21",
          "raw_journal_devices": [
            "/dev/vdg",
```

```
            "/dev/vdi",
            "/dev/vde",
            "/dev/vdc",
            "/dev/vdk"
          ]
        },
        "10.0.0.22": {
          "ansible_user": "ansible",
          "devices": [
            "/dev/vdj",
            "/dev/vdb",
            "/dev/vdd",
            "/dev/vdh",
            "/dev/vdf"
          ],
          "monitor_address": "10.0.0.22",
          "raw_journal_devices": [
            "/dev/vdg",
            "/dev/vdi",
            "/dev/vde",
            "/dev/vdc",
            "/dev/vdk"
          ]
        },
        "10.0.0.23": {
          "ansible_user": "ansible"
        },
        "10.0.0.24": {
          "ansible_user": "ansible"
        }
      }
    },
    "already_deployed": [
      "10.0.0.21",
      "10.0.0.20",
      "10.0.0.22"
    ],
    "mons": [
      "10.0.0.20"
    ],
    "nfss": [
      "10.0.0.24",
      "10.0.0.23"
    ],
    "rgws": [
      "10.0.0.24",
      "10.0.0.23"
    ]
  }
}
```


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.

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 following table lists the available hints for the plugin:

Hint	Title	Default value	Description
`dmcrypt`	Use dmcrypted OSDs	`False`	Defines the dmcrypt usage for OSD devices.
`collocation`	Collocate OSD data and journal on same devices	`False`	Defines whether the journal and data will be placed on the same devices.

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
>=1.1,<1.2	v2.2.4

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.

Note: For consistency, Decapod checks the Ceph version it is going to deploy. If a Ceph cluster has inconsistent versions, the deployment stops and you must fix the versions withing the cluster. If the Ceph version you are going to deploy is newer that the deployed ones, the process will also stop and you must update the cluster packages first.

The following parameters are responsble for such checks:

`ceph_version_verify` A boolean setting that checks that strict mode is enabled. If set to `false`, no verification described above is performed.

`ceph_version_verify_packagename` The name of the package to check. It is not required to configure this setting.

Configuration example

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

```
{
  "global_vars": {
    "ceph_nfs_access_type": "RW",
    "ceph_nfs_ceph_access_type": "RW",
    "ceph_nfs_ceph_export_id": 20134,
    "ceph_nfs_ceph_protocols": "3,4",
    "ceph_nfs_ceph_pseudo_path": "/cephobject",
    "ceph_nfs_export_id": 20134,
    "ceph_nfs_log_file": "/var/log/ganesha.log",
    "ceph_nfs_protocols": "3,4",
    "ceph_nfs_pseudo_path": "/cephfile",
    "ceph_nfs_rgw_access_type": "RW",
    "ceph_nfs_rgw_export_id": 20134,
    "ceph_nfs_rgw_protocols": "3,4",
    "ceph_nfs_rgw_pseudo_path": "/ceph",
    "ceph_nfs_rgw_user": "cephnfs",
    "ceph_restapi_port": 5000,
    "ceph_stable": true,
    "ceph_stable_distro_source": "jewel-xenial",
    "ceph_stable_release": "jewel",
    "ceph_stable_release_uca": "jewel-xenial",
    "ceph_stable_repo": "http://mirror.fuel-infra.org/decapod/ceph/jewel-xenial",
    "ceph_stable_repo_key": "AF94F6A6A254F5F0",
    "ceph_stable_repo_keyserver": "hkp://keyserver.ubuntu.com:80",
    "ceph_version_verify": true,
    "ceph_version_verify_packagename": "ceph-common",
    "cluster": "ceph",
    "cluster_network": "10.0.0.0/24",
    "common_single_host_mode": true,
    "copy_admin_key": true,
    "dmcrypt_dedicated_journal": false,
    "dmcrypt_journal_collocation": false,
    "fsid": "0860ebc0-35af-465f-80e8-c394fc9af2de",
    "journal_collocation": false,
    "journal_size": 512,
    "max_open_files": 131072,
    "mds_allow_multimds": true,
    "mds_max_mds": 3,
    "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": "10.0.0.0/24",
    "radosgw_civetweb_num_threads": 50,
    "radosgw_civetweb_port": 8080,
    "radosgw_dns_s3website_name": "your.subdomain.tld",
    "radosgw_static_website": false,
    "radosgw_usage_log": false,
    "radosgw_usage_log_flush_threshold": 1024,
    "radosgw_usage_log_tick_interval": 30,
```

```
    "radosgw_usage_max_shards": 32,
    "radosgw_usage_max_user_shards": 1,
    "raw_multi_journal": true
  },
  "inventory": {
    "_meta": {
      "hostvars": {
        "10.0.0.20": {
          "ansible_user": "ansible",
          "devices": [
            "/dev/vdb",
            "/dev/vdj",
            "/dev/vdd",
            "/dev/vdh",
            "/dev/vdf"
          ],
          "monitor_address": "10.0.0.20",
          "raw_journal_devices": [
            "/dev/vdg",
            "/dev/vdi",
            "/dev/vde",
            "/dev/vdc",
            "/dev/vdk"
          ]
        },
        "10.0.0.21": {
          "ansible_user": "ansible",
          "devices": [
            "/dev/vdb",
            "/dev/vdj",
            "/dev/vdd",
            "/dev/vdh",
            "/dev/vdf"
          ],
          "monitor_address": "10.0.0.21",
          "raw_journal_devices": [
            "/dev/vdg",
            "/dev/vdi",
            "/dev/vde",
            "/dev/vdc",
            "/dev/vdk"
          ]
        },
        "10.0.0.22": {
          "ansible_user": "ansible",
          "devices": [
            "/dev/vdb",
            "/dev/vdj",
            "/dev/vdd",
            "/dev/vdh",
            "/dev/vdf"
          ],
          "monitor_address": "10.0.0.22",
          "raw_journal_devices": [
            "/dev/vdg",
            "/dev/vdi",
            "/dev/vde",
            "/dev/vdc",
```

```
            "/dev/vdk"
          ]
        },
        "10.0.0.23": {
          "ansible_user": "ansible",
          "devices": [
            "/dev/vdj",
            "/dev/vdb",
            "/dev/vdd",
            "/dev/vdh",
            "/dev/vdf"
          ],
          "monitor_address": "10.0.0.23",
          "raw_journal_devices": [
            "/dev/vdg",
            "/dev/vdi",
            "/dev/vde",
            "/dev/vdc",
            "/dev/vdk"
          ]
        },
        "10.0.0.24": {
          "ansible_user": "ansible",
          "devices": [
            "/dev/vdj",
            "/dev/vdb",
            "/dev/vdd",
            "/dev/vdh",
            "/dev/vdf"
          ],
          "monitor_address": "10.0.0.24",
          "raw_journal_devices": [
            "/dev/vdg",
            "/dev/vdi",
            "/dev/vde",
            "/dev/vdc",
            "/dev/vdk"
          ]
        }
      }
    },
    "already_deployed": [
      "10.0.0.21",
      "10.0.0.20",
      "10.0.0.22"
    ],
    "mons": [
      "10.0.0.20"
    ],
    "osds": [
      "10.0.0.24",
      "10.0.0.23"
    ]
  }
}
```

Add Ceph Pool

The *Add Ceph pool* playbook plugin allows you to setup new pool (or modify existing one) in Ceph.

Overview

The following table shows the general information about the *Add Ceph pool* plugin:

Property	Value
ID	`add_pool`
Name	Add Ceph pool
Required server list	No

The following table lists the available hints for the plugin:

Hint	Title	Default value	Description
`pool_name`	The name of the pool to add or modify. Must be unique.	`test`	Defines the name of the pool you are going to create/modify.
`pg_num`	The total number of placement groups for the pool	`0`	The number of placement groups for the pool. 0 means default setting
`pgp_num`	The total number of placement groups for the placement purposes	`0`	The number of placement groups for the pool placement purposes. 0 means default setting
`pool_type`	The type of the pool.	`replicated`	Defines type of the pool. Please check official documentation for details.
`crush_ruleset_name`	The name of a CRUSH ruleset to use for this pool. The specified ruleset must exist.	`replicated_ruleset`	For replicated pools it is the ruleset specified by the `osd pool default crush replicated ruleset` config variable. This ruleset must exist. For erasure pools it is erasure-code if the default erasure code profile is used or {pool-name} otherwise. This ruleset will be created implicitly if it doesn't exist already.
`erasure_code_profile`	The name of a erasure code profile.	`default`	The name of the erasure code profile. Please check ' documentation <http://docs.ceph.com/docs/kraken/rados/operations/erasure-code-profile>'_ for details.
`expected_num_objects`	The expected number of objects in the pool.	`0`	The expected number of objects for this pool. By setting this value (together with a negative filestore merge threshold), the PG folder splitting would happen at the pool creation time, to avoid the latency impact to do a runtime folder splitting.
`quota_max_objects`	Max number of objects within a pool.	`0`	0 means unlimited.
`quota_max_bytes`	Max number of bytes of the pool, capacity.	`0`	0 means unlimited.
`replica_size`	The number of object replicas	`1`	The number of object copies within a pool.
`replica_min_size`	The minimal number of object replicas	`1`	The minimal number of object replicas for I/O tasks.

Parameters and roles

Please check official documentation to get a meaning of the parameters.

Configuration example

The following is an example of the *Add Ceph pool* plugin configuration:

```
{
    "global_vars": {
        "cluster": "ceph",
        "crush_ruleset_name": "replicated_ruleset",
        "erasure_code_profile": "default",
        "expected_num_objects": 0,
        "pg_num": 0,
        "pgp_num": 0,
        "pool_name": "test",
        "pool_type": "replicated",
        "quota_max_bytes": 1024,
        "quota_max_objects": 50,
        "replica_min_size": 1,
        "replica_size": 1
    },
    "inventory": {
        "_meta": {
            "hostvars": {
                "10.0.0.21": {
                    "ansible_user": "ansible"
                }
            }
        },
        "mons": [
            "10.0.0.21"
        ]
    }
}
```

Add RBD Mirror Host

The *Add RBD Mirror Host* playbook plugin deploys rbdmirror daemon and setup mirroring of pools between different clusters.

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

Overview

The following table shows the general information about the *Add RBD Mirror* plugin:

Property	Value
ID	`add_rbdmirror`
Name	Add RBD Mirror
Required Server List	Yes

This 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
>=1.1,<1.2	v2.2.4

Parameters and roles

For consistency, Decapod checks the Ceph version it is going to deploy. If a Ceph cluster has inconsistent versions, the deployment stops and you must fix the versions withing the cluster. If the Ceph version you are going to deploy is newer that the deployed ones, the process will also stop and you must update the cluster packages first.

The following parameters are responsble for such checks:

`ceph_version_verify` A boolean setting that checks that strict mode is enabled. If set to `false`, no verification described above is performed.

`ceph_version_verify_packagename` The name of the package to check. It is not required to configure this setting.

Configuration example

The following is an example of the *Add RBD Mirror* plugin configuration:

```
{
  "global_vars": {
    "add_peers": true,
    "ceph_nfs_access_type": "RW",
    "ceph_nfs_ceph_access_type": "RW",
    "ceph_nfs_ceph_export_id": 20134,
    "ceph_nfs_ceph_protocols": "3,4",
    "ceph_nfs_ceph_pseudo_path": "/cephobject",
    "ceph_nfs_export_id": 20134,
    "ceph_nfs_log_file": "/var/log/ganesha.log",
    "ceph_nfs_protocols": "3,4",
    "ceph_nfs_pseudo_path": "/cephfile",
    "ceph_nfs_rgw_access_type": "RW",
    "ceph_nfs_rgw_export_id": 20134,
    "ceph_nfs_rgw_protocols": "3,4",
    "ceph_nfs_rgw_pseudo_path": "/ceph",
    "ceph_nfs_rgw_user": "cephnfs",
    "ceph_restapi_port": 5000,
    "ceph_stable": true,
    "ceph_stable_distro_source": "jewel-xenial",
    "ceph_stable_release": "jewel",
    "ceph_stable_release_uca": "jewel-xenial",
    "ceph_stable_repo": "http://mirror.fuel-infra.org/decapod/ceph/jewel-xenial",
    "ceph_stable_repo_key": "AF94F6A6A254F5F0",
    "ceph_stable_repo_keyserver": "hkp://keyserver.ubuntu.com:80",
    "ceph_version_verify": true,
    "ceph_version_verify_packagename": "ceph-common",
    "cluster": "ceph",
    "cluster_network": "10.0.0.0/24",
    "common_single_host_mode": true,
    "copy_admin_key": true,
    "dmcrypt_dedicated_journal": false,
    "dmcrypt_journal_collocation": false,
    "fsid": "0860ebc0-35af-465f-80e8-c394fc9af2de",
    "journal_collocation": false,
    "journal_size": 512,
```

```
    "max_open_files": 131072,
    "mds_allow_multimds": true,
    "mds_max_mds": 3,
    "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": "10.0.0.0/24",
    "radosgw_civetweb_num_threads": 50,
    "radosgw_civetweb_port": 8080,
    "radosgw_dns_s3website_name": "your.subdomain.tld",
    "radosgw_static_website": false,
    "radosgw_usage_log": false,
    "radosgw_usage_log_flush_threshold": 1024,
    "radosgw_usage_log_tick_interval": 30,
    "radosgw_usage_max_shards": 32,
    "radosgw_usage_max_user_shards": 1,
    "raw_multi_journal": true
  },
  "inventory": {
    "_meta": {
      "hostvars": {
        "10.0.0.20": {
          "ansible_user": "ansible",
          "devices": [
            "/dev/vdj",
            "/dev/vdb",
            "/dev/vdd",
            "/dev/vdh",
            "/dev/vdf"
          ],
          "monitor_address": "10.0.0.20",
          "raw_journal_devices": [
            "/dev/vdg",
            "/dev/vdi",
            "/dev/vde",
            "/dev/vdc",
            "/dev/vdk"
          ]
        },
        "10.0.0.21": {
          "ansible_user": "ansible",
          "devices": [
            "/dev/vdj",
            "/dev/vdb",
            "/dev/vdd",
            "/dev/vdh",
            "/dev/vdf"
          ],
          "monitor_address": "10.0.0.21",
```

```
          "raw_journal_devices": [
            "/dev/vdg",
            "/dev/vdi",
            "/dev/vde",
            "/dev/vdc",
            "/dev/vdk"
          ]
        },
        "10.0.0.22": {
          "ansible_user": "ansible",
          "devices": [
            "/dev/vdj",
            "/dev/vdb",
            "/dev/vdd",
            "/dev/vdh",
            "/dev/vdf"
          ],
          "monitor_address": "10.0.0.22",
          "raw_journal_devices": [
            "/dev/vdg",
            "/dev/vdi",
            "/dev/vde",
            "/dev/vdc",
            "/dev/vdk"
          ]
        },
        "10.0.0.24": {
          "ansible_user": "ansible",
          "rbd_mirrors": {
            "admin@another": [
              "rbd"
            ]
          }
        }
      }
    },
    "already_deployed": [
      "10.0.0.21",
      "10.0.0.20",
      "10.0.0.22"
    ],
    "mons": [
      "10.0.0.20"
    ],
    "rbdmirrors": [
      "10.0.0.24"
    ]
  }
}
```


Add REST API to the host

This plugin allows to deploy Ceph admin REST API to the host.

This REST API is not well documented, but you may find some description on Ceph Planet agregator.

Overview

The following table shows the general information about the *Add Ceph REST API* plugin:

Property	Value
ID	`add_restapi`
Name	Add Ceph REST API
Required Server List	Yes

This 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
>=1.1,<1.2	v2.2.4

Parameters and roles

For consistency, Decapod checks the Ceph version it is going to deploy. If a Ceph cluster has inconsistent versions, the deployment stops and you must fix the versions withing the cluster. If the Ceph version you are going to deploy is newer that the deployed ones, the process will also stop and you must update the cluster packages first.

The following parameters are responsble for such checks:

`ceph_version_verify` A boolean setting that checks that strict mode is enabled. If set to `false`, no verification described above is performed.

`ceph_version_verify_packagename` The name of the package to check. It is not required to configure this setting.

Configuration example

The following is an example of the *Add Ceph REST API* plugin configuration:

```
{
  "global_vars": {
    "ceph_nfs_access_type": "RW",
    "ceph_nfs_ceph_access_type": "RW",
    "ceph_nfs_ceph_export_id": 20134,
    "ceph_nfs_ceph_protocols": "3,4",
    "ceph_nfs_ceph_pseudo_path": "/cephobject",
    "ceph_nfs_export_id": 20134,
    "ceph_nfs_log_file": "/var/log/ganesha.log",
    "ceph_nfs_protocols": "3,4",
    "ceph_nfs_pseudo_path": "/cephfile",
    "ceph_nfs_rgw_access_type": "RW",
    "ceph_nfs_rgw_export_id": 20134,
    "ceph_nfs_rgw_protocols": "3,4",
    "ceph_nfs_rgw_pseudo_path": "/ceph",
    "ceph_nfs_rgw_user": "cephnfs",
    "ceph_restapi_port": 5000,
    "ceph_stable": true,
    "ceph_stable_distro_source": "jewel-xenial",
    "ceph_stable_release": "jewel",
    "ceph_stable_release_uca": "jewel-xenial",
    "ceph_stable_repo": "http://mirror.fuel-infra.org/decapod/ceph/jewel-xenial",
    "ceph_stable_repo_key": "AF94F6A6A254F5F0",
    "ceph_stable_repo_keyserver": "hkp://keyserver.ubuntu.com:80",
```

```
    "ceph_version_verify": true,
    "ceph_version_verify_packagename": "ceph-common",
    "cluster": "ceph",
    "cluster_network": "10.0.0.0/24",
    "common_single_host_mode": true,
    "copy_admin_key": true,
    "dmcrypt_dedicated_journal": false,
    "dmcrypt_journal_collocation": false,
    "fsid": "0860ebc0-35af-465f-80e8-c394fc9af2de",
    "journal_collocation": false,
    "journal_size": 512,
    "max_open_files": 131072,
    "mds_allow_multimds": true,
    "mds_max_mds": 3,
    "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": "10.0.0.0/24",
    "radosgw_civetweb_num_threads": 50,
    "radosgw_civetweb_port": 8080,
    "radosgw_dns_s3website_name": "your.subdomain.tld",
    "radosgw_static_website": false,
    "radosgw_usage_log": false,
    "radosgw_usage_log_flush_threshold": 1024,
    "radosgw_usage_log_tick_interval": 30,
    "radosgw_usage_max_shards": 32,
    "radosgw_usage_max_user_shards": 1,
    "raw_multi_journal": true
  },
  "inventory": {
    "_meta": {
      "hostvars": {
        "10.0.0.20": {
          "ansible_user": "ansible",
          "devices": [
            "/dev/vdj",
            "/dev/vdb",
            "/dev/vdd",
            "/dev/vdh",
            "/dev/vdf"
          ],
          "monitor_address": "10.0.0.20",
          "raw_journal_devices": [
            "/dev/vdg",
            "/dev/vdi",
            "/dev/vde",
            "/dev/vdc",
            "/dev/vdk"
          ]
```

```
        },
        "10.0.0.21": {
          "ansible_user": "ansible",
          "devices": [
            "/dev/vdj",
            "/dev/vdb",
            "/dev/vdd",
            "/dev/vdh",
            "/dev/vdf"
          ],
          "monitor_address": "10.0.0.21",
          "raw_journal_devices": [
            "/dev/vdg",
            "/dev/vdi",
            "/dev/vde",
            "/dev/vdc",
            "/dev/vdk"
          ]
        },
        "10.0.0.22": {
          "ansible_user": "ansible",
          "devices": [
            "/dev/vdj",
            "/dev/vdb",
            "/dev/vdd",
            "/dev/vdh",
            "/dev/vdf"
          ],
          "monitor_address": "10.0.0.22",
          "raw_journal_devices": [
            "/dev/vdg",
            "/dev/vdi",
            "/dev/vde",
            "/dev/vdc",
            "/dev/vdk"
          ]
        },
        "10.0.0.23": {
          "ansible_user": "ansible"
        },
        "10.0.0.24": {
          "ansible_user": "ansible"
        }
      }
    },
    "already_deployed": [
      "10.0.0.21",
      "10.0.0.20",
      "10.0.0.22"
    ],
    "mons": [
      "10.0.0.20"
    ],
    "restapis": [
      "10.0.0.24",
      "10.0.0.23"
    ]
  }
```

```
}
```

Add Rados Gateway host

This plugin allows to deploy Ceph Rados Gateway to the host.

Overview

The following table shows the general information about the *Add Rados Gateway* plugin:

Property	Value
ID	`add_rgw`
Name	Add Rados Gateway host
Required Server List	Yes

This 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
>=1.1,<1.2	v2.2.4

Parameters and roles

For consistency, Decapod checks the Ceph version it is going to deploy. If a Ceph cluster has inconsistent versions, the deployment stops and you must fix the versions withing the cluster. If the Ceph version you are going to deploy is newer that the deployed ones, the process will also stop and you must update the cluster packages first.

The following parameters are responsble for such checks:

`ceph_version_verify` A boolean setting that checks that strict mode is enabled. If set to `false`, no verification described above is performed.

`ceph_version_verify_packagename` The name of the package to check. It is not required to configure this setting.

Configuration example

The following is an example of the *Add Rados Gateway* plugin configuration:

```
{
  "global_vars": {
    "ceph_nfs_access_type": "RW",
    "ceph_nfs_ceph_access_type": "RW",
    "ceph_nfs_ceph_export_id": 20134,
    "ceph_nfs_ceph_protocols": "3,4",
    "ceph_nfs_ceph_pseudo_path": "/cephobject",
    "ceph_nfs_export_id": 20134,
    "ceph_nfs_log_file": "/var/log/ganesha.log",
    "ceph_nfs_protocols": "3,4",
    "ceph_nfs_pseudo_path": "/cephfile",
    "ceph_nfs_rgw_access_type": "RW",
    "ceph_nfs_rgw_export_id": 20134,
    "ceph_nfs_rgw_protocols": "3,4",
    "ceph_nfs_rgw_pseudo_path": "/ceph",
```

```
    "ceph_nfs_rgw_user": "cephnfs",
    "ceph_restapi_port": 5000,
    "ceph_stable": true,
    "ceph_stable_distro_source": "jewel-xenial",
    "ceph_stable_release": "jewel",
    "ceph_stable_release_uca": "jewel-xenial",
    "ceph_stable_repo": "http://mirror.fuel-infra.org/decapod/ceph/jewel-xenial",
    "ceph_stable_repo_key": "AF94F6A6A254F5F0",
    "ceph_stable_repo_keyserver": "hkp://keyserver.ubuntu.com:80",
    "ceph_version_verify": true,
    "ceph_version_verify_packagename": "ceph-common",
    "cluster": "ceph",
    "cluster_network": "10.0.0.0/24",
    "common_single_host_mode": true,
    "copy_admin_key": true,
    "dmcrypt_dedicated_journal": false,
    "dmcrypt_journal_collocation": false,
    "fsid": "0860ebc0-35af-465f-80e8-c394fc9af2de",
    "journal_collocation": false,
    "journal_size": 512,
    "max_open_files": 131072,
    "mds_allow_multimds": true,
    "mds_max_mds": 3,
    "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": "10.0.0.0/24",
    "radosgw_civetweb_num_threads": 50,
    "radosgw_civetweb_port": 8080,
    "radosgw_dns_s3website_name": "your.subdomain.tld",
    "radosgw_static_website": false,
    "radosgw_usage_log": false,
    "radosgw_usage_log_flush_threshold": 1024,
    "radosgw_usage_log_tick_interval": 30,
    "radosgw_usage_max_shards": 32,
    "radosgw_usage_max_user_shards": 1,
    "raw_multi_journal": true
  },
  "inventory": {
    "_meta": {
      "hostvars": {
        "10.0.0.20": {
          "ansible_user": "ansible",
          "devices": [
            "/dev/vdj",
            "/dev/vdb",
            "/dev/vdd",
            "/dev/vdh",
            "/dev/vdf"
```

```
        ],
        "monitor_address": "10.0.0.20",
        "raw_journal_devices": [
          "/dev/vdg",
          "/dev/vdi",
          "/dev/vde",
          "/dev/vdc",
          "/dev/vdk"
        ]
      },
      "10.0.0.21": {
        "ansible_user": "ansible",
        "devices": [
          "/dev/vdj",
          "/dev/vdb",
          "/dev/vdd",
          "/dev/vdh",
          "/dev/vdf"
        ],
        "monitor_address": "10.0.0.21",
        "raw_journal_devices": [
          "/dev/vdg",
          "/dev/vdi",
          "/dev/vde",
          "/dev/vdc",
          "/dev/vdk"
        ]
      },
      "10.0.0.22": {
        "ansible_user": "ansible",
        "devices": [
          "/dev/vdj",
          "/dev/vdb",
          "/dev/vdd",
          "/dev/vdh",
          "/dev/vdf"
        ],
        "monitor_address": "10.0.0.22",
        "raw_journal_devices": [
          "/dev/vdg",
          "/dev/vdi",
          "/dev/vde",
          "/dev/vdc",
          "/dev/vdk"
        ]
      },
      "10.0.0.23": {
        "ansible_user": "ansible"
      },
      "10.0.0.24": {
        "ansible_user": "ansible"
      }
    }
  },
  "already_deployed": [
    "10.0.0.21",
    "10.0.0.20",
    "10.0.0.22"
```

```
    ],
    "mons": [
      "10.0.0.20"
    ],
    "rgws": [
      "10.0.0.24",
      "10.0.0.23"
    ]
  }
}
```

Cinder integration

The *Cinder integration* plugin allows you to perform an integration between a deployed Ceph cluster and the OpenStack Block storage service (Cinder).

Overview

The following table shows general information about the *Cinder integration* plugin:

Property	Value
ID	`cinder_integration`
Name	Cinder Integration
Required Server List	No

The following table lists the available hints for the plugin:

Hint	Title	Default value	Description
`cinder`	Use Cinder with Ceph back end	`True`	Defines if Cinder will be used with Ceph back end. This is required to create a `volumes` pool by default.

Cinder requires keyrings and the contents of the Ceph configuration file, for example, `ceph.conf`. This plugin creates required keyrings in Ceph, creates required pools, and allows Decapod to return required files.

To integrate Cinder:

1. Run the plugin through the Decapod Web UI.
2. Obtain the required files:

   ```
   $ decapod cluster cinder-integration a2b813b2-df23-462b-8dab-6a80f9bc7fce
   ```

   Where `a2b813b2-df23-462b-8dab-6a80f9bc7fce` is the cluster ID. This command will return the contents of required files.

   To obtain the files and store in the file system, use the `--store` option:

   ```
   $ decapod cluster cinder-integration --store 8b205db5-3d29-4f1b-82a5-e5cefb522d4f
   ```

   This command will output the contents of the files and store them in the file system.

Parameters and roles

The *Cinder integration* plugin has the following parameters:

cluster Name of the cluster to use.

clients A mapping of client to create in Ceph for permissions.

pools A mapping of pool name to PG count that should be used.

Configuration example

The following is an example of the *Cinder integration* plugin configuration:

```
{
  "global_vars": {
    "cluster": "ceph"
  },
  "inventory": {
    "_meta": {
      "hostvars": {
        "10.0.0.20": {
          "ansible_user": "ansible",
          "clients": {
            "compute": {
              "mon": "allow r",
              "osd": "allow class-read object_prefix rbd_children, allow \
              rwx pool=compute, allow rwx pool=volumes, allow rx pool=images"
            },
            "images": {
              "mon": "allow r",
              "osd": "allow class-read object_prefix rbd_children, allow \
              rwx pool=images"
            },
            "volumes": {
              "mon": "allow r",
              "osd": "allow class-read object_prefix rbd_children, allow \
              rwx pool=volumes, allow rx pool=images"
            }
          },
          "pools": {
            "compute": 64,
            "images": 64,
            "volumes": 64
          }
        }
      }
    },
    "mons": [
      "10.0.0.20"
    ]
  }
}
```

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.

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 following table lists the available hints for the plugin:

Hint	Title	Default value	Description
`dmcrypt`	Use dmcrypted OSDs	`False`	Defines the dmcrypt usage for OSD devices.
`collocation`	Collocate OSD data and journal on same devices	`False`	Defines whether the journal and data have to be placed on the same devices.
`rest_api`	Setup Ceph RestAPI	`False`	Defines the RestAPI installation for Ceph.
`mon_count`	The number of monitors to deploy	`3`	Defines the number of monitors.

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
>=1.1,<1.2	v2.2.4

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.

The `ceph-ansible` project supports two deployment modes of a Ceph cluster: with journal collocation and on separate drives, and also with `dmcrypt` and without it. Therefore, there are four possible combinations.

The following table lists the possible combinations:

Setting	Combination 1	Combination 2	Combination 3	Combination 4
collocation	`true`	`true`	`false`	`false`
dmcrypt	`true`	`false`	`true`	`false`
journal_collocation	`false`	`true`	`false`	`false`
raw_multi_journal	`true`	`false`	`false`	`true`
dmcrypt_journal_collocation	`false`	`false`	`false`	`false`
dmcrypt_dedicated_journal	`false`	`false`	`true`	`false`
Data devices option name	`devices`	`devices`	`devices`	`devices`
Journal devices option name	–	–	`raw_journal_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_nfs_access_type": "RW",
    "ceph_nfs_ceph_access_type": "RW",
    "ceph_nfs_ceph_export_id": 20134,
    "ceph_nfs_ceph_protocols": "3,4",
    "ceph_nfs_ceph_pseudo_path": "/cephobject",
    "ceph_nfs_export_id": 20134,
    "ceph_nfs_log_file": "/var/log/ganesha.log",
    "ceph_nfs_protocols": "3,4",
    "ceph_nfs_pseudo_path": "/cephfile",
    "ceph_nfs_rgw_access_type": "RW",
    "ceph_nfs_rgw_export_id": 20134,
    "ceph_nfs_rgw_protocols": "3,4",
    "ceph_nfs_rgw_pseudo_path": "/ceph",
    "ceph_nfs_rgw_user": "cephnfs",
    "ceph_restapi_port": 5000,
    "ceph_stable": true,
    "ceph_stable_distro_source": "jewel-xenial",
    "ceph_stable_release": "jewel",
    "ceph_stable_release_uca": "jewel-xenial",
    "ceph_stable_repo": "http://mirror.fuel-infra.org/decapod/ceph/jewel-xenial",
    "ceph_stable_repo_key": "AF94F6A6A254F5F0",
    "ceph_stable_repo_keyserver": "hkp://keyserver.ubuntu.com:80",
    "ceph_version_verify": false,
    "ceph_version_verify_packagename": "ceph-common",
    "cluster": "ceph",
    "cluster_network": "10.0.0.0/24",
    "common_single_host_mode": true,
    "copy_admin_key": true,
    "dmcrypt_dedicated_journal": false,
    "dmcrypt_journal_collocation": false,
```

```
    "fsid": "0860ebc0-35af-465f-80e8-c394fc9af2de",
    "journal_collocation": false,
    "journal_size": 512,
    "max_open_files": 131072,
    "mds_allow_multimds": true,
    "mds_max_mds": 3,
    "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": "10.0.0.0/24",
    "radosgw_civetweb_num_threads": 50,
    "radosgw_civetweb_port": 8080,
    "radosgw_dns_s3website_name": "your.subdomain.tld",
    "radosgw_static_website": false,
    "radosgw_usage_log": false,
    "radosgw_usage_log_flush_threshold": 1024,
    "radosgw_usage_log_tick_interval": 30,
    "radosgw_usage_max_shards": 32,
    "radosgw_usage_max_user_shards": 1,
    "raw_multi_journal": true,
    "restapi_template_local_path": "/usr/local/lib/python3.5/dist-packages/decapod_
↪plugin_playbook_deploy_cluster/ceph-rest-api.service"
  },
  "inventory": {
    "_meta": {
      "hostvars": {
        "10.0.0.20": {
          "ansible_user": "ansible",
          "devices": [
            "/dev/vdj",
            "/dev/vdb",
            "/dev/vdd",
            "/dev/vdh",
            "/dev/vdf"
          ],
          "monitor_address": "10.0.0.20",
          "raw_journal_devices": [
            "/dev/vdg",
            "/dev/vdi",
            "/dev/vde",
            "/dev/vdc",
            "/dev/vdk"
          ]
        },
        "10.0.0.21": {
          "ansible_user": "ansible",
          "devices": [
            "/dev/vdj",
            "/dev/vdb",
```

```
            "/dev/vdd",
            "/dev/vdh",
            "/dev/vdf"
          ],
          "monitor_address": "10.0.0.21",
          "raw_journal_devices": [
            "/dev/vdg",
            "/dev/vdi",
            "/dev/vde",
            "/dev/vdc",
            "/dev/vdk"
          ]
        },
        "10.0.0.22": {
          "ansible_user": "ansible",
          "devices": [
            "/dev/vdj",
            "/dev/vdb",
            "/dev/vdd",
            "/dev/vdh",
            "/dev/vdf"
          ],
          "monitor_address": "10.0.0.22",
          "raw_journal_devices": [
            "/dev/vdg",
            "/dev/vdi",
            "/dev/vde",
            "/dev/vdc",
            "/dev/vdk"
          ]
        }
      }
    },
    "clients": [],
    "iscsi_gw": [],
    "mdss": [],
    "mons": [
      "10.0.0.20"
    ],
    "nfss": [],
    "osds": [
      "10.0.0.21",
      "10.0.0.22"
    ],
    "rbdmirrors": [],
    "restapis": [],
    "rgws": []
  }
}
```


Purge cluster

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

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.

Telegraf removal

While the *Telegraf integration* plugin installs and configures Telegraf, the *Telegraf removal* plugin uninstalls Telegraf or its managed section from the configuration.

Overview

The following table shows general information about the *Telegraf removal* plugin:

Property	Value
ID	`purge_telegraf`
Name	Telegraf removal
Required Server List	Yes

The following hints are available for the plugin:

remove_config_section_only If set to `true`, the plugin will remove the corresponding section created by *Telegraf integration* plugin from `/etc/telegraf/telegraf.conf`.

Configuration example

The following is an example of the *Telegraf removal* plugin configuration:

```
{
  "global_vars": {
    "configpath": "/etc/telegraf/telegraf.conf",
    "remove_config_section_only": false
  },
  "inventory": {
    "_meta": {
      "hostvars": {
        "10.0.0.20": {
          "ansible_user": "ansible"
        }
      }
    },
    "telegraf": [
      "10.0.0.20"
    ]
  }
}
```

Remove CLI/RBD clients from host

The *Remove CLI/RBD clients from host* playbook plugin removes stuff which is necessary to run **ceph**, **rados** and **rbd** clients from the host.

Overview

The following table shows general information about the *Remove CLI/RBD clients from the host* plugin:

Property	Value
ID	`remove_client`
Name	Remove CLI/RBD clients from the host
Required server list	Yes

Parameters and roles

uninstall_packages By default this plugin do not uninstall packages, it only removes keyrings and settings. With this setting enabled, packages will be uninstalled.

apt_purge Uninstall packages with `--purge` option of APT (remove package config files).

Configuration example

The following is an example of the *Remove CLI/RBD clients from the host* plugin configuration:

```
{
  "global_vars": {
    "apt_purge": true,
    "cluster": "ceph",
    "uninstall_packages": true
  },
  "inventory": {
    "_meta": {
      "hostvars": {
        "10.0.0.21": {
          "ansible_user": "ansible",
          "devices": [
            "/dev/vdj",
            "/dev/vdb",
            "/dev/vdd",
            "/dev/vdh",
            "/dev/vdf"
          ],
          "monitor_address": "10.0.0.21",
          "raw_journal_devices": [
            "/dev/vdg",
            "/dev/vdi",
            "/dev/vde",
            "/dev/vdc",
            "/dev/vdk"
          ]
        },
        "10.0.0.22": {
          "ansible_user": "ansible",
          "devices": [
```

```
            "/dev/vdj",
            "/dev/vdb",
            "/dev/vdd",
            "/dev/vdh",
            "/dev/vdf"
          ],
          "monitor_address": "10.0.0.22",
          "raw_journal_devices": [
            "/dev/vdg",
            "/dev/vdi",
            "/dev/vde",
            "/dev/vdc",
            "/dev/vdk"
          ]
        }
      }
    },
    "clients": [
      "10.0.0.21",
      "10.0.0.22"
    ]
  }
}
```

Remove metadata servers from host

the *Remove metadata servers from host* playbook plugin removes and disables metadata server on the host.

Overview

The following table shows general information about the *Remove metadata server from host* plugin:

Property	Value
ID	`remove_mds`
Name	Remove metadata server from host
Required server list	Yes

Configuration example

The following is an example of the *Remove metadata server from host* plugin configuration:

```
{
  "global_vars": {
    "ceph_nfs_access_type": "RW",
    "ceph_nfs_ceph_access_type": "RW",
    "ceph_nfs_ceph_export_id": 20134,
    "ceph_nfs_ceph_protocols": "3,4",
    "ceph_nfs_ceph_pseudo_path": "/cephobject",
    "ceph_nfs_export_id": 20134,
    "ceph_nfs_log_file": "/var/log/ganesha.log",
    "ceph_nfs_protocols": "3,4",
    "ceph_nfs_pseudo_path": "/cephfile",
    "ceph_nfs_rgw_access_type": "RW",
    "ceph_nfs_rgw_export_id": 20134,
```

```
    "ceph_nfs_rgw_protocols": "3,4",
    "ceph_nfs_rgw_pseudo_path": "/ceph",
    "ceph_nfs_rgw_user": "cephnfs",
    "ceph_restapi_port": 5000,
    "ceph_stable": true,
    "ceph_stable_distro_source": "jewel-xenial",
    "ceph_stable_release": "jewel",
    "ceph_stable_release_uca": "jewel-xenial",
    "ceph_stable_repo": "http://mirror.fuel-infra.org/decapod/ceph/jewel-xenial",
    "ceph_stable_repo_key": "AF94F6A6A254F5F0",
    "ceph_stable_repo_keyserver": "hkp://keyserver.ubuntu.com:80",
    "ceph_version_verify": false,
    "ceph_version_verify_packagename": "ceph-common",
    "cluster": "ceph",
    "cluster_network": "10.0.0.0/24",
    "common_single_host_mode": true,
    "copy_admin_key": true,
    "dmcrypt_dedicated_journal": false,
    "dmcrypt_journal_collocation": false,
    "fsid": "0860ebc0-35af-465f-80e8-c394fc9af2de",
    "journal_collocation": false,
    "journal_size": 512,
    "max_open_files": 131072,
    "mds_allow_multimds": true,
    "mds_max_mds": 3,
    "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": "10.0.0.0/24",
    "radosgw_civetweb_num_threads": 50,
    "radosgw_civetweb_port": 8080,
    "radosgw_dns_s3website_name": "your.subdomain.tld",
    "radosgw_static_website": false,
    "radosgw_usage_log": false,
    "radosgw_usage_log_flush_threshold": 1024,
    "radosgw_usage_log_tick_interval": 30,
    "radosgw_usage_max_shards": 32,
    "radosgw_usage_max_user_shards": 1,
    "raw_multi_journal": true
  },
  "inventory": {
    "_meta": {
      "hostvars": {
        "10.0.0.20": {
          "ansible_user": "ansible",
          "devices": [
            "/dev/vdj",
            "/dev/vdb",
            "/dev/vdd",
```

```
            "/dev/vdh",
            "/dev/vdf"
          ],
          "monitor_address": "10.0.0.20",
          "raw_journal_devices": [
            "/dev/vdg",
            "/dev/vdi",
            "/dev/vde",
            "/dev/vdc",
            "/dev/vdk"
          ]
        },
        "10.0.0.21": {
          "ansible_user": "ansible",
          "devices": [
            "/dev/vdj",
            "/dev/vdb",
            "/dev/vdd",
            "/dev/vdh",
            "/dev/vdf"
          ],
          "monitor_address": "10.0.0.21",
          "raw_journal_devices": [
            "/dev/vdg",
            "/dev/vdi",
            "/dev/vde",
            "/dev/vdc",
            "/dev/vdk"
          ]
        },
        "10.0.0.22": {
          "ansible_user": "ansible",
          "devices": [
            "/dev/vdj",
            "/dev/vdb",
            "/dev/vdd",
            "/dev/vdh",
            "/dev/vdf"
          ],
          "monitor_address": "10.0.0.22",
          "raw_journal_devices": [
            "/dev/vdg",
            "/dev/vdi",
            "/dev/vde",
            "/dev/vdc",
            "/dev/vdk"
          ]
        }
      }
    },
    "mdss_to_remove": [
      "10.0.0.21",
      "10.0.0.22"
    ],
    "mons": [
      "10.0.0.20"
    ]
  }
```

```
}
```

Parameters and roles

This plugin has no specific parameters and roles.

Remove monitor host

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

Overview

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

Property	Value
ID	`remove_mon`
Name	Remove monitor host
Required server list	Yes

Note: You must have enough monitor hosts to make PAXOS quorum.

Configuration example

The following is an example of the *Remove monitor 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",
      "10.10.0.12"
    ]
  }
}
```

This playbook has the simplest possible configuration. You only need to define the monitors you want to remove.

Remove NFS Gatweway

The *Remove NFS Gateway* playbook plugin allows you to remove a host with monitor from a cluster.

Overview

The following table shows general information about the *Remove NFS Gateway* plugin:

Property	Value
ID	`remove_nfs`
Name	Remove NFS Gateway
Required server list	Yes

Parameters and roles

`remove_rgw` By default, this plugin removes only NFS gateway but not RGW. Activate this hint if you want to remove RGW as well.

`remove_nfs_rgw_user` This hint removes special auth data for NFS/RGW user from the cluster.

Configuration example

The following is an example of the *Remove NFS Gateway* plugin configuration:

```
{
  "global_vars": {
    "ceph_nfs_rgw_user": "cephnfs",
    "cluster": "ceph",
    "remove_nfs_rgw_user": true
  },
  "inventory": {
    "_meta": {
      "hostvars": {
        "10.0.0.22": {
          "ansible_user": "ansible",
          "devices": [
            "/dev/vdj",
            "/dev/vdb",
            "/dev/vdd",
            "/dev/vdh",
            "/dev/vdf"
          ],
          "monitor_address": "10.0.0.22",
          "raw_journal_devices": [
            "/dev/vdg",
            "/dev/vdi",
            "/dev/vde",
            "/dev/vdc",
            "/dev/vdk"
          ]
        }
      }
    },
    "nfss": [
      "10.0.0.22"
```

```
    ],
    "rgws": [
      "10.0.0.22"
    ]
  }
}
```

Remove OSD host

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

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": {
        "10.0.0.20": {
          "ansible_user": "ansible",
          "devices": [
            "/dev/vdj",
            "/dev/vdb",
            "/dev/vdd",
            "/dev/vdh",
            "/dev/vdf"
          ],
          "monitor_address": "10.0.0.20",
          "raw_journal_devices": [
            "/dev/vdg",
            "/dev/vdi",
            "/dev/vde",
            "/dev/vdc",
            "/dev/vdk"
          ]
        },
        "10.0.0.21": {
          "ansible_user": "ansible",
          "devices": [
            "/dev/vdj",
            "/dev/vdb",
```

```
            "/dev/vdd",
            "/dev/vdh",
            "/dev/vdf"
          ],
          "monitor_address": "10.0.0.21",
          "raw_journal_devices": [
            "/dev/vdg",
            "/dev/vdi",
            "/dev/vde",
            "/dev/vdc",
            "/dev/vdk"
          ]
        },
        "10.0.0.22": {
          "ansible_user": "ansible",
          "devices": [
            "/dev/vdj",
            "/dev/vdb",
            "/dev/vdd",
            "/dev/vdh",
            "/dev/vdf"
          ],
          "monitor_address": "10.0.0.22",
          "raw_journal_devices": [
            "/dev/vdg",
            "/dev/vdi",
            "/dev/vde",
            "/dev/vdc",
            "/dev/vdk"
          ]
        }
      }
    },
    "mons": [
      "10.0.0.20"
    ],
    "osds": [
      "10.0.0.21",
      "10.0.0.22"
    ]
  }
}
```

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

Remove Ceph pool

The *Remove Ceph pool* playbook plugin allows you to remove one or many pools from Ceph cluster.

Overview

The following table shows the general information about the *Remove Ceph pool* plugin:

Property	Value
ID	`remove_pool`
Name	Remove Ceph pool
Required Server List	No

The following table lists the available hints for the plugin:

Hint	Title	Default value	Description
`pool_name`	The name of the pool to remove.	`test`	Defines the name of the pool you are going to remove. The single name, please check playbook configuration to put more names.

Parameters and roles

Plugin supports only one parameter: `pool_names`. This is a list of pools user wants to remove from Ceph cluster.

Configuration example

The following is an example of the *Remove Ceph pool* plugin configuration:

```
{
  "global_vars": {
    "cluster": "ceph",
    "pool_names": [
      "test",
      "images"
    ]
  },
  "inventory": {
    "_meta": {
      "hostvars": {
        "10.0.0.21": {
          "ansible_user": "ansible"
        }
      }
    },
    "mons": [
      "10.0.0.21"
    ]
  }
}
```

Remove RBD Mirror from the host

The *Remove RBD Mirror from the host* playbook plugin removes RBD mirroring daemon.

Overview

The following table shows general information about the *Remove RBD Mirror from the host* plugin:

Property	Value
ID	`remove_rbdmirror`
Name	Remove RBD mirror from the host
Required server list	Yes

Parameters and roles

This plugin does not have specific parameters or roles.

Configuration example

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

```
{
  "global_vars": {
    "cluster": "ceph"
  },
  "inventory": {
    "_meta": {
      "hostvars": {
        "10.0.0.21": {
          "ansible_user": "ansible",
          "devices": [
            "/dev/vdj",
            "/dev/vdb",
            "/dev/vdd",
            "/dev/vdh",
            "/dev/vdf"
          ],
          "monitor_address": "10.0.0.21",
          "raw_journal_devices": [
            "/dev/vdg",
            "/dev/vdi",
            "/dev/vde",
            "/dev/vdc",
            "/dev/vdk"
          ]
        },
        "10.0.0.22": {
          "ansible_user": "ansible",
          "devices": [
            "/dev/vdj",
            "/dev/vdb",
            "/dev/vdd",
            "/dev/vdh",
            "/dev/vdf"
          ],
          "monitor_address": "10.0.0.22",
          "raw_journal_devices": [
            "/dev/vdg",
            "/dev/vdi",
            "/dev/vde",
            "/dev/vdc",
            "/dev/vdk"
          ]
        }
      }
    },
    "rbdmirrors": [
      "10.0.0.21",
      "10.0.0.22"
    ]
```

```
  }
}
```

Remove Ceph REST API from host

The *Remove Ceph REST API from host* playbook plugin removes Ceph management REST API.

Overview

The following table shows general information about the *Remove Ceph REST API* plugin:

Property	Value
ID	`remove_restapi`
Name	Remove Ceph REST API
Required server list	Yes

Parameters and roles

This plugin does not have specific parameters or roles.

Configuration example

The following is an example of the *Remove Ceph REST API* plugin configuration:

```
{
  "global_vars": {
    "cluster": "ceph"
  },
  "inventory": {
    "_meta": {
      "hostvars": {
        "10.0.0.21": {
          "ansible_user": "ansible",
          "devices": [
            "/dev/vdj",
            "/dev/vdb",
            "/dev/vdd",
            "/dev/vdh",
            "/dev/vdf"
          ],
          "monitor_address": "10.0.0.21",
          "raw_journal_devices": [
            "/dev/vdg",
            "/dev/vdi",
            "/dev/vde",
            "/dev/vdc",
            "/dev/vdk"
          ]
        },
        "10.0.0.22": {
          "ansible_user": "ansible",
          "devices": [
            "/dev/vdj",
```

```
            "/dev/vdb",
            "/dev/vdd",
            "/dev/vdh",
            "/dev/vdf"
          ],
          "monitor_address": "10.0.0.22",
          "raw_journal_devices": [
            "/dev/vdg",
            "/dev/vdi",
            "/dev/vde",
            "/dev/vdc",
            "/dev/vdk"
          ]
        }
      }
    },
    "restapis": [
      "10.0.0.21",
      "10.0.0.22"
    ]
  }
}
```

Remove Rados Gateway from the host

The *Remove Rados Gateway from the host* playbook plugin removes RGW daemon.

Overview

The following table shows general information about the *Remove Rados Gateway from the host* plugin:

Property	Value
ID	`remove_rgw`
Name	Remove Rados Gateway from the host
Required server list	Yes

Parameters and roles

This plugin does not have specific parameters or roles.

Configuration example

The following is an example of the *Remove Rados Gateway from the host* plugin configuration:

```
{
  "global_vars": {
    "cluster": "ceph"
  },
  "inventory": {
    "_meta": {
      "hostvars": {
        "10.0.0.21": {
          "ansible_user": "ansible",
```

```
          "devices": [
            "/dev/vdj",
            "/dev/vdb",
            "/dev/vdd",
            "/dev/vdh",
            "/dev/vdf"
          ],
          "monitor_address": "10.0.0.21",
          "raw_journal_devices": [
            "/dev/vdg",
            "/dev/vdi",
            "/dev/vde",
            "/dev/vdc",
            "/dev/vdk"
          ]
        },
        "10.0.0.22": {
          "ansible_user": "ansible",
          "devices": [
            "/dev/vdj",
            "/dev/vdb",
            "/dev/vdd",
            "/dev/vdh",
            "/dev/vdf"
          ],
          "monitor_address": "10.0.0.22",
          "raw_journal_devices": [
            "/dev/vdg",
            "/dev/vdi",
            "/dev/vde",
            "/dev/vdc",
            "/dev/vdk"
          ]
        }
      }
    },
    "rgws": [
      "10.0.0.21",
      "10.0.0.22"
    ]
  }
}
```

Restart Services

This plugin restarts services on Ceph nodes. The main idea of this plugin is to make restart zero downtime: after each restart we verify that cluster converges and everything is up and running.

Overview

The following table shows general information about the *Restart Services* plugin:

Property	Value
ID	`restart_services`
Name	Restart Services
Required server list	Yes

Parameters and roles

`restart_osd` Restart OSDs on selected nodes.

`restart_mon` Restart monitors on selected nodes.

`restart_rgw` Restart RGWs on selected nodes.

`restart_restapi` Restart REST API on selected nodes.

Configuration example

The following is an example of the *Remove Rados Gateway from the host* plugin configuration:

```
{
  "global_vars": {
    "cluster": "ceph",
    "mon": {
      "retry_attempts": 10,
      "retry_delay": 10,
      "startup_wait": 60
    },
    "osd": {
      "retry_attempts": 20,
      "retry_delay": 10
    },
    "radosgw": {
      "startup_wait": 60
    },
    "restapi": {
      "startup_wait": 10
    }
  },
  "inventory": {
    "_meta": {
      "hostvars": {
        "10.0.0.20": {
          "ansible_user": "ansible"
        },
        "10.0.0.21": {
          "ansible_user": "ansible"
        },
        "10.0.0.22": {
          "ansible_user": "ansible"
        }
      }
    },
    "mons": [
      "10.0.0.20"
    ],
    "osds": [
      "10.0.0.21",
```

```
      "10.0.0.22"
    ]
  }
}
```

Telegraf integration

The *Telegraf integration* playbook plugin activates the Ceph metrics in Telegraf. These metrics can be sent to Prometheus, InfluxDB, or any other endpoint.

Overview

The following table shows general information about the *Telegraf integration* plugin:

Property	Value
ID	`telegraf_integration`
Name	Telegraf Integration
Required Server List	Yes

The plugin uses a standalone Ansible role from Ansible Galaxy. The following table shows the versions mapping:

Plugin version	Ansible Galaxy version
>=0.2,<0.3	dj-wasabi.telegraf 0.7.0

Configuration example

The following is an example of the *Telegraf integration* plugin configuration:

```
{
  "global_vars": {
    "ceph_binary": "/usr/bin/ceph",
    "ceph_config": "/etc/ceph/ceph.conf",
    "ceph_user": "client.admin",
    "configpath": "/etc/telegraf/telegraf.conf",
    "gather_admin_socket_stats": true,
    "gather_cluster_stats": true,
    "install": true,
    "interval": "1m",
    "mon_prefix": "ceph-mon",
    "osd_prefix": "ceph-osd",
    "socket_dir": "/var/run/ceph",
    "socket_suffix": "asock",
    "telegraf_agent_collection_jitter": 0,
    "telegraf_agent_deb_url": "https://dl.influxdata.com/telegraf/releases/telegraf_1.
↪1.2_amd64.deb",
    "telegraf_agent_debug": false,
    "telegraf_agent_flush_interval": 10,
    "telegraf_agent_flush_jitter": 0,
    "telegraf_agent_interval": 10,
    "telegraf_agent_logfile": "",
    "telegraf_agent_metric_batch_size": 1000,
    "telegraf_agent_metric_buffer_limit": 10000,
    "telegraf_agent_omit_hostname": false,
    "telegraf_agent_output": [
```

```
      {
        "config": [
          "urls = [\"http://localhost:8086\"]",
          "database = \"telegraf\"",
          "precision = \"s\""
        ],
        "type": "influxdb"
      }
    ],
    "telegraf_agent_quiet": false,
    "telegraf_agent_round_interval": true,
    "telegraf_agent_tags": {},
    "telegraf_agent_version": "1.1.2",
    "telegraf_plugins_default": [
      {
        "config": [
          "percpu = true"
        ],
        "plugin": "cpu"
      },
      {
        "plugin": "disk"
      },
      {
        "plugin": "io"
      },
      {
        "plugin": "mem"
      },
      {
        "plugin": "net"
      },
      {
        "plugin": "system"
      },
      {
        "plugin": "swap"
      },
      {
        "plugin": "netstat"
      }
    ],
    "telegraf_plugins_extra": {}
  },
  "inventory": {
    "_meta": {
      "hostvars": {
        "10.0.0.20": {
          "ansible_user": "ansible"
        }
      }
    },
    "telegraf": [
      "10.0.0.20"
    ]
  }
}
```

See also:

- Ceph storage input plugin
- Telegraf source for Ceph storage

Update Ceph Configuration

This plugin allows to update Ceph configuration, propagating settings to */etc/ceph/{{ cluster }}.conf*.

This will not restart services, please use *Restart Services* for that puprose.

Overview

The following table shows general information about the *Update Ceph Configuration* plugin:

Property	Value
ID	`update_ceph_configuration`
Name	Update Ceph Configuration
Required server list	Yes

Parameters and roles

`ceph_conf_overrides` This parameter redefines some options which are propagated in ceph cluster configuration (e.g. */etc/ceph/ceph.conf*). This is a mapping where keys are sections in config file and values are mappings between options and their actual values.

Example

```
[global]
journal size = 512

[some other section]
option1 = 1
option2 = yes
```

The relevant `ceph_conf_overrides` is:

```
{
    "global": {
        "journal size": 512
    },
    "some other section": {
        "option1": "1",
        "option2": "yes"
    }
}
```

Configuration example

The following is an example of the *Update Ceph Configuration* plugin configuration:

```
{
  "global_vars": {
    "ceph_conf_overrides": {}
  },
  "inventory": {
    "_meta": {
      "hostvars": {
        "10.0.0.20": {
          "ansible_user": "ansible",
          "devices": [
            "/dev/vdj",
            "/dev/vdb",
            "/dev/vdd",
            "/dev/vdh",
            "/dev/vdf"
          ],
          "monitor_address": "10.0.0.20",
          "raw_journal_devices": [
            "/dev/vdg",
            "/dev/vdi",
            "/dev/vde",
            "/dev/vdc",
            "/dev/vdk"
          ]
        },
        "10.0.0.21": {
          "ansible_user": "ansible",
          "devices": [
            "/dev/vdj",
            "/dev/vdb",
            "/dev/vdd",
            "/dev/vdh",
            "/dev/vdf"
          ],
          "monitor_address": "10.0.0.21",
          "raw_journal_devices": [
            "/dev/vdg",
            "/dev/vdi",
            "/dev/vde",
            "/dev/vdc",
            "/dev/vdk"
          ]
        },
        "10.0.0.22": {
          "ansible_user": "ansible",
          "devices": [
            "/dev/vdj",
            "/dev/vdb",
            "/dev/vdd",
            "/dev/vdh",
            "/dev/vdf"
          ],
          "monitor_address": "10.0.0.22",
          "raw_journal_devices": [
            "/dev/vdg",
            "/dev/vdi",
            "/dev/vde",
            "/dev/vdc",
```

```
            "/dev/vdk"
          ]
        }
      }
    },
    "mons": [
      "10.0.0.20"
    ],
    "osds": [
      "10.0.0.21",
      "10.0.0.22"
    ]
  }
}
```

Upgrade Ceph cluster

This plugin allows to safely upgrade existing Ceph cluster version without downtime. Also, this plugin allows to check upgrade possibility and prevent upgrading if cluster has inconsistent package version (also it checks version of running services).

It upgrades cluster by rolling model: 1 service by one, trying to keep 0 downtime.

Overview

The following table shows general information about the *Upgrade Ceph Cluster* plugin:

Property	Value
ID	`upgrade_ceph`
Name	Upgrade Ceph
Required server list	Yes

Parameters and roles

`sync_time` Synchronize time using the same NTP server for all nodes before starting upgrade.

Configuration example

The following is an example of the *Upgrade Ceph Cluster* plugin configuration:

```
{
  "global_vars": {
    "cluster": "ceph",
    "do_timesync": false,
    "mon": {
      "restart_attempts": 10,
      "restart_delay": 10
    },
    "ntp_server": "pool.ntp.org",
    "osd": {
      "restart_attempts": 10,
      "restart_delay": 60
    }
```

```
  },
  "inventory": {
    "_meta": {
      "hostvars": {
        "10.0.0.20": {
          "ansible_user": "ansible",
          "has_radosgw": false
        },
        "10.0.0.21": {
          "ansible_user": "ansible",
          "has_radosgw": false
        },
        "10.0.0.22": {
          "ansible_user": "ansible",
          "has_radosgw": false
        }
      }
    },
    "mons": [
      "10.0.0.20"
    ],
    "osds": [
      "10.0.0.21",
      "10.0.0.22"
    ]
  }
}
```

Use 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 includes 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 OS 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:

Ceph cluster deployed by Decapod in *MCP Reference Architecture*

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 \
            sucessfull deployment, 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"
            ],
            "rbdmirrors": [],
            "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"
                ],
                "rbdmirrors": [],
                "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
```

Admin service

Important: The Admin service must be used only by experienced users and administrators.

Along with ordinary Decapod Docker containers, `docker-compose` runs an optional but strongly recommended service called the *admin* service. This service provides special containers that act like a lightweight virtual machine with configured *command-line interface* and a **`decapod-admin`** tool that performs maintenance of low-level operations on Decapod or cluster.

This service has a number of additional utilities, such as `vim`, `nano`, `less`, `jq`, `yaql`, `jmespath-terminal`, and `jp`. Vim is configured as a default editor. Basically, it means that you can execute **`decapod`** from a container as is.

```
root@7252bfd5947d:/# decapod user get-all
[
    {
        "data": {
            "email": "noreply@example.com",
            "full_name": "Root User",
            "login": "root",
            "role_id": "e6ba587a-6256-401a-8734-8cead3d7a4c7"
        },
        "id": "7a52f762-7c2d-4164-b779-15f86f4aef2a",
        "initiator_id": null,
        "model": "user",
        "time_deleted": 0,
        "time_updated": 1487146111,
        "version": 1
    }
]
root@7252bfd5947d:/# decapod user get-all | jp '[0].id'
"7a52f762-7c2d-4164-b779-15f86f4aef2a"
root@7252bfd5947d:/# decapod user get-all | jq -r '.[0]|.id'
7a52f762-7c2d-4164-b779-15f86f4aef2a
```

The admin service runs Cron jobs that perform Keystone synchronization, *monitoring*, and data collection.

Additionally, the Decapod admin service enables various maintenance and admin tasks as described in the following topics.

Access the admin service

To access the Decapod admin service:

```
$ docker-compose -p PROJECT_NAME exec admin bash
```

Note: The `-p` option is the name of the project. If you have not specified it when running `docker-compose`, do not specify it now.

As a result, you will enter the container. The default environment allows you to run the **`decapod`** utility with a configured URL and login/password pair `root`/`root`.

```
root@7252bfd5947d:/# env | grep DECAPOD
DECAPOD_PASSWORD=root
DECAPOD_LOGIN=root
DECAPOD_URL=http://frontend:80
```

The `admin` service provides bundled documentation within the container. To access documentation:

1. Obtain the documentation port. The port is the value of the `DECAPOD_DOCS_PORT` environment variable and is `9998` by default.
2. Access the documentation using the obtained port and your credentials. For example, if you access Decapod using `http://10.0.0.10:9999`, the documentation will be served on `http://10.0.0.10:9998`.

Apply migrations

Migrations in Decapod are similar to migrations in databases but affect not only the schema but also the data. The main idea of such a migration is to adapt the existing data to a newer version of Decapod. For all available commands and options related to migrations, run **`decapod-admin migration --help`**.

To get a list of migrations, run **`decapod-admin migration list all`**.

Example:

```
root@7252bfd5947d:/# decapod-admin migration list all
[applied]     0000_index_models.py
[applied]     0001_insert_default_role.py
[applied]     0002_insert_default_user.py
[applied]     0003_native_ttl_index.py
[applied]     0004_migrate_to_native_ttls.py
[applied]     0005_index_cluster_data.py
[applied]     0006_create_cluster_data.py
[applied]     0007_add_external_id_to_user.py
```

You can apply migrations at any time. Decapod tracks migrations that have already been applied. To apply migrations, run **`decapod-admin migration apply`** or **`decapod-admin migration apply MIGRATION_NAME`** to apply a particular migration.

Example:

```
root@7252bfd5947d:/# decapod-admin migration apply
2017-02-15 10:19:25 [DEBUG   ] (        lock.py:118 ): Lock \
applying_migrations was acquire by locker 071df271-d0ba-4fdc-83d0-49575d0acf3c
2017-02-15 10:19:25 [DEBUG   ] (        lock.py:183 ): Prolong thread for \
locker applying_migrations of lock 071df271-d0ba-4fdc-83d0-49575d0acf3c \
has been started. Thread MongoLock prolonger \
071df271-d0ba-4fdc-83d0-49575d0acf3c for applying_migrations, ident 140625762334464
2017-02-15 10:19:25 [INFO    ] (   migration.py:119 ): No migration are \
required to be applied.
2017-02-15 10:19:25 [DEBUG   ] (        lock.py:202 ): Prolong thread for \
locker applying_migrations of lock 071df271-d0ba-4fdc-83d0-49575d0acf3c \
has been stopped. Thread MongoLock prolonger \
071df271-d0ba-4fdc-83d0-49575d0acf3c for applying_migrations, ident 140625762334464
2017-02-15 10:19:25 [DEBUG   ] (        lock.py:124 ): Try to release lock \
applying_migrations by locker 071df271-d0ba-4fdc-83d0-49575d0acf3c.
2017-02-15 10:19:25 [DEBUG   ] (        lock.py:140 ): Lock \
applying_migrations was released by locker 071df271-d0ba-4fdc-83d0-49575d0acf3c.
```

To show details on a migration, run **`decapod-admin migration show MIGRATION_NAME`**.

Example:

```
root@7252bfd5947d:/# decapod-admin migration show 0006_create_cluster_data.py
Name:           0006_create_cluster_data.py
Result:         ok
Executed at:    Wed Feb 15 08:08:36 2017
SHA1 of script: 73eb7adeb1b4d82dd8f9bdb5aadddccbcef4a8b3

-- Stdout:
Migrate 0 clusters.

-- Stderr:
```

Generate cloud-init user data configuration

You can generate user data configuration for `cloud-init` as described in *Generate user data*. Alternatively, use the `decapod-admin`.

To generate user data configuration using the `decapod-admin` tool, run **`decapod-admin cloud-config [OPTIONS] PUBLIC_URL`** specifying the URL accessible by Ceph nodes. For all available options, run **`decapod-admin cloud-config --help`**.

Example:

```
root@7252bfd5947d:/# decapod-admin cloud-config http://10.0.0.10:9999
#cloud-config
packages: [python]
runcmd:
- [echo, === START DECAPOD SERVER DISCOVERY ===]
- [sh, -xc, 'grep -q ''/usr/share/server_discovery.sh'' /etc/rc.local || \
sed -i ''s?^exit 0?/usr/share/server_discovery.sh >> /var/log/\
server_discovery.log 2>\&1\nexit 0?'' /etc/rc.local']
- [sh, -xc, systemctl enable rc-local.service || true]
- [sh, -xc, /usr/share/server_discovery.sh 2>&1 | tee -a /var/log/\
server_discovery.log]
- [echo, === FINISH DECAPOD SERVER DISCOVERY ===]
users:
- groups: [sudo]
  name: ansible
  shell: /bin/bash
  ssh-authorized-keys: [ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAACAQC7K9bHPrSu5V\
  HnUOis2Uwc822fMyPTtwjfOkzNi/\
  oVOxmd1QE3DilrO5fJ33pRwEj7r1DfTlJmZWs8XwWaaUXkQ+iyfRPtgt/Ox+X5A/XaLdi/\
  yz7UjnHc8ERDUT/z73RzDwf21KNQOopGRuyhe+gvGZ5mhYDz3bnnYY9IRBNYaGw4bjS0q1\
  AbkPa1PvCo7P5b5UuRjhi4H74zCFkQD4evQsrQOcgev5GimnODqMntU0jnI/eEJwnnd1TcY\
  G7dS6FqMWpFX1gqcKjFIuqNTZLYzJu9U8mxxKmGOQSI6KWfP0etBw1YRHRIfdZmdaqSKHh0\
  ZhUUHjbf8Hb5Vqv1Fkzf0cGPbfrazEDI5FaVjkZMGFfdgs1be6xO7NHqzu1JJ3ZEur28o0A\
  QyOVvrEJIxQayDM0qyKi7B4+j6QDL0CDaWN3dUZO45il/KOm/eXCm4yQg0ImXHUmsDoW+6W\
  6akI/fSCAn8r9GK2QBBJPeTPA95WlOSXtICnrsqgb74yKPEsslzfrTUIiyoXBuuR9o5OoPX\
  ghKrazqcTeK/Vdl7w4nZ00O4jllHMTrS1xyubN0QeBd+3D8Hy2bN5h7WjiJsZ2XhlKR0Z1i5\
  AbgCR9hfQl84aFIXRARz+6uuDDHe2ONXujcS9jhuN7SOLGckiaXNfAeAsbEkYZytnUgdoxbHYSfzw==]
  sudo: ['ALL=(ALL) NOPASSWD:ALL']
write_files:
- content: |
    #-*- coding: utf-8 -*-

    from __future__ import print_function
```

```
    import json
    import ssl
    import sys

    try:
        import urllib.request as urllib2
    except ImportError:
        import urllib2

    data = {
        "username": 'ansible',
        "host": sys.argv[1].lower().strip(),
        "id": sys.argv[2].lower().strip()
    }
    headers = {
        "Content-Type": "application/json",
        "Authorization": '26758c32-3421-4f3d-9603-e4b5337e7ecc',
        "User-Agent": "cloud-init server discovery"
    }

    def get_response(url, data=None):
        if data is not None:
            data = json.dumps(data).encode("utf-8")
        request = urllib2.Request(url, data=data, headers=headers)
        request_kwargs = {"timeout": 20}
        if sys.version_info >= (2, 7, 9):
            ctx = ssl.create_default_context()
            ctx.check_hostname = False
            ctx.verify_mode = ssl.CERT_NONE
            request_kwargs["context"] = ctx
        try:
            return urllib2.urlopen(request, **request_kwargs).read()
        except Exception as exc:
            print("Cannot request {0}: {1}".format(url, exc))

    metadata_ip = get_response('http://169.254.169.254/latest/meta-data/public-ipv4')
    if metadata_ip is not None:
        data["host"] = metadata_ip
        print("Use IP {0} discovered from metadata API".format(metadata_ip))

    response = get_response('http://10.0.0.10:9999', data)
    if response is None:
        sys.exit("Server discovery failed.")
    print("Server discovery completed.")
  path: /usr/share/server_discovery.py
  permissions: '0440'
- content: |
    #!/bin/bash
    set -xe -o pipefail

    echo "Date $(date) | $(date -u) | $(date '+%s')"

    main() {
        local ip="$(get_local_ip)"
        local hostid="$(get_local_hostid)"

        python /usr/share/server_discovery.py "$ip" "$hostid"
```

```
    }

    get_local_ip() {
        local remote_ipaddr="$(getent ahostsv4 "10.0.0.10" | head -n 1 | cut -f 1 -d
↪' ')"

        ip route get "$remote_ipaddr" | head -n 1 | rev | cut -d ' ' -f 2 | rev
    }

    get_local_hostid() {
        dmidecode | grep UUID | rev | cut -d ' ' -f 1 | rev
    }

    main
  path: /usr/share/server_discovery.sh
  permissions: '0550'
```

Back up and restore database

Using the `decapod-admin` tool, you can back up and restore MongoDB, the main storage system used by Decapod. The archive format created by this tool is a native MongoDB archive that is compressed by default.

The output of **`decapod-admin db backup`** and **`decapod-admin db restore`** is similar to the output of **`mongodump --archive --gzip`** and **`mongorestore --archive --gzip`**. The `decapod-admin` tool uses `/etc/decapod/config.yaml` to read Decapod MongoDB settings and correctly constructs the command line taking the SSL settings into account. To get a list of available commands and options, run **`decapod-admin db --help`**.

To back up the database:

```
$ decapod-admin db backup > backupfile
```

To restore the database:

```
$ decapod-admin db restore < backupfile
```

If you do not require compression, use the `-r` flag. In such case, `mongodump` and `mongorestore` will not use the `--gzip` flag.

See also:

Archiving and compression in MongoDB tools

SSH to Ceph hosts

Using the `decapod-admin` tool, you can SSH to remote hosts with the same user as used by Ansible.

To SSH to a remote host, use the **`decapod-admin ssh server-ip SERVER_IP`** or **`decapod-admin ssh server-id SERVER_ID`** command.

Example:

```
root@7252bfd5947d:/# decapod-admin ssh server-id 8cf8af12-89a0-477d-85e7-ce6cbe5f8a07
2017-02-15 09:42:40 [DEBUG   ] (         ssh.py:111 ): Execute \
['/usr/bin/ssh', '-4', '-tt', '-x', '-o', 'UserKnownHostsFile=/dev/null', \
'-o', 'StrictHostKeyChecking=no', '-l', 'ansible', '-i', \
```

```
'/root/.ssh/id_rsa', '10.0.0.23']
Warning: Permanently added '10.0.0.23' (ECDSA) to the list of known hosts.
Welcome to Ubuntu 16.04 LTS (GNU/Linux 4.4.0-22-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

171 packages can be updated.
73 updates are security updates.


Last login: Wed Feb 15 09:30:45 2017 from 10.0.0.10
ansible@ceph-node04:~$ whoami
ansible
```

For all available options, run **`decapod-admin ssh --help`**.

Execute SSH in parallel

You may need to execute commands on remote hosts in parallel. For such purposes, `decapod-admin` uses its own implementation of pdsh integrated with Decapod.

Using **`decapod-admin pdsh`**, you can execute commands on multiple hosts in parallel, upload files and download them from remote hosts. For details on **`decapod-admin pdsh`** usage and all available commands and options, run **`decapod-admin pdsh --help`**.

Examples

```
root@7252bfd5947d:/# decapod-admin pdsh exec -- ls -la
9f01297e-e6fb-4d9f-ae96-09d4fcb8e1f5 | 10.0.0.21      \
: total 32
9f01297e-e6fb-4d9f-ae96-09d4fcb8e1f5 | 10.0.0.21      \
: drwxr-xr-x 5 ansible ansible 4096 Feb 15 09:22 .
9f01297e-e6fb-4d9f-ae96-09d4fcb8e1f5 | 10.0.0.21      \
: drwxr-xr-x 4 root    root    4096 Feb 15 08:48 ..
9f01297e-e6fb-4d9f-ae96-09d4fcb8e1f5 | 10.0.0.21      \
: drwx------ 3 ansible ansible 4096 Feb 15 09:22 .ansible
9f01297e-e6fb-4d9f-ae96-09d4fcb8e1f5 | 10.0.0.21      \
: -rw-r--r-- 1 ansible ansible  220 Aug 31  2015 .bash_logout
9f01297e-e6fb-4d9f-ae96-09d4fcb8e1f5 | 10.0.0.21      \
: -rw-r--r-- 1 ansible ansible 3771 Aug 31  2015 .bashrc
9f01297e-e6fb-4d9f-ae96-09d4fcb8e1f5 | 10.0.0.21      \
: drwx------ 2 ansible ansible 4096 Feb 15 09:22 .cache
9f01297e-e6fb-4d9f-ae96-09d4fcb8e1f5 | 10.0.0.21      \
: -rw-r--r-- 1 ansible ansible  675 Aug 31  2015 .profile
9f01297e-e6fb-4d9f-ae96-09d4fcb8e1f5 | 10.0.0.21      \
: drwx------ 2 ansible ansible 4096 Feb 15 08:49 .ssh
9f01297e-e6fb-4d9f-ae96-09d4fcb8e1f5 | 10.0.0.21      \
: -rw-r--r-- 1 ansible ansible    0 Feb 15 09:22 .sudo_as_admin_successful
62adf9cb-3f2d-4ea6-94f5-bca3aebfdb93 | 10.0.0.22      : total 32
62adf9cb-3f2d-4ea6-94f5-bca3aebfdb93 | 10.0.0.22      \
: drwxr-xr-x 5 ansible ansible 4096 Feb 15 10:40 .
62adf9cb-3f2d-4ea6-94f5-bca3aebfdb93 | 10.0.0.22      \
: drwxr-xr-x 4 root    root    4096 Feb 15 08:48 ..
62adf9cb-3f2d-4ea6-94f5-bca3aebfdb93 | 10.0.0.22      \
: drwx------ 3 ansible ansible 4096 Feb 15 09:22 .ansible
62adf9cb-3f2d-4ea6-94f5-bca3aebfdb93 | 10.0.0.22      \
: -rw-r--r-- 1 ansible ansible  220 Aug 31  2015 .bash_logout
```

```
62adf9cb-3f2d-4ea6-94f5-bca3aebfdb93 | 10.0.0.22      \
: -rw-r--r-- 1 ansible ansible 3771 Aug 31  2015 .bashrc
62adf9cb-3f2d-4ea6-94f5-bca3aebfdb93 | 10.0.0.22      \
: drwx------ 2 ansible ansible 4096 Feb 15 09:22 .cache
62adf9cb-3f2d-4ea6-94f5-bca3aebfdb93 | 10.0.0.22      \
: -rw-r--r-- 1 ansible ansible  675 Aug 31  2015 .profile
62adf9cb-3f2d-4ea6-94f5-bca3aebfdb93 | 10.0.0.22      \
: drwx------ 2 ansible ansible 4096 Feb 15 08:49 .ssh
62adf9cb-3f2d-4ea6-94f5-bca3aebfdb93 | 10.0.0.22      \
: -rw-r--r-- 1 ansible ansible    0 Feb 15 09:22 .sudo_as_admin_successful
8cf8af12-89a0-477d-85e7-ce6cbe5f8a07 | 10.0.0.23      \
: total 36
8cf8af12-89a0-477d-85e7-ce6cbe5f8a07 | 10.0.0.23      \
: drwxr-xr-x 5 ansible ansible 4096 Feb 15 10:00 .
8cf8af12-89a0-477d-85e7-ce6cbe5f8a07 | 10.0.0.23      \
: drwxr-xr-x 4 root    root    4096 Feb 15 08:48 ..
8cf8af12-89a0-477d-85e7-ce6cbe5f8a07 | 10.0.0.23      \
: drwx------ 3 ansible ansible 4096 Feb 15 09:22 .ansible
8cf8af12-89a0-477d-85e7-ce6cbe5f8a07 | 10.0.0.23      \
: -rw------- 1 ansible ansible    7 Feb 15 09:43 .bash_history
8cf8af12-89a0-477d-85e7-ce6cbe5f8a07 | 10.0.0.23      \
: -rw-r--r-- 1 ansible ansible  220 Aug 31  2015 .bash_logout
8cf8af12-89a0-477d-85e7-ce6cbe5f8a07 | 10.0.0.23      \
: -rw-r--r-- 1 ansible ansible 3771 Aug 31  2015 .bashrc
8cf8af12-89a0-477d-85e7-ce6cbe5f8a07 | 10.0.0.23      \
: drwx------ 2 ansible ansible 4096 Feb 15 09:22 .cache
8cf8af12-89a0-477d-85e7-ce6cbe5f8a07 | 10.0.0.23      \
: -rw-r--r-- 1 ansible ansible  675 Aug 31  2015 .profile
8cf8af12-89a0-477d-85e7-ce6cbe5f8a07 | 10.0.0.23      \
: drwx------ 2 ansible ansible 4096 Feb 15 08:49 .ssh
8cf8af12-89a0-477d-85e7-ce6cbe5f8a07 | 10.0.0.23      \
: -rw-r--r-- 1 ansible ansible    0 Feb 15 09:22 .sudo_as_admin_successful
26261da0-2dde-41e9-8ab6-8836c806623e | 10.0.0.20      : total 32
26261da0-2dde-41e9-8ab6-8836c806623e | 10.0.0.20      \
: drwxr-xr-x 5 ansible ansible 4096 Feb 15 10:30 .
26261da0-2dde-41e9-8ab6-8836c806623e | 10.0.0.20      \
: drwxr-xr-x 4 root    root    4096 Feb 15 08:48 ..
26261da0-2dde-41e9-8ab6-8836c806623e | 10.0.0.20      \
: drwx------ 3 ansible ansible 4096 Feb 15 09:22 .ansible
26261da0-2dde-41e9-8ab6-8836c806623e | 10.0.0.20      \
: -rw-r--r-- 1 ansible ansible  220 Aug 31  2015 .bash_logout
26261da0-2dde-41e9-8ab6-8836c806623e | 10.0.0.20      \
: -rw-r--r-- 1 ansible ansible 3771 Aug 31  2015 .bashrc
26261da0-2dde-41e9-8ab6-8836c806623e | 10.0.0.20      \
: drwx------ 2 ansible ansible 4096 Feb 15 09:22 .cache
26261da0-2dde-41e9-8ab6-8836c806623e | 10.0.0.20      \
: -rw-r--r-- 1 ansible ansible  675 Aug 31  2015 .profile
26261da0-2dde-41e9-8ab6-8836c806623e | 10.0.0.20      \
: drwx------ 2 ansible ansible 4096 Feb 15 08:49 .ssh
26261da0-2dde-41e9-8ab6-8836c806623e | 10.0.0.20      \
: -rw-r--r-- 1 ansible ansible    0 Feb 15 09:22 .sudo_as_admin_successful
```

```
root@7252bfd5947d:/# decapod-admin pdsh upload /etc/decapod/config.yaml .
8cf8af12-89a0-477d-85e7-ce6cbe5f8a07 | 10.0.0.23      \
: Start to upload /etc/decapod/config.yaml to .
8cf8af12-89a0-477d-85e7-ce6cbe5f8a07 | 10.0.0.23      \
: Finished uploading of /etc/decapod/config.yaml to .
62adf9cb-3f2d-4ea6-94f5-bca3aebfdb93 | 10.0.0.22      \
```

```
: Start to upload /etc/decapod/config.yaml to .
9f01297e-e6fb-4d9f-ae96-09d4fcb8e1f5 | 10.0.0.21      \
: Start to upload /etc/decapod/config.yaml to .
9f01297e-e6fb-4d9f-ae96-09d4fcb8e1f5 | 10.0.0.21      \
: Finished uploading of /etc/decapod/config.yaml to .
62adf9cb-3f2d-4ea6-94f5-bca3aebfdb93 | 10.0.0.22      \
: Finished uploading of /etc/decapod/config.yaml to .
26261da0-2dde-41e9-8ab6-8836c806623e | 10.0.0.20      \
: Start to upload /etc/decapod/config.yaml to .
26261da0-2dde-41e9-8ab6-8836c806623e | 10.0.0.20      \
: Finished uploading of /etc/decapod/config.yaml to .

root@7252bfd5947d:/# decapod-admin pdsh exec -- ls -lah config.yaml
26261da0-2dde-41e9-8ab6-8836c806623e | 10.0.0.20      \
: -rw-r--r-- 1 ansible ansible 3.0K Feb 15 07:37 config.yaml
9f01297e-e6fb-4d9f-ae96-09d4fcb8e1f5 | 10.0.0.21      \
: -rw-r--r-- 1 ansible ansible 3.0K Feb 15 07:37 config.yaml
62adf9cb-3f2d-4ea6-94f5-bca3aebfdb93 | 10.0.0.22      \
: -rw-r--r-- 1 ansible ansible 3.0K Feb 15 07:37 config.yaml
8cf8af12-89a0-477d-85e7-ce6cbe5f8a07 | 10.0.0.23      \
: -rw-r--r-- 1 ansible ansible 3.0K Feb 15 07:37 config.yaml
```

```
root@7252bfd5947d:/# decapod-admin pdsh download config.yaml results/
9f01297e-e6fb-4d9f-ae96-09d4fcb8e1f5 | 10.0.0.21      \
: Start to download config.yaml to results/9f01297e-e6fb-4d9f-ae96-09d4fcb8e1f5
26261da0-2dde-41e9-8ab6-8836c806623e | 10.0.0.20      \
: Start to download config.yaml to results/26261da0-2dde-41e9-8ab6-8836c806623e
8cf8af12-89a0-477d-85e7-ce6cbe5f8a07 | 10.0.0.23      \
: Start to download config.yaml to results/8cf8af12-89a0-477d-85e7-ce6cbe5f8a07
62adf9cb-3f2d-4ea6-94f5-bca3aebfdb93 | 10.0.0.22      \
: Start to download config.yaml to results/62adf9cb-3f2d-4ea6-94f5-bca3aebfdb93
```

Restore entities

You can restore entities that were explicitly or accidentally deleted, for example, a cluster, user, server, role, and others. To restore a deleted entity, use the **`decapod-admin restore ITEM_TYPE ITEM_ID`** command specifying the type of the entity and its ID.

Example:

```
root@7252bfd5947d:/# decapod-admin restore user 6805075b-e40d-4800-8520-8569dd7327bd
{
    "data": {
        "email": "test@example.com",
        "full_name": "Full",
        "login": "test",
        "role_id": null
    },
    "id": "6805075b-e40d-4800-8520-8569dd7327bd",
    "initiator_id": "7a52f762-7c2d-4164-b779-15f86f4aef2a",
    "model": "user",
    "time_deleted": 1487154755,
    "time_updated": 1487154755,
    "version": 2
}
Undelete item? [y/N]: y
```

```
{
    "data": {
        "email": "test@example.com",
        "full_name": "Full",
        "login": "test",
        "role_id": null
    },
    "id": "6805075b-e40d-4800-8520-8569dd7327bd",
    "initiator_id": "7a52f762-7c2d-4164-b779-15f86f4aef2a",
    "model": "user",
    "time_deleted": 0,
    "time_updated": 1487154769,
    "version": 3
}
```

For command options and entity types, run **`decapod-admin restore -h`**.

Unlock servers

All playbook executions lock the servers they use to eliminate issues caused by a concurrent execution. However, you may be required to manually unlock servers. To do so, use the **`decapod-admin locked-servers`** command.

To list all locked servers:

```
decapod-admin locked-servers get-all
```

To unlock a server:

```
decapod-admin locked-servers unlock SERVER_ID
```

For all available options, run **`decapod-admin locked-servers --help`**.

Reset password

You can reset a user password through the Decapod web UI. However, in some cases you may want to change the password bypassing the usual procedure. For example, if a user has an obsolete, non-working email or if you want to change the default `root`/`root` username and password pair.

To explicitly reset a user password:

```
decapod-admin password-reset [OPTIONS] USER_ID
```

If you do not pass the new password, `decapod-admin` will prompt you to enter it.

Example:

```
$ decapod-admin password-reset c83d0ede-aad1-4f1f-b6f0-730879974763
New password []:
Repeat for confirmation:
```

If you do not pass any password, the tool will generate one and output it to `stdout`.

Example:

```
$ decapod-admin password-reset c83d0ede-aad1-4f1f-b6f0-730879974763
New password []:
54\gE'1Ck_
```

For all available options, run **decapod-admin password-reset -h**.

External Playbook Execution

Sometimes Decapod does not work as expected of fails with errors. This may block endusers on cluster operations: sometimes these cluster operations should be perfomed as soon as possible and you cannot wait for Decapod fix. This command allows user to create tarball with data enough to be executed externally, without Decapod itself.

Important: Since Decapod does not control external execution, it does not know anything about cluster modifications which were performed. So if you've been added new role to the node within a cluster, you need to inject this data into Decapod afterwards. Right now there is no ready functionality to perform such low-level tasks.

To be available to execute externally, you need to have Ansible 2.3 or newer installed.

To create tarball with data execute the following:

```
$ decapod-admin external-execution f5ace5b8-35f4-4d12-9a43-32a8f62edfa9 1
/f5ace5b8-35f4-4d12-9a43-32a8f62edfa9-1.tar.gz
```

Script will output the path to the tarball. If you are using contaierized Decapod installation, you need to copy it out of the container.

```
$ docker cp decapod_admin_1:/f5ace5b8-35f4-4d12-9a43-32a8f62edfa9-1.tar.gz .
```

Extract tarball with **tar**:

```
$ tar xf f5ace5b8-35f4-4d12-9a43-32a8f62edfa9-1.tar.gz
```

To execute ansible externally, please run **execute.sh** script.

Tarball contents

```
$ ls -1
ansible.cfg
ceph-ansible
common_playbooks
decapod_data
execute.sh
fetch_directory
inventory.yaml
plugin
ssh-private-key.pem
```

Here is the description for every file in the tarball:

ansible.cfg Ansible config which is used by Decapod to run playbooks with minor differences. The main difference are plugin paths, they are rewritten to support local execution. Also, paths to the custom Decapod plugins are excluded because local installation does not have it (for example, there is no need to keep execution log in database).

ceph-ansible Directory with version of ceph-ansible used by current installation. So, there is no need to clone it from GitHub.

common_playbooks Different playbooks which are included into Decapod playbook plugins.

decapod_data Decapod models: playbook configuration, cluster etc. This is not required for execution, it is placed here just to help with debug.

execute.sh Script which runs Ansible. This script executed playbook plugin for configuration which was created.

fetch_directory ceph-ansible requires to have special directory on local machine where it stores different data (keyring etc).

inventory.yaml Inventory file for Ansible.

plugin Contents of the Decapod plugin. This is required because a lot of plugins have their data which is involved into execution.

ssh-private-key.pem Private SSH key to access Ceph node. This is the Decapod private key so please not distribute it.

See also:

- jq
- YAQL
- JMESPath terminal
- jp

Monitor Ceph

Decapod supports integration with various monitoring systems. By default, it uses an in-house tool called ceph-monitoring. You can also integrate Decapod with other tools, such as Prometheus through Telegraf. For a list of supported plugins, see *Playbook plugins*.

The `ceph-monitoring` tool collects statistics on the cluster state and monitors its performance by running particular scripts under the `admin` service.

To access the collected data:

1. Obtain the Decapod monitoring port. The port is the value of the `DECAPOD_MONITORING_PORT` environment variable and is `10001` by default.
2. Access the data using the obtained port and your credentials. For example, if you access Decapod using `http://10.0.0.10:9999`, the data will be served on `http://10.0.0.10:10001`.

If there is no information available about a recently deployed cluster, try again in 15 minutes. If the data is still not accessible, obtain the logs of the `admin` service:

```
$ docker-compose -p myprojectname logs admin
```

Generate a diagnostic snapshot

To simplify the interaction between development and operation, Decapod supports diagnostic or debug snapshots, similar to Fuel snapshots. A snapshot is an archive that contains all information required to debug and troubleshoot issues.

Snapshots store the following information:

- Backup of the database

- Logs from services
- File `docker-compose.yml`
- Configuration files from Decapod services (`config.yaml`)
- Datetimes from services
- Data from *ceph-monitoring*
- Version of installed packages
- Git commit SHAs of Decapod itself
- Information about docker and containers

Snapshots do not store Ansible private keys or user passwords. Passwords are hashed by Argon2.

To generate a diagnostic snapshot:

Run the script:

```
$ ./scripts/debug_snapshot.py snapshot
```

Alternatively, if you have containers only, follow the steps below.

1. Run the following command:

```
$ docker-compose exec -T admin cat /debug-snapshot | python - snapshot
```

2. Configure the snapshot settings as required:

```
$ docker-compose -p myproject exec -T admin cat /debug-snapshot | python - --help
usage: - [-h] [-f COMPOSE_FILE] [-p PROJECT_NAME] snapshot_path

Create a debug snapshot for Decapod.

positional arguments:
  snapshot_path         Path where to store snapshot (do not append extension,
                        we will do it for you).

optional arguments:
  -h, --help            show this help message and exit
  -f COMPOSE_FILE, --compose-file COMPOSE_FILE
                        path to docker-compose.yml file. (default:
                        /vagrant/docker-compose.yml)
  -p PROJECT_NAME, --project-name PROJECT_NAME
                        the name of the project. (default: vagrant)

Please find all logs in syslog by ident 'decapod-debug-snapshot'.

$ docker-compose -p myproject exec -T admin cat /debug-snapshot | python - -p␣
↪myproject snapshot
```

As a result, you will get a snapshot like `snapshot_path.*`. The snapshot tool calculates the best compression algorithm available on your platform and uses its extension. Therefore, the snapshot may look like `snapshot_path.tar.bz2` or `snapshot_path.tar.xz` depending on how your Python was built.

Upgrade Decapod

Before starting any upgrade procedure, perform a backup of Decapod. The upgrade procedure is consequent, which means before upgrading to the latest version, you must first upgrade to the next version. For example, to upgrade Decapod from version *X* to *X+3*, first upgrade to *X+1*, then to *X+2*, and only then to *X+3*.

Verify Decapod version

To verify the Decapod version installed:

1. Obtain the project name:

```
$ docker-compose ps | grep api | cut -f 1 -d '_' | sort -u
shrimp
```

2. Verify the Decapod version:

```
docker inspect -f '{{ .Config.Labels.version }}' $(docker-compose -p \
PROJ ps -q database | head -n 1)
```

   Where `PROJ` is the project name.

   Example:

```
docker inspect -f '{{ .Config.Labels.version }}' $(docker-compose -p \
PROJ ps -q database | head -n 1)
0.1.0
```

Upgrade Decapod from 0.1.x to 1.0

Prior to upgrading Decapod, perform the steps described in *Prerequisites*.

Note: You do not need to perform any changes in the existing Ceph deployments.

Prerequisites

Prior to upgrading Decapod from 0.1.x to 1.0.0, verify that you have completed the following tasks:

1. From the machine that runs Decapod, obtain the latest versions of Decapod 1.0 release series:

```
$ git clone -b stable-1.0 --recurse-submodules https://github.com/Mirantis/ceph-
↪lcm.git ~/decapod
```

2. Create a directory to store configuration files and private keys for Decapod:

```
$ mkdir -p ~/decapod_runtime
```

3. Obtain the project name:

```
$ docker-compose ps | grep api | cut -f 1 -d '_' | sort -u
shrimp
```

For simplicity, further examples use `PROJ` as the project name.

Note: To avoid passing `-p` each time you run `docker-compose`, use the `COMPOSE_PROJECT_NAME` environment variable.

---

4. Copy the required configuration files to the `~/decapod_runtime` directory:

```
$ cp ~/decapod/{.env,docker-compose.yml,docker-compose.override.yml} ~/decapod_
↪runtime
```

5. Set the path to the SSH private key in the `.env` file:

```
$ sed -i "s?^DECAPOD_SSH_PRIVATE_KEY=.*?DECAPOD_SSH_PRIVATE_KEY=$HOME/\
decapod_runtime/id_rsa?" ~/decapod_runtime/.env
```

Use the name of your private key if it differs from the `id_rsa` in the example above.

Upgrade Decapod

To upgrade Decapod from 0.1.x to 1.0.0:

1. Back up the database:
   - To use the existing configuration, run the following command from the directory where you run Decapod:

```
$ docker exec -i proj_database_1 mongodump --gzip --archive --ssl \
--sslAllowInvalidCertificates > ~/pre_upgrade
```

Where `proj` is the lowercase container name.

Note: To restore the database:

```
$ docker exec -i proj_database_1 mongorestore --drop --gzip \
--archive --ssl --sslAllowInvalidCertificates < ~/pre_upgrade
```

   - To use the default configuration, rename the database in MongoDB from `shrimp` to `decapod` and back up the data.

     (a) Rename the database:

```
$ docker-compose -p PROJ exec database moshell
MongoDB shell version: 3.2.10
connecting to: false
2017-02-14T06:38:15.400+0000 W NETWORK  [thread1] The server \
certificate does not match the host name 127.0.0.1
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
        http://docs.mongodb.org/
Questions? Try the support group
        http://groups.google.com/group/mongodb-user
Server has startup warnings:
2017-02-14T06:20:54.806+0000 I CONTROL  [initandlisten]
2017-02-14T06:20:54.806+0000 I CONTROL  [initandlisten] ** WARNING:\
 /sys/kernel/mm/transparent_hugepage/enabled is 'always'.
```

```
2017-02-14T06:20:54.806+0000 I CONTROL  [initandlisten] **        \
We suggest setting it to 'never'
2017-02-14T06:20:54.806+0000 I CONTROL  [initandlisten]
2017-02-14T06:20:54.806+0000 I CONTROL  [initandlisten] ** WARNING:\
 /sys/kernel/mm/transparent_hugepage/defrag is 'always'.
2017-02-14T06:20:54.806+0000 I CONTROL  [initandlisten] **        \
We suggest setting it to 'never'
2017-02-14T06:20:54.806+0000 I CONTROL  [initandlisten]
> db.copyDatabase("shrimp", "decapod", "localhost")
{ "ok" : 1 }
> use shrimp
switched to db shrimp
> db.dropDatabase()
{ "dropped" : "shrimp", "ok" : 1 }
```

(b) Back up the database:

```
$ docker exec -i proj_database_1 mongodump --gzip --archive --ssl \
--sslAllowInvalidCertificates > ~/pre_upgrade_renamed
```

2. Optional. If you have modified any configuration files such as `config.yaml` or `id_rsa`, copy the files to a custom directory, for example, *~/decapod_runtime*. To do so, run the following commands from the same directory used to run Decapod 0.1:

```
$ mkdir ~/decapod_runtime
$ docker cp "$(docker-compose -p PROJ ps -q api):/etc/shrimp/config.yaml" ~/
↪decapod_runtime
$ docker cp "$(docker-compose -p PROJ ps -q controller):/root/.ssh/id_rsa" ~/
↪decapod_runtime
$ docker cp "$(docker-compose -p PROJ ps -q frontend):/ssl/dhparam.pem" ~/decapod_
↪runtime
$ docker cp "$(docker-compose -p PROJ ps -q frontend):/ssl/ssl.crt" ~/decapod_
↪runtime
$ docker cp "$(docker-compose -p PROJ ps -q frontend):/ssl/ssl.key" ~/decapod_
↪runtime
$ docker cp "$(docker-compose -p PROJ ps -q database):/certs/mongodb.pem" ~/
↪decapod_runtime
$ docker cp "$(docker-compose -p PROJ ps -q database):/certs/mongodb-ca.crt" ~/
↪decapod_runtime
```

Note: If you did not generate any custom configuration files and used the default configuration, skip this step and proceed to step 4.

---

3. Obtain the Decapod 1.0.0 images. To do so, follow steps 1-2 in the *Install Decapod* section of *MCP Deployment Guide*.

Note: The required configuration files are stored in `~/decapod_runtime` and the repository for Decapod 1.0.0 is cloned to `~/decapod` as described in *Prerequisites*.

---

4. Stop and remove containers for version 0.1.x. Since Docker containers are stateless and you have created a backup of the state (the database backup), drop the existing containers and start new ones. Execute the following command from the directory where you run Decapod:

```
$ docker-compose -p PROJ down -v
```

5. Run Decapod 1.0.0.

   (a) Change the directory to `~/decapod_runtime`.

   (b) Run Decapod:

   ```
   $ docker-compose -p PROJ up --remove-orphans -d
   ```

6. Restore the database:

```
$ docker exec -i $(docker-compose -p PROJ ps -q admin) decapod-admin db \
restore < ~/pre_upgrade_renamed
```

Alternatively, if you did not rename the database:

```
$ docker exec -i (docker-compose -p PROJ ps admin) decapod-admin db restore < ~/
↪pre_upgrade
```

7. Apply migrations:

```
$ docker-compose -p PROJ exec admin decapod-admin migration apply
```

8. Optional. You can configure MongoDB to be not backward compatible with the previous release. To do so, run:

```
$ docker-compose -p PROJ exec database moshell
MongoDB server version: 3.4.2
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
        http://docs.mongodb.org/
Questions? Try the support group
        http://groups.google.com/group/mongodb-user
Server has startup warnings:
2017-02-14T07:00:13.729+0000 I STORAGE  [initandlisten]
2017-02-14T07:00:13.730+0000 I STORAGE  [initandlisten] ** WARNING: \
Using the XFS filesystem is strongly recommended with the WiredTiger storage␣
↪engine
2017-02-14T07:00:13.730+0000 I STORAGE  [initandlisten] **          \
See http://dochub.mongodb.org/core/prodnotes-filesystem
2017-02-14T07:00:15.199+0000 I CONTROL  [initandlisten]
2017-02-14T07:00:15.199+0000 I CONTROL  [initandlisten] ** WARNING: \
Access control is not enabled for the database.
2017-02-14T07:00:15.199+0000 I CONTROL  [initandlisten] **          \
Read and write access to data and configuration is unrestricted.
2017-02-14T07:00:15.199+0000 I CONTROL  [initandlisten]
2017-02-14T07:00:15.199+0000 I CONTROL  [initandlisten]
2017-02-14T07:00:15.199+0000 I CONTROL  [initandlisten] ** WARNING: \
/sys/kernel/mm/transparent_hugepage/enabled is 'always'.
2017-02-14T07:00:15.199+0000 I CONTROL  [initandlisten] **        \
We suggest setting it to 'never'
2017-02-14T07:00:15.199+0000 I CONTROL  [initandlisten]
2017-02-14T07:00:15.199+0000 I CONTROL  [initandlisten] ** WARNING: \
/sys/kernel/mm/transparent_hugepage/defrag is 'always'.
2017-02-14T07:00:15.199+0000 I CONTROL  [initandlisten] **        \
We suggest setting it to 'never'
2017-02-14T07:00:15.199+0000 I CONTROL  [initandlisten]
```

```
> db.adminCommand({setFeatureCompatibilityVersion: "3.4"})
{ "ok" : 1  }
```

9. Optional. Change the `root` password as described in *Reset password*.

Upgrade Decapod from 1.0.x to 1.1

Prior to upgrading Decapod, perform the steps described in *Prerequisites*.

Note: You do not need to perform any changes in the existing Ceph deployments.

Prerequisites

Prior to upgrading Decapod from 1.0.x to 1.1.x verify that you have `~/decapod_runtime` directory and have a contents from *Prerequisites*. If not, please create it first following that article.

Upgrade Decapod

To upgrade Decapod from 1.0.x to 1.1.x:

1. Back up the database.
   - Follow procedure, described in *Back up and restore Decapod*.
   - Try to restore database on another installation to verify that backup is not corrupted.
2. Obtain images for Decapod 1.1.x. to do so, follow steps 1-2 in the *Install Decapod*
3. Create containers from existing images:

   ```
   $ docker-compose -p PROJECT create
   decapod_database_1 is up-to-date
   Recreating decapod_api_1
   Recreating decapod_frontend_1
   Recreating decapod_controller_1
   Recreating decapod_admin_1
   ```

   This command won't restart existing containers, just prepare new ones.
4. Restart services.
   - If you can have a maintenence window, then restart all services at once:

     ```
     $ docker-compose -p PROJECT restart
     Restarting decapod_frontend_1 ... done
     Restarting decapod_api_1 ... done
     Restarting decapod_admin_1 ... done
     Restarting decapod_controller_1 ... done
     Restarting decapod_database_1 ... done
     Restarting decapod_database_data_1 ... done
     ```

   - Otherwise, please restart services in following sequence:

     (a) `api` service

```
$ docker-compose -p PROJECT restart api
Restarting decapod_api_1 ... done
```

(a) `frontend` service

```
$ docker-compose -p PROJECT restart frontend
Restarting decapod_frontend_1 ... done
```

(a) `admin` service

```
$ docker-compose -p PROJECT restart admin
Restarting decapod_admin_1 ... done
```

(a) `controller` service

```
$ docker-compose -p PROJECT restart controller
Restarting decapod_controller_1 ... done
```

(a) `database` service

```
$ docker-compose -p PROJECT restart database
Restarting decapod_database_1 ... done
```

5. Run database migrations:

```
$ docker-compose -p PROJECT exec -T admin decapod-admin migration apply
```

See also:

Back up and restore Decapod

Back up and restore Decapod

The `decapod-admin` tool allows you to manually create a backup of Decapod and its configuration and restore it.

Decapod stores its state in MongoDB. Restoring the database backups restores all the Decapod data except the internal container state, such as the data from ceph-monitoring that is refreshed every 10 minutes by default. However, you can collect such data explicitly using the `docker-compose exec controller decapod-collect-data` command.

To perform a backup:

```
$ docker-compose exec -T admin decapod-admin db backup > db_backup
```

To restore Decapod:

```
$ docker exec -i $(docker-compose ps -q admin) admin decapod-admin restore < db_backup
```

Note: Using `docker-compose exec` to perform the restore is currently not possible due to a `docker-compose` bug.

Alternatively, use the `backup_db.py` and `restore_db.py` scripts in the `./scripts` directory:

1. Run the scripts:

```
$ ./scripts/backup_db.py /var/backup/decapod_db
$ ./scripts/restore_db.py /var/backup/decapod_db
```

2. Add the backup to Cron:

```
0 */6 * * * /home/user/decapod_scripts/backup_db.py -p decapod -f \
/home/user/decapod_runtime/docker-compose.yml /var/backups/decapod/\
decapod_$(date --iso-8601) > /var/log/cron.log 2>&1
```

See also:

- *Monitor Ceph*
- *Admin service*

CHAPTER 3

API Reference

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*

API models

Decapod API is classical RESTful JSON API, but it operates with models. By term "models" it is meant JSON structured data in some generic way. Each entity for end user is present is some generic way.

This chapter tries to cover models in details, describing meaning of each field in each model. If you check *Usage example* or *decapodlib API* chapters, you will see some references to `data` field, `version` etc. Also, you will see, that updating of models require whole model. This chapter is intended to explain how to update models and why whole model is required.

Basic model

Basically, simple model looks like this:

```
{
    "data": {
        "somefield": "somevalue",
    },
    "id": "ee3944e8-758e-45dc-8e9e-e220478e442c",
    "initiator_id": null,
    "model": "something",
    "time_deleted": 0,
    "time_updated": 1479295535,
    "version": 1
}
```

As you can see, model has 2 parts: `data` field and *envelope*. Envelope is a set of fields which are common for every model and guaranteed to be there. `data` field is the model specific set of data and can be arbitrary. The only guarantee here is that field is mapping one (i.e `data` field cannot be list or null).

Basic Model JSON Schema definitions

There are some JSON Schema definitions that mentioned here to avoid duplication.

```
{
    "non_empty_string": {
        "type": "string",
        "minLength": 1,
        "maxLength": 1024
    },
    "email": {
        "allOf": [
            {"type": "string", "format": "email"},
            {
                "type": "string",
                "pattern": "^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$"
            }
        ]
    },
    "positive_integer": {
        "type": "number",
        "multipleOf": 1.0,
        "minimum": 0
    },
    "uuid4_array": {
        "type": "array",
        "items": {"$ref": "#/definitions/uuid4"}
    },
    "uuid4": {
        "type": "string",
        "pattern": "^[a-f0-9]{8}-?[a-f0-9]{4}-?4[a-f0-9]{3}-?[89ab][a-f0-9]{3}-?[a-f0-
↪9]{12}$"
    },
    "dmidecode_uuid": {
        "type": "string",
        "pattern": "^[0-9a-fA-F]{8}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-
↪9a-fA-F]{12}$"
    },
    "dmidecode_uuid_array": {
        "type": "array",
        "items": {"$ref": "#/definitions/dmidecode_uuid"}
    },
    "hostname": {
        "type": "string",
        "format": "hostname"
    },
    "ip": {
        "oneOf": [
            {"type": "string", "format": "ipv4"},
            {"type": "string", "format": "ipv6"}
        ]
    }
}
```

Basic Model JSON Schema

```
{
    "type": "object",
    "properties": {
        "id": {"$ref": "#/definitions/uuid4"},
        "model": {"$ref": "#/definitions/non_empty_string"},
        "time_updated": {"$ref": "#/definitions/positive_integer"},
        "time_deleted": {"$ref": "#/definitions/positive_integer"},
        "version": {"$ref": "#/definitions/positive_integer"},
        "initiator_id": {
            "anyOf": [
                {"type": "null"},
                {"$ref": "#/definitions/uuid4"}
            ]
        },
        "data": {"type": "object"}
    },
    "additionalProperties": false,
    "required": [
        "id",
        "model",
        "time_updated",
        "time_deleted",
        "version",
        "initiator_id",
        "data"
    ]
}
```

All model description below contains JSON Schema only for `data` field.

Field description

Field	Description
id	Unique identifier of the model. Most identifiers are simply UUID4 (**RFC 4122**).
initiator_id	ID of the user who initiated creation of that version.
model	Name of the model.
time_deleted	UNIX timestamp when model was deleted. If model is not deleted, then this field is `0`.
time_updated	UNIX timestamp when this model was modified last time.
version	Version of the model. Numbering starts from `1`.

A few things to know about data model in Decapod:

1. Nothing is deleted. Nothing is overwritten. You can always get whole history of changes for every model.
2. Decapod uses numbered versioning for a model. You may consider each version as value of the value.
3. If you update any field for a model, update does not occur inplace. Instead, new version is created. You can always access previous versions later to verify changes made in new version.
4. Deletion is not actual removing from database. Instead, new version is created. The only difference is in `time_deleted` field. If model was *deleted*, then `time_deleted` is UNIX timestamp of the moment when such event was occured. It is better to consider Decapod deletion as a mix of archivation and sealing.
5. Any active model (not deleted) has `time_deleted == 0`.
6. If model was deleted, any further progression is forbidden.

7. Deleted model is excluded from listings by default but it is always possible to access it with parametrized listing or direct request.

User

User model presents a data about Decapod user. This model never displays password of the user.

JSON Schema

```
{
    "login": {"$ref": "#/definitions/non_empty_string"},
    "email": {"$ref": "#/definitions/email"},
    "full_name": {"$ref": "#/definitions/non_empty_string"},
    "role_id": {
        "oneOf": [
            {"$ref": "#/definitions/uuid4"},
            {"type": "null"}
        ]
    }
}
```

Real-world Example

```
{
    "data": {
        "email": "noreply@example.com",
        "full_name": "Root User",
        "login": "root",
        "role_id": "4f96f3b0-85e5-4735-8c97-34fbef157c9d"
    },
    "id": "ee3944e8-758e-45dc-8e9e-e220478e442c",
    "initiator_id": null,
    "model": "user",
    "time_deleted": 0,
    "time_updated": 1479295535,
    "version": 1
}
```

Field description

Field	Description
email	Email of the user. This has to be real email, because user will get some important notifications like password reset here.
full_name	Full name of the user.
login	Username in Decapod
role_id	ID of role assigned to user. Can be `null` if no role is assigned.

Role

Role presents a set of permissions. Each API action require permissions, sometimes API may require conditional permissions: for example, playbook execution require permission on every playbook type.

JSON Schema

```
{
    "name": {"$ref": "#/definitions/non_empty_string"},
    "permissions": {
        "type": "array",
        "items": {
            "type": "object",
            "required": ["name", "permissions"],
            "additionalProperties": false,
            "properties": {
                "name": {"$ref": "#/definitions/non_empty_string"},
                "permissions": {
                    "type": "array",
                    "items": {"$ref": "#/definitions/non_empty_string"}
                }
            }
        }
    }
}
```

Real-world Example

```
{
    "data": {
        "name": "wheel",
        "permissions": [
            {
                "name": "playbook",
                "permissions": [
                    "add_osd",
                    "cluster_deploy",
                    "hello_world",
                    "purge_cluster",
                    "remove_osd"
                ]
            },
            {
                "name": "api",
                "permissions": [
                    "create_cluster",
                    "create_execution",
                    "create_playbook_configuration",
                    "create_role",
                    "create_server",
                    "create_user",
                    "delete_cluster",
                    "delete_execution",
                    "delete_playbook_confuiguration",
                    "delete_role",
                    "delete_server",
                    "delete_user",
                    "edit_cluster",
                    "edit_playbook_configuration",
                    "edit_role",
                    "edit_server",
```

```
                    "edit_user",
                    "view_cluster",
                    "view_cluster_versions",
                    "view_execution",
                    "view_execution_steps",
                    "view_execution_version",
                    "view_playbook_configuration",
                    "view_playbook_configuration_version",
                    "view_role",
                    "view_role_versions",
                    "view_server",
                    "view_server_versions",
                    "view_user",
                    "view_user_versions"
                ]
            }
        ]
    },
    "id": "4f96f3b0-85e5-4735-8c97-34fbef157c9d",
    "initiator_id": null,
    "model": "role",
    "time_deleted": 0,
    "time_updated": 1479295534,
    "version": 1
}
```

Field description

Field	Description
name	Name of the role.
permissions	A list of permissions for the role. Each permission refer some subset of interest: `api` permission is responsible for access to endpoints, `playbook` is responsible for playbook which this role allows to execute.

Cluster

Cluster model has all data, related to the cluster. Also, it provides credentials to access or configure apps to use with this Ceph cluster.

JSON Schema

```
{
    "name": {"$ref": "#/definitions/non_empty_string"},
    "configuration": {
        "type": "object",
        "additionalProperties": {
            "type": "array",
            "items": {
                "type": "object",
                "required": ["server_id", "version"],
                "properties": {
                    "server_id": {"$ref": "#/definitions/dmidecode_uuid"},
                    "version": {"$ref": "#/definitions/positive_integer"}
```

```
                }
            }
        }
    }
}
```

Real-world Example

```
{
    "data": {
        "configuration": {
            "mons": [
                {
                    "server_id": "3ee25709-215d-4f51-8348-20b4e7390fdb",
                    "version": 2
                }
            ],
            "osds": [
                {
                    "server_id": "045cdedf-898d-450d-8b3e-10a1bd20ece1",
                    "version": 2
                },
                {
                    "server_id": "0f26c53a-4ce6-4fdd-9e4b-ed7400abf8eb",
                    "version": 2
                },
                {
                    "server_id": "6cafad99-6353-448c-afbc-f161d0664522",
                    "version": 2
                },
                {
                    "server_id": "73079fc7-58a8-40b0-ba03-f02d7a4f2817",
                    "version": 2
                }
            ],
            "restapis": [
                {
                    "server_id": "3ee25709-215d-4f51-8348-20b4e7390fdb",
                    "version": 2
                }
            ]
        },
        "name": "ceph"
    },
    "id": "1597a71f-6619-4db6-9cda-a153f4f19097",
    "initiator_id": "9d010f3f-2ec0-4079-ae8c-f46415e2530c",
    "model": "cluster",
    "time_deleted": 0,
    "time_updated": 1478175677,
    "version": 3
}
```

Field description

Field	Description
name	Name of the cluster. This name will be propagated to Ceph by default (but always possible to redefine in playbook configuration).
configuration	Configuration of the cluster. In most cases this is a mapping of node role name (mon, osd etc) to the list of servers which have that role.

Server

Server model presents all information about Ceph node.

JSON Schema

```
{
    "name": {"$ref": "#/definitions/non_empty_string"},
    "fqdn": {"$ref": "#/definitions/hostname"},
    "ip": {"$ref": "#/definitions/ip"},
    "state": {
        "type": "string",
        "enum": {"$ref": "#/definitions/non_empty_string"}
    },
    "cluster_id": {"$ref": "#/definitions/uuid4"},
    "facts": {"type": "object"}
}
```

Real-world Example

```
{
    "data": {
        "cluster_id": "1597a71f-6619-4db6-9cda-a153f4f19097",
        "facts": {
            "ansible_all_ipv4_addresses": [
                "10.10.0.7"
            ],
            "ansible_all_ipv6_addresses": [
                "fe80::5054:ff:fe36:85df"
            ],
            "ansible_architecture": "x86_64",
            "ansible_bios_date": "04/01/2014",
            "ansible_bios_version": "Ubuntu-1.8.2-1ubuntu2",
            "ansible_cmdline": {
                "BOOT_IMAGE": "/boot/vmlinuz-4.4.0-45-generic",
                "ro": true,
                "root": "UUID=411bdb0c-80be-4a23-9876-9ce59f8f1f6a"
            },
            "ansible_date_time": {
                "date": "2016-11-03",
                "day": "03",
                "epoch": "1478174060",
                "hour": "11",
                "iso8601": "2016-11-03T11:54:20Z",
                "iso8601_basic": "20161103T115420460649",
```

```
                "iso8601_basic_short": "20161103T115420",
                "iso8601_micro": "2016-11-03T11:54:20.460724Z",
                "minute": "54",
                "month": "11",
                "second": "20",
                "time": "11:54:20",
                "tz": "UTC",
                "tz_offset": "+0000",
                "weekday": "Thursday",
                "weekday_number": "4",
                "weeknumber": "44",
                "year": "2016"
            },
            "ansible_default_ipv4": {
                "address": "10.10.0.7",
                "alias": "ens3",
                "broadcast": "10.10.0.255",
                "gateway": "10.10.0.1",
                "interface": "ens3",
                "macaddress": "52:54:00:36:85:df",
                "mtu": 1500,
                "netmask": "255.255.255.0",
                "network": "10.10.0.0",
                "type": "ether"
            },
            "ansible_default_ipv6": {},
            "ansible_devices": {
                "vda": {
                    "holders": [],
                    "host": "SCSI storage controller: Red Hat, Inc Virtio block device
↪",
                    "model": null,
                    "partitions": {
                        "vda1": {
                            "sectors": "31455199",
                            "sectorsize": 512,
                            "size": "15.00 GB",
                            "start": "2048"
                        }
                    },
                    "removable": "0",
                    "rotational": "1",
                    "sas_address": null,
                    "sas_device_handle": null,
                    "scheduler_mode": "",
                    "sectors": "31457280",
                    "sectorsize": "512",
                    "size": "15.00 GB",
                    "support_discard": "0",
                    "vendor": "0x1af4"
                },
                "vdb": {
                    "holders": [],
                    "host": "SCSI storage controller: Red Hat, Inc Virtio block device
↪",
                    "model": null,
                    "partitions": {},
                    "removable": "0",
```

```
                    "rotational": "1",
                    "sas_address": null,
                    "sas_device_handle": null,
                    "scheduler_mode": "",
                    "sectors": "41943040",
                    "sectorsize": "512",
                    "size": "20.00 GB",
                    "support_discard": "0",
                    "vendor": "0x1af4"
                },
                "vdc": {
                    "holders": [],
                    "host": "SCSI storage controller: Red Hat, Inc Virtio block device
↪",
                    "model": null,
                    "partitions": {},
                    "removable": "0",
                    "rotational": "1",
                    "sas_address": null,
                    "sas_device_handle": null,
                    "scheduler_mode": "",
                    "sectors": "41943040",
                    "sectorsize": "512",
                    "size": "20.00 GB",
                    "support_discard": "0",
                    "vendor": "0x1af4"
                },
                "vdd": {
                    "holders": [],
                    "host": "SCSI storage controller: Red Hat, Inc Virtio block device
↪",
                    "model": null,
                    "partitions": {},
                    "removable": "0",
                    "rotational": "1",
                    "sas_address": null,
                    "sas_device_handle": null,
                    "scheduler_mode": "",
                    "sectors": "41943040",
                    "sectorsize": "512",
                    "size": "20.00 GB",
                    "support_discard": "0",
                    "vendor": "0x1af4"
                },
                "vde": {
                    "holders": [],
                    "host": "SCSI storage controller: Red Hat, Inc Virtio block device
↪",
                    "model": null,
                    "partitions": {},
                    "removable": "0",
                    "rotational": "1",
                    "sas_address": null,
                    "sas_device_handle": null,
                    "scheduler_mode": "",
                    "sectors": "41943040",
                    "sectorsize": "512",
                    "size": "20.00 GB",
```

```
                    "support_discard": "0",
                    "vendor": "0x1af4"
                }
            },
            "ansible_distribution": "Ubuntu",
            "ansible_distribution_major_version": "16",
            "ansible_distribution_release": "xenial",
            "ansible_distribution_version": "16.04",
            "ansible_dns": {
                "nameservers": [
                    "10.10.0.5"
                ],
                "search": [
                    "maas"
                ]
            },
            "ansible_domain": "maas",
            "ansible_ens3": {
                "active": true,
                "device": "ens3",
                "ipv4": {
                    "address": "10.10.0.7",
                    "broadcast": "10.10.0.255",
                    "netmask": "255.255.255.0",
                    "network": "10.10.0.0"
                },
                "ipv6": [
                    {
                        "address": "fe80::5054:ff:fe36:85df",
                        "prefix": "64",
                        "scope": "link"
                    }
                ],
                "macaddress": "52:54:00:36:85:df",
                "mtu": 1500,
                "pciid": "virtio0",
                "promisc": false,
                "type": "ether"
            },
            "ansible_env": {
                "HOME": "/root",
                "LANG": "C.UTF-8",
                "LC_ALL": "C.UTF-8",
                "LC_MESSAGES": "C.UTF-8",
                "LOGNAME": "root",
                "MAIL": "/var/mail/root",
                "PATH": "/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/
↪bin:/snap/bin",
                "PWD": "/home/ansible",
                "SHELL": "/bin/bash",
                "SUDO_COMMAND": "/bin/sh -c echo BECOME-SUCCESS-
↪asonqrabuzwmtwyrpvxcbvdcgteywelc; LANG=C.UTF-8 LC_ALL=C.UTF-8 LC_MESSAGES=C.UTF-8 /
↪usr/bin/python /home/ansible/.ansible/tmp/ansible-tmp-1478174055.69-205903417866656/
↪setup; rm -rf \"/home/ansible/.ansible/tmp/ansible-tmp-1478174055.69-
↪205903417866656/\" > /dev/null 2>&1",
                "SUDO_GID": "1000",
                "SUDO_UID": "1000",
                "SUDO_USER": "ansible",
```

```
            "TERM": "unknown",
            "USER": "root",
            "USERNAME": "root"
        },
        "ansible_fips": false,
        "ansible_form_factor": "Other",
        "ansible_fqdn": "keen-skunk.maas",
        "ansible_gather_subset": [
            "hardware",
            "network",
            "virtual"
        ],
        "ansible_hostname": "keen-skunk",
        "ansible_interfaces": [
            "lo",
            "ens3"
        ],
        "ansible_kernel": "4.4.0-45-generic",
        "ansible_lo": {
            "active": true,
            "device": "lo",
            "ipv4": {
                "address": "127.0.0.1",
                "broadcast": "host",
                "netmask": "255.0.0.0",
                "network": "127.0.0.0"
            },
            "ipv6": [
                {
                    "address": "::1",
                    "prefix": "128",
                    "scope": "host"
                }
            ],
            "mtu": 65536,
            "promisc": false,
            "type": "loopback"
        },
        "ansible_lsb": {
            "codename": "xenial",
            "description": "Ubuntu 16.04.1 LTS",
            "id": "Ubuntu",
            "major_release": "16",
            "release": "16.04"
        },
        "ansible_lvm": {
            "lvs": {},
            "vgs": {}
        },
        "ansible_machine": "x86_64",
        "ansible_machine_id": "0e6a3562c17049e8a294af590f730ed4",
        "ansible_memfree_mb": 128,
        "ansible_memory_mb": {
            "nocache": {
                "free": 384,
                "used": 104
            },
            "real": {
```

```
                "free": 128,
                "total": 488,
                "used": 360
            },
            "swap": {
                "cached": 0,
                "free": 975,
                "total": 975,
                "used": 0
            }
        },
        "ansible_memtotal_mb": 488,
        "ansible_mounts": [
            {
                "device": "/dev/vda1",
                "fstype": "ext4",
                "mount": "/",
                "options": "rw,relatime,data=ordered",
                "size_available": 12425428992,
                "size_total": 15718117376,
                "uuid": "411bdb0c-80be-4a23-9876-9ce59f8f1f6a"
            }
        ],
        "ansible_nodename": "keen-skunk",
        "ansible_os_family": "Debian",
        "ansible_pkg_mgr": "apt",
        "ansible_processor": [
            "GenuineIntel",
            "Intel Core Processor (Haswell, no TSX)"
        ],
        "ansible_processor_cores": 1,
        "ansible_processor_count": 1,
        "ansible_processor_threads_per_core": 1,
        "ansible_processor_vcpus": 1,
        "ansible_product_name": "Standard PC (i440FX + PIIX, 1996)",
        "ansible_product_serial": "NA",
        "ansible_product_uuid": "0F26C53A-4CE6-4FDD-9E4B-ED7400ABF8EB",
        "ansible_product_version": "pc-i440fx-xenial",
        "ansible_python": {
            "executable": "/usr/bin/python",
            "has_sslcontext": true,
            "type": "CPython",
            "version": {
                "major": 2,
                "micro": 12,
                "minor": 7,
                "releaselevel": "final",
                "serial": 0
            },
            "version_info": [
                2,
                7,
                12,
                "final",
                0
            ]
        },
        "ansible_python_version": "2.7.12",
```

```
            "ansible_selinux": false,
            "ansible_service_mgr": "systemd",
            "ansible_ssh_host_key_dsa_public":
↪"AAAAB3NzaC1kc3MAAACBAI1VgHKG80TcfuMIwCwbGyT+IoA+wTxzx/CscE/
↪QI+DiNQFV3vbE3pRZuAuzWu+SeNfxfP7ZCc57Yc9KvZjImvsOTk1mzMO1xCuHWmLUOAzvmf3fuTYAp6+UzpqKuOHbAVyD7Qzccu
↪e3IX4uh1FmKruSB6FbQAAAIA/
↪8jPxLt3zZ7cwNQhQevwQ3MCU6cgzIUJnZaVdw+G1uWIw6bbYxB60clT4+Z3jajIIx6pWnviQUQqKy3Uj4Ua+N9vnEz5JgMrvxVX
↪xgKuE52xV+B3+gJMA3prSdlRGhuAwQbx9ql/B7PmTdND7ZNw35GOalbMrIY/yw==",
            "ansible_ssh_host_key_ecdsa_public":
↪"AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBP02SVRKRQDJllThy5fVh0Rm8hx8fkKvYzgt73ghPx/
↪FSCWnvzuzrA9yNWR7iBnkcgkpNiUHJwH1Seg3V1NTZ/Y=",
            "ansible_ssh_host_key_ed25519_public":
↪"AAAAC3NzaC1lZDI1NTE5AAAAIPcT3RxDxCA1Adc/k+eDRN5IpAkx201rypKJpnydPXLw",
            "ansible_ssh_host_key_rsa_public":
↪"AAAAB3NzaC1yc2EAAAADAQABAAABAQC7ur+mkamaX/Wnsz90mlwca8GxW58ti/UQwqT89rCv12JSlR2v/
↪Crer8b4zcea06EgCP/Z0ow6RF/
↪LxVNEUFlwtkZJ6inXL6WOrNu9BphBuBMy8+f3BqlMMIs4zEQAoESOQssHA66JhQSYdM1cHYAUUtFNmP8Ht9Ik32qpkGPwU2bEau
↪Guv//RegNpErT7apAp/fZ/
↪OdJw6+6cE13AgzXyjcBWkrnHVyvUMB8VWr9ExNKtEwetBYVGVt6CT6icrr4r3ceD+aQDYczzawZIKA+TTjTrLy6l9hpCId81/
↪PywaddJJWqmQNSZHmva+GX",
            "ansible_swapfree_mb": 975,
            "ansible_swaptotal_mb": 975,
            "ansible_system": "Linux",
            "ansible_system_capabilities": [
                "cap_chown",
                "cap_dac_override",
                "cap_dac_read_search",
                "cap_fowner",
                "cap_fsetid",
                "cap_kill",
                "cap_setgid",
                "cap_setuid",
                "cap_setpcap",
                "cap_linux_immutable",
                "cap_net_bind_service",
                "cap_net_broadcast",
                "cap_net_admin",
                "cap_net_raw",
                "cap_ipc_lock",
                "cap_ipc_owner",
                "cap_sys_module",
                "cap_sys_rawio",
                "cap_sys_chroot",
                "cap_sys_ptrace",
                "cap_sys_pacct",
                "cap_sys_admin",
                "cap_sys_boot",
                "cap_sys_nice",
                "cap_sys_resource",
                "cap_sys_time",
                "cap_sys_tty_config",
                "cap_mknod",
                "cap_lease",
                "cap_audit_write",
                "cap_audit_control",
                "cap_setfcap",
                "cap_mac_override",
                "cap_mac_admin",
```

```
                "cap_syslog",
                "cap_wake_alarm",
                "cap_block_suspend",
                "37+ep"
            ],
            "ansible_system_capabilities_enforced": "True",
            "ansible_system_vendor": "QEMU",
            "ansible_uptime_seconds": 107,
            "ansible_user_dir": "/root",
            "ansible_user_gecos": "root",
            "ansible_user_gid": 0,
            "ansible_user_id": "root",
            "ansible_user_shell": "/bin/bash",
            "ansible_user_uid": 0,
            "ansible_userspace_architecture": "x86_64",
            "ansible_userspace_bits": "64",
            "ansible_virtualization_role": "guest",
            "ansible_virtualization_type": "kvm",
            "module_setup": true
        },
        "fqdn": "keen-skunk",
        "ip": "10.10.0.7",
        "name": "keen-skunk",
        "state": "operational",
        "username": "ansible"
    },
    "id": "0f26c53a-4ce6-4fdd-9e4b-ed7400abf8eb",
    "initiator_id": null,
    "model": "server",
    "time_deleted": 0,
    "time_updated": 1478174236,
    "version": 2
}
```


Field description

Field	Description
cluster_id	ID of the cluster which has this server.
facts	Ansible facts for that server.
fqdn	FQDN of the server.
name	Human-readable name of the server.
state	State of the server (operational, off etc)
username	Username which Ansible uses to connect to this server.

Possible states:

State	Description
operational	Server is up and running.
off	Server was excluded from Decapod.
maintenance_no_reconfig	Server is in maintenance, but no cluster reconfiguration is required.
maintenance_reconfig	Server is in maintenance, cluster reconfiguration is required.

Playbook Configuration

Every playbook requires configuration. This model presents such configuration. On create of playbook configuration, Decapod generates config for given server list and playbook according to best practices. It just proposes a good config, user always may update it.

JSON Schema

```
{
    "name": {"$ref": "#/definitions/non_empty_string"},
    "playbook_id": {"$ref": "#/definitions/non_empty_string"},
    "cluster_id": {"$ref": "#/definitions/uuid4"},
    "configuration": {"type": "object"}
}
```

Real-world Example

```
{
    "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"
                },
                "10.10.0.4": {
                    "ansible_user": "ansible",
                    "devices": [
                        "/dev/vdb"
                    ],
                    "monitor_interface": "ens3"
                },
                "10.10.0.7": {
                    "ansible_user": "ansible",
                    "devices": [
                        "/dev/vdd",
                        "/dev/vde",
                        "/dev/vdc",
                        "/dev/vdb"
                    ],
                    "monitor_interface": "ens3"
                },
                "10.10.0.8": {
                    "ansible_user": "ansible",
                    "devices": [
                        "/dev/vdd",
                        "/dev/vde",
                        "/dev/vdc",
                        "/dev/vdb"
                    ],
                    "monitor_interface": "ens3"
                }
            }
        },
        "clients": [],
        "iscsi_gw": [],
        "mdss": [],
        "mons": [
            "10.10.0.2"
        ],
        "nfss": [],
        "osds": [
            "10.10.0.7",
            "10.10.0.8",
            "10.10.0.3",
            "10.10.0.4"
        ],
        "rbdmirrors": [],
        "restapis": [
            "10.10.0.2"
        ],
        "rgws": []
```

```
            }
        },
        "name": "deploy",
        "playbook_id": "cluster_deploy"
    },
    "id": "fd76cea9-3efa-4432-854c-fee30ca79ddb",
    "initiator_id": "9d010f3f-2ec0-4079-ae8c-f46415e2530c",
    "model": "playbook_configuration",
    "time_deleted": 0,
    "time_updated": 1478174220,
    "version": 2
}
```

Field description

Field	Description
cluster_id	ID of the cluster to deploy.
configuration	Configuration of the playbook.
name	Name of the playbook configuration.
playbook_id	ID of the playbook to use.

Configuration differs from one playbook to another. Please check documentation on playbook plugins (TODO) to get a meaning of each configuration option.

Execution

Execution is the model, which incapsulates data about execution of certain playbook configuration on the cluster. You may consider it as a run of **`ansible-playbook`**.

JSON Schema

```
{
    "playbook_configuration": {
        "type": "object",
        "additionalProperties": false,
        "required": ["id", "version"],
        "properties": {
            "id": {"$ref": "#/definitions/uuid4"},
            "version": {"$ref": "#/definitions/positive_integer"}
        }
    },
    "state": {"$ref": "#/definitions/non_empty_string"}
}
```

Real-world Example

```
{
    "data": {
        "playbook_configuration": {
            "id": "fd76cea9-3efa-4432-854c-fee30ca79ddb",
            "version": 2
```

```
        },
        "state": "failed"
    },
    "id": "6f016e18-97c4-4069-9e99-70862d98e46a",
    "initiator_id": null,
    "model": "execution",
    "time_deleted": 0,
    "time_updated": 1478175025,
    "version": 3
}
```


Field description

Field	Description
playbook_configuration	Information about ID and version of used playbook configuration.
state	State of execution (failed, completed etc)

Possible states:

State	Description
created	Execution was created but not started yet.
started	Execution is in progress.
completed	Execution was completed successfuly.
failed	Execution was failed.
canceling	Canceling of execution is in progress.
canceled	Execution has been canceled.

Execution Step

This is a model of step of playbook execution. Step is a granular task of configuration management system.

JSON Schema


```
{
    "error": {"type": "object"},
    "execution_id": {"$ref": "#/definitions/uuid4"},
    "name": {"$ref": "#/definitions/non_empty_string"},
    "result": {"$ref": "#/definitions/non_empty_string"},
    "role": {"$ref": "#/definitions/non_empty_string"},
    "server_id": {"$ref": "#/definitions/uuid4"},
    "time_started": {"$ref": "#/definitions/positive_integer"},
    "time_finished": {"$ref": "#/definitions/positive_integer"}
}
```


Real-world Example


```
{
    "data": {
        "error": {},
        "execution_id": "6f016e18-97c4-4069-9e99-70862d98e46a",
        "name": "set config and keys paths",
```

```
        "result": "skipped",
        "role": "ceph-restapi",
        "server_id": "3ee25709-215d-4f51-8348-20b4e7390fdb",
        "time_finished": 1478175019,
        "time_started": 1478175019
    },
    "id": "581b292b3ceda10087ab8d41",
    "initiator_id": "6f016e18-97c4-4069-9e99-70862d98e46a",
    "model": "execution_step",
    "time_deleted": 0,
    "time_updated": 1478175019,
    "version": 1
}
```

Field description

Field	Description
error	Error data from Ansible
execution_id	ID of execution made
name	Name of the task which was executed
result	Result of the task execution (failed, ok, ...).
role	Role which task belongs to.
server_id	ID of the server where task was performed.
time_started	UNIX timestamp when task was started.
time_finished	UNIX timestamp when task was finished.

Possible states:

State	Description
ok	Task was executed without any problems.
skipped	Task execution was skipped.
failed	Task was failed.
unreachable	Task was not executed because remote host is unreachable.

Token

Token model presents an authentication token. Token is a string which should be put in **Authorization** header of every request and Decapod API uses it as an authentication mean for operations.

`version` is rudimentary field here and kept for consistency. Do not rely on this field, it always equals 1.

JSON Schema

```
{
    "user": {"type": "User Model"}
    "expires_at": {"$ref": "#/definitions/positive_integer"}
}
```

Real-world Example


```
{
    "data":{
        "expires_at":1479455919,
        "user":{
            "data":{
                "email":"noreply@example.com",
                "full_name":"Root User",
                "login":"root",
                "role_id":"4f96f3b0-85e5-4735-8c97-34fbef157c9d"
            },
            "id":"ee3944e8-758e-45dc-8e9e-e220478e442c",
            "initiator_id":null,
            "model":"user",
            "time_deleted":0,
            "time_updated":1479295535,
            "version":1
        }
    },
    "id":"cc6cf706-2f26-4975-9885-0d9c234491b2",
    "initiator_id":"ee3944e8-758e-45dc-8e9e-e220478e442c",
    "model":"token",
    "time_deleted":0,
    "time_updated":1479454119,
    "version":1
}
```


Field description

Field	Description
expires_at	UNIX timestamp of moment when this token will be considered as expired.
user	Expanded model of user logged in.

API models

Decapod API is classical RESTful JSON API, but it operates with models. By term "models" it is meant JSON structured data in some generic way. Each entity for end user is present is some generic way.

This chapter tries to cover models in details, describing meaning of each field in each model. If you check *Usage example* or *decapodlib API* chapters, you will see some references to `data` field, `version` etc. Also, you will see, that updating of models require whole model. This chapter is intended to explain how to update models and why whole model is required.

Basic model

Basically, simple model looks like this:

```
{
    "data": {
        "somefield": "somevalue",
    },
    "id": "ee3944e8-758e-45dc-8e9e-e220478e442c",
```

```
    "initiator_id": null,
    "model": "something",
    "time_deleted": 0,
    "time_updated": 1479295535,
    "version": 1
}
```

As you can see, model has 2 parts: `data` field and *envelope*. Envelope is a set of fields which are common for every model and guaranteed to be there. `data` field is the model specific set of data and can be arbitrary. The only guarantee here is that field is mapping one (i.e `data` field cannot be list or null).

Basic Model JSON Schema definitions

There are some JSON Schema definitions that mentioned here to avoid duplication.

```
{
    "non_empty_string": {
        "type": "string",
        "minLength": 1,
        "maxLength": 1024
    },
    "email": {
        "allOf": [
            {"type": "string", "format": "email"},
            {
                "type": "string",
                "pattern": "^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$"
            }
        ]
    },
    "positive_integer": {
        "type": "number",
        "multipleOf": 1.0,
        "minimum": 0
    },
    "uuid4_array": {
        "type": "array",
        "items": {"$ref": "#/definitions/uuid4"}
    },
    "uuid4": {
        "type": "string",
        "pattern": "^[a-f0-9]{8}-?[a-f0-9]{4}-?4[a-f0-9]{3}-?[89ab][a-f0-9]{3}-?[a-f0-
↪9]{12}$"
    },
    "dmidecode_uuid": {
        "type": "string",
        "pattern": "^[0-9a-fA-F]{8}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-
↪9a-fA-F]{12}$"
    },
    "dmidecode_uuid_array": {
        "type": "array",
        "items": {"$ref": "#/definitions/dmidecode_uuid"}
    },
    "hostname": {
        "type": "string",
        "format": "hostname"
    },
```

```
    "ip": {
        "oneOf": [
            {"type": "string", "format": "ipv4"},
            {"type": "string", "format": "ipv6"}
        ]
    }
}
```

Basic Model JSON Schema

```
{
    "type": "object",
    "properties": {
        "id": {"$ref": "#/definitions/uuid4"},
        "model": {"$ref": "#/definitions/non_empty_string"},
        "time_updated": {"$ref": "#/definitions/positive_integer"},
        "time_deleted": {"$ref": "#/definitions/positive_integer"},
        "version": {"$ref": "#/definitions/positive_integer"},
        "initiator_id": {
            "anyOf": [
                {"type": "null"},
                {"$ref": "#/definitions/uuid4"}
            ]
        },
        "data": {"type": "object"}
    },
    "additionalProperties": false,
    "required": [
        "id",
        "model",
        "time_updated",
        "time_deleted",
        "version",
        "initiator_id",
        "data"
    ]
}
```

All model description below contains JSON Schema only for `data` field.

Field description

Field	Description
id	Unique identifier of the model. Most identifiers are simply UUID4 (**RFC 4122**).
initiator_id	ID of the user who initiated creation of that version.
model	Name of the model.
time_deleted	UNIX timestamp when model was deleted. If model is not deleted, then this field is `0`.
time_updated	UNIX timestamp when this model was modified last time.
version	Version of the model. Numbering starts from `1`.

A few things to know about data model in Decapod:

1. Nothing is deleted. Nothing is overwritten. You can always get whole history of changes for every model.
2. Decapod uses numbered versioning for a model. You may consider each version as value of the value.

3. If you update any field for a model, update does not occur inplace. Instead, new version is created. You can always access previous versions later to verify changes made in new version.
4. Deletion is not actual removing from database. Instead, new version is created. The only difference is in `time_deleted` field. If model was *deleted*, then `time_deleted` is UNIX timestamp of the moment when such event was occured. It is better to consider Decapod deletion as a mix of archivation and sealing.
5. Any active model (not deleted) has `time_deleted == 0`.
6. If model was deleted, any further progression is forbidden.
7. Deleted model is excluded from listings by default but it is always possible to access it with parametrized listing or direct request.

User

User model presents a data about Decapod user. This model never displays password of the user.

JSON Schema

```
{
    "login": {"$ref": "#/definitions/non_empty_string"},
    "email": {"$ref": "#/definitions/email"},
    "full_name": {"$ref": "#/definitions/non_empty_string"},
    "role_id": {
        "oneOf": [
            {"$ref": "#/definitions/uuid4"},
            {"type": "null"}
        ]
    }
}
```

Real-world Example

```
{
    "data": {
        "email": "noreply@example.com",
        "full_name": "Root User",
        "login": "root",
        "role_id": "4f96f3b0-85e5-4735-8c97-34fbef157c9d"
    },
    "id": "ee3944e8-758e-45dc-8e9e-e220478e442c",
    "initiator_id": null,
    "model": "user",
    "time_deleted": 0,
    "time_updated": 1479295535,
    "version": 1
}
```

Field description

Field	Description
email	Email of the user. This has to be real email, because user will get some important notifications like password reset here.
full_name	Full name of the user.
login	Username in Decapod
role_id	ID of role assigned to user. Can be `null` if no role is assigned.

Role

Role presents a set of permissions. Each API action require permissions, sometimes API may require conditional permissions: for example, playbook execution require permission on every playbook type.

JSON Schema

```
{
    "name": {"$ref": "#/definitions/non_empty_string"},
    "permissions": {
        "type": "array",
        "items": {
            "type": "object",
            "required": ["name", "permissions"],
            "additionalProperties": false,
            "properties": {
                "name": {"$ref": "#/definitions/non_empty_string"},
                "permissions": {
                    "type": "array",
                    "items": {"$ref": "#/definitions/non_empty_string"}
                }
            }
        }
    }
}
```

Real-world Example

```
{
    "data": {
        "name": "wheel",
        "permissions": [
            {
                "name": "playbook",
                "permissions": [
                    "add_osd",
                    "cluster_deploy",
                    "hello_world",
                    "purge_cluster",
                    "remove_osd"
                ]
            },
            {
                "name": "api",
```

```
                "permissions": [
                    "create_cluster",
                    "create_execution",
                    "create_playbook_configuration",
                    "create_role",
                    "create_server",
                    "create_user",
                    "delete_cluster",
                    "delete_execution",
                    "delete_playbook_confuiguration",
                    "delete_role",
                    "delete_server",
                    "delete_user",
                    "edit_cluster",
                    "edit_playbook_configuration",
                    "edit_role",
                    "edit_server",
                    "edit_user",
                    "view_cluster",
                    "view_cluster_versions",
                    "view_execution",
                    "view_execution_steps",
                    "view_execution_version",
                    "view_playbook_configuration",
                    "view_playbook_configuration_version",
                    "view_role",
                    "view_role_versions",
                    "view_server",
                    "view_server_versions",
                    "view_user",
                    "view_user_versions"
                ]
            }
        ]
    },
    "id": "4f96f3b0-85e5-4735-8c97-34fbef157c9d",
    "initiator_id": null,
    "model": "role",
    "time_deleted": 0,
    "time_updated": 1479295534,
    "version": 1
}
```


Field description

Field	Description
name	Name of the role.
permissions	A list of permissions for the role. Each permission refer some subset of interest: `api` permission is responsible for access to endpoints, `playbook` is responsible for playbook which this role allows to execute.

Cluster

Cluster model has all data, related to the cluster. Also, it provides credentials to access or configure apps to use with this Ceph cluster.

JSON Schema

```
{
    "name": {"$ref": "#/definitions/non_empty_string"},
    "configuration": {
        "type": "object",
        "additionalProperties": {
            "type": "array",
            "items": {
                "type": "object",
                "required": ["server_id", "version"],
                "properties": {
                    "server_id": {"$ref": "#/definitions/dmidecode_uuid"},
                    "version": {"$ref": "#/definitions/positive_integer"}
                }
            }
        }
    }
}
```

Real-world Example

```
{
    "data": {
        "configuration": {
            "mons": [
                {
                    "server_id": "3ee25709-215d-4f51-8348-20b4e7390fdb",
                    "version": 2
                }
            ],
            "osds": [
                {
                    "server_id": "045cdedf-898d-450d-8b3e-10a1bd20ece1",
                    "version": 2
                },
                {
                    "server_id": "0f26c53a-4ce6-4fdd-9e4b-ed7400abf8eb",
                    "version": 2
                },
                {
                    "server_id": "6cafad99-6353-448c-afbc-f161d0664522",
                    "version": 2
                },
                {
                    "server_id": "73079fc7-58a8-40b0-ba03-f02d7a4f2817",
                    "version": 2
                }
            ],
            "restapis": [
                {
                    "server_id": "3ee25709-215d-4f51-8348-20b4e7390fdb",
                    "version": 2
                }
            ]
        },
```

```
        "name": "ceph"
    },
    "id": "1597a71f-6619-4db6-9cda-a153f4f19097",
    "initiator_id": "9d010f3f-2ec0-4079-ae8c-f46415e2530c",
    "model": "cluster",
    "time_deleted": 0,
    "time_updated": 1478175677,
    "version": 3
}
```

Field description

Field	Description
name	Name of the cluster. This name will be propagated to Ceph by default (but always possible to redefine in playbook configuration).
configuration	Configuration of the cluster. In most cases this is a mapping of node role name (mon, osd etc) to the list of servers which have that role.

Server

Server model presents all information about Ceph node.

JSON Schema

```
{
    "name": {"$ref": "#/definitions/non_empty_string"},
    "fqdn": {"$ref": "#/definitions/hostname"},
    "ip": {"$ref": "#/definitions/ip"},
    "state": {
        "type": "string",
        "enum": {"$ref": "#/definitions/non_empty_string"}
    },
    "cluster_id": {"$ref": "#/definitions/uuid4"},
    "facts": {"type": "object"}
}
```

Real-world Example

```
{
    "data": {
        "cluster_id": "1597a71f-6619-4db6-9cda-a153f4f19097",
        "facts": {
            "ansible_all_ipv4_addresses": [
                "10.10.0.7"
            ],
            "ansible_all_ipv6_addresses": [
                "fe80::5054:ff:fe36:85df"
            ],
            "ansible_architecture": "x86_64",
            "ansible_bios_date": "04/01/2014",
            "ansible_bios_version": "Ubuntu-1.8.2-1ubuntu2",
```

```
            "ansible_cmdline": {
                "BOOT_IMAGE": "/boot/vmlinuz-4.4.0-45-generic",
                "ro": true,
                "root": "UUID=411bdb0c-80be-4a23-9876-9ce59f8f1f6a"
            },
            "ansible_date_time": {
                "date": "2016-11-03",
                "day": "03",
                "epoch": "1478174060",
                "hour": "11",
                "iso8601": "2016-11-03T11:54:20Z",
                "iso8601_basic": "20161103T115420460649",
                "iso8601_basic_short": "20161103T115420",
                "iso8601_micro": "2016-11-03T11:54:20.460724Z",
                "minute": "54",
                "month": "11",
                "second": "20",
                "time": "11:54:20",
                "tz": "UTC",
                "tz_offset": "+0000",
                "weekday": "Thursday",
                "weekday_number": "4",
                "weeknumber": "44",
                "year": "2016"
            },
            "ansible_default_ipv4": {
                "address": "10.10.0.7",
                "alias": "ens3",
                "broadcast": "10.10.0.255",
                "gateway": "10.10.0.1",
                "interface": "ens3",
                "macaddress": "52:54:00:36:85:df",
                "mtu": 1500,
                "netmask": "255.255.255.0",
                "network": "10.10.0.0",
                "type": "ether"
            },
            "ansible_default_ipv6": {},
            "ansible_devices": {
                "vda": {
                    "holders": [],
                    "host": "SCSI storage controller: Red Hat, Inc Virtio block device
↪",
                    "model": null,
                    "partitions": {
                        "vda1": {
                            "sectors": "31455199",
                            "sectorsize": 512,
                            "size": "15.00 GB",
                            "start": "2048"
                        }
                    },
                    "removable": "0",
                    "rotational": "1",
                    "sas_address": null,
                    "sas_device_handle": null,
                    "scheduler_mode": "",
                    "sectors": "31457280",
```

```
                    "sectorsize": "512",
                    "size": "15.00 GB",
                    "support_discard": "0",
                    "vendor": "0x1af4"
                },
                "vdb": {
                    "holders": [],
                    "host": "SCSI storage controller: Red Hat, Inc Virtio block device
↪",
                    "model": null,
                    "partitions": {},
                    "removable": "0",
                    "rotational": "1",
                    "sas_address": null,
                    "sas_device_handle": null,
                    "scheduler_mode": "",
                    "sectors": "41943040",
                    "sectorsize": "512",
                    "size": "20.00 GB",
                    "support_discard": "0",
                    "vendor": "0x1af4"
                },
                "vdc": {
                    "holders": [],
                    "host": "SCSI storage controller: Red Hat, Inc Virtio block device
↪",
                    "model": null,
                    "partitions": {},
                    "removable": "0",
                    "rotational": "1",
                    "sas_address": null,
                    "sas_device_handle": null,
                    "scheduler_mode": "",
                    "sectors": "41943040",
                    "sectorsize": "512",
                    "size": "20.00 GB",
                    "support_discard": "0",
                    "vendor": "0x1af4"
                },
                "vdd": {
                    "holders": [],
                    "host": "SCSI storage controller: Red Hat, Inc Virtio block device
↪",
                    "model": null,
                    "partitions": {},
                    "removable": "0",
                    "rotational": "1",
                    "sas_address": null,
                    "sas_device_handle": null,
                    "scheduler_mode": "",
                    "sectors": "41943040",
                    "sectorsize": "512",
                    "size": "20.00 GB",
                    "support_discard": "0",
                    "vendor": "0x1af4"
                },
                "vde": {
                    "holders": [],
```

```
                    "host": "SCSI storage controller: Red Hat, Inc Virtio block device
↪",
                    "model": null,
                    "partitions": {},
                    "removable": "0",
                    "rotational": "1",
                    "sas_address": null,
                    "sas_device_handle": null,
                    "scheduler_mode": "",
                    "sectors": "41943040",
                    "sectorsize": "512",
                    "size": "20.00 GB",
                    "support_discard": "0",
                    "vendor": "0x1af4"
                }
            },
            "ansible_distribution": "Ubuntu",
            "ansible_distribution_major_version": "16",
            "ansible_distribution_release": "xenial",
            "ansible_distribution_version": "16.04",
            "ansible_dns": {
                "nameservers": [
                    "10.10.0.5"
                ],
                "search": [
                    "maas"
                ]
            },
            "ansible_domain": "maas",
            "ansible_ens3": {
                "active": true,
                "device": "ens3",
                "ipv4": {
                    "address": "10.10.0.7",
                    "broadcast": "10.10.0.255",
                    "netmask": "255.255.255.0",
                    "network": "10.10.0.0"
                },
                "ipv6": [
                    {
                        "address": "fe80::5054:ff:fe36:85df",
                        "prefix": "64",
                        "scope": "link"
                    }
                ],
                "macaddress": "52:54:00:36:85:df",
                "mtu": 1500,
                "pciid": "virtio0",
                "promisc": false,
                "type": "ether"
            },
            "ansible_env": {
                "HOME": "/root",
                "LANG": "C.UTF-8",
                "LC_ALL": "C.UTF-8",
                "LC_MESSAGES": "C.UTF-8",
                "LOGNAME": "root",
                "MAIL": "/var/mail/root",
```

```
                "PATH": "/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/
↪bin:/snap/bin",
                "PWD": "/home/ansible",
                "SHELL": "/bin/bash",
                "SUDO_COMMAND": "/bin/sh -c echo BECOME-SUCCESS-
↪asonqrabuzwmtwyrpvxcbvdcgteywelc; LANG=C.UTF-8 LC_ALL=C.UTF-8 LC_MESSAGES=C.UTF-8 /
↪usr/bin/python /home/ansible/.ansible/tmp/ansible-tmp-1478174055.69-205903417866656/
↪setup; rm -rf \"/home/ansible/.ansible/tmp/ansible-tmp-1478174055.69-
↪205903417866656/\" > /dev/null 2>&1",
                "SUDO_GID": "1000",
                "SUDO_UID": "1000",
                "SUDO_USER": "ansible",
                "TERM": "unknown",
                "USER": "root",
                "USERNAME": "root"
            },
            "ansible_fips": false,
            "ansible_form_factor": "Other",
            "ansible_fqdn": "keen-skunk.maas",
            "ansible_gather_subset": [
                "hardware",
                "network",
                "virtual"
            ],
            "ansible_hostname": "keen-skunk",
            "ansible_interfaces": [
                "lo",
                "ens3"
            ],
            "ansible_kernel": "4.4.0-45-generic",
            "ansible_lo": {
                "active": true,
                "device": "lo",
                "ipv4": {
                    "address": "127.0.0.1",
                    "broadcast": "host",
                    "netmask": "255.0.0.0",
                    "network": "127.0.0.0"
                },
                "ipv6": [
                    {
                        "address": "::1",
                        "prefix": "128",
                        "scope": "host"
                    }
                ],
                "mtu": 65536,
                "promisc": false,
                "type": "loopback"
            },
            "ansible_lsb": {
                "codename": "xenial",
                "description": "Ubuntu 16.04.1 LTS",
                "id": "Ubuntu",
                "major_release": "16",
                "release": "16.04"
            },
            "ansible_lvm": {
```

```
            "lvs": {},
            "vgs": {}
        },
        "ansible_machine": "x86_64",
        "ansible_machine_id": "0e6a3562c17049e8a294af590f730ed4",
        "ansible_memfree_mb": 128,
        "ansible_memory_mb": {
            "nocache": {
                "free": 384,
                "used": 104
            },
            "real": {
                "free": 128,
                "total": 488,
                "used": 360
            },
            "swap": {
                "cached": 0,
                "free": 975,
                "total": 975,
                "used": 0
            }
        },
        "ansible_memtotal_mb": 488,
        "ansible_mounts": [
            {
                "device": "/dev/vda1",
                "fstype": "ext4",
                "mount": "/",
                "options": "rw,relatime,data=ordered",
                "size_available": 12425428992,
                "size_total": 15718117376,
                "uuid": "411bdb0c-80be-4a23-9876-9ce59f8f1f6a"
            }
        ],
        "ansible_nodename": "keen-skunk",
        "ansible_os_family": "Debian",
        "ansible_pkg_mgr": "apt",
        "ansible_processor": [
            "GenuineIntel",
            "Intel Core Processor (Haswell, no TSX)"
        ],
        "ansible_processor_cores": 1,
        "ansible_processor_count": 1,
        "ansible_processor_threads_per_core": 1,
        "ansible_processor_vcpus": 1,
        "ansible_product_name": "Standard PC (i440FX + PIIX, 1996)",
        "ansible_product_serial": "NA",
        "ansible_product_uuid": "0F26C53A-4CE6-4FDD-9E4B-ED7400ABF8EB",
        "ansible_product_version": "pc-i440fx-xenial",
        "ansible_python": {
            "executable": "/usr/bin/python",
            "has_sslcontext": true,
            "type": "CPython",
            "version": {
                "major": 2,
                "micro": 12,
                "minor": 7,
```

```
                    "releaselevel": "final",
                    "serial": 0
                },
                "version_info": [
                    2,
                    7,
                    12,
                    "final",
                    0
                ]
            },
            "ansible_python_version": "2.7.12",
            "ansible_selinux": false,
            "ansible_service_mgr": "systemd",
            "ansible_ssh_host_key_dsa_public":
↪"AAAAB3NzaC1kc3MAAACBAI1VgHKG80TcfuMIwCwbGyT+IoA+wTxzx/CscE/
↪QI+DiNQFV3vbE3pRZuAuzWu+SeNfxfP7ZCc57Yc9KvZjImvsOTk1mzMO1xCuHWmLUOAzvmf3fuTYAp6+UzpqKuOHbAVyD7Qzccu
↪e3IX4uh1FmKruSB6FbQAAAIA/
↪8jPxLt3zZ7cwNQhQevwQ3MCU6cgzIUJnZaVdw+G1uWIw6bbYxB60clT4+Z3jajIIx6pWnviQUQqKy3Uj4Ua+N9vnEz5JgMrvxVX
↪xgKuE52xV+B3+gJMA3prSdlRGhuAwQbx9ql/B7PmTdND7ZNw35GOalbMrIY/yw==",
            "ansible_ssh_host_key_ecdsa_public":
↪"AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBP02SVRKRQDJllThy5fVh0Rm8hx8fkKvYzgt73ghPx/
↪FSCWnvzuzrA9yNWR7iBnkcgkpNiUHJwH1Seg3V1NTZ/Y=",
            "ansible_ssh_host_key_ed25519_public":
↪"AAAAC3NzaC1lZDI1NTE5AAAAIPcT3RxDxCA1Adc/k+eDRN5IpAkx201rypKJpnydPXLw",
            "ansible_ssh_host_key_rsa_public":
↪"AAAAB3NzaC1yc2EAAAADAQABAAABAQC7ur+mkamaX/Wnsz90mlwca8GxW58ti/UQwqT89rCv12JSlR2v/
↪Crer8b4zcea06EgCP/Z0ow6RF/
↪LxVNEUFlwtkZJ6inXL6WOrNu9BphBuBMy8+f3BqlMMIs4zEQAoESOQssHA66JhQSYdM1cHYAUUtFNmP8Ht9Ik32qpkGPwU2bEau
↪Guv//RegNpErT7apAp/fZ/
↪OdJw6+6cE13AgzXyjcBWkrnHVyvUMB8VWr9ExNKtEwetBYVGVt6CT6icrr4r3ceD+aQDYczzawZIKA+TTjTrLy6l9hpCId81/
↪PywaddJJWqmQNSZHmva+GX",
            "ansible_swapfree_mb": 975,
            "ansible_swaptotal_mb": 975,
            "ansible_system": "Linux",
            "ansible_system_capabilities": [
                "cap_chown",
                "cap_dac_override",
                "cap_dac_read_search",
                "cap_fowner",
                "cap_fsetid",
                "cap_kill",
                "cap_setgid",
                "cap_setuid",
                "cap_setpcap",
                "cap_linux_immutable",
                "cap_net_bind_service",
                "cap_net_broadcast",
                "cap_net_admin",
                "cap_net_raw",
                "cap_ipc_lock",
                "cap_ipc_owner",
                "cap_sys_module",
                "cap_sys_rawio",
                "cap_sys_chroot",
                "cap_sys_ptrace",
                "cap_sys_pacct",
                "cap_sys_admin",
```

```
                "cap_sys_boot",
                "cap_sys_nice",
                "cap_sys_resource",
                "cap_sys_time",
                "cap_sys_tty_config",
                "cap_mknod",
                "cap_lease",
                "cap_audit_write",
                "cap_audit_control",
                "cap_setfcap",
                "cap_mac_override",
                "cap_mac_admin",
                "cap_syslog",
                "cap_wake_alarm",
                "cap_block_suspend",
                "37+ep"
            ],
            "ansible_system_capabilities_enforced": "True",
            "ansible_system_vendor": "QEMU",
            "ansible_uptime_seconds": 107,
            "ansible_user_dir": "/root",
            "ansible_user_gecos": "root",
            "ansible_user_gid": 0,
            "ansible_user_id": "root",
            "ansible_user_shell": "/bin/bash",
            "ansible_user_uid": 0,
            "ansible_userspace_architecture": "x86_64",
            "ansible_userspace_bits": "64",
            "ansible_virtualization_role": "guest",
            "ansible_virtualization_type": "kvm",
            "module_setup": true
        },
        "fqdn": "keen-skunk",
        "ip": "10.10.0.7",
        "name": "keen-skunk",
        "state": "operational",
        "username": "ansible"
    },
    "id": "0f26c53a-4ce6-4fdd-9e4b-ed7400abf8eb",
    "initiator_id": null,
    "model": "server",
    "time_deleted": 0,
    "time_updated": 1478174236,
    "version": 2
}
```


Field description

Field	Description
cluster_id	ID of the cluster which has this server.
facts	Ansible facts for that server.
fqdn	FQDN of the server.
name	Human-readable name of the server.
state	State of the server (operational, off etc)
username	Username which Ansible uses to connect to this server.

Possible states:

State	Description
operational	Server is up and running.
off	Server was excluded from Decapod.
maintenance_no_reconfig	Server is in maintenance, but no cluster reconfiguration is required.
maintenance_reconfig	Server is in maintenance, cluster reconfiguration is required.

Playbook Configuration

Every playbook requires configuration. This model presents such configuration. On create of playbook configuration, Decapod generates config for given server list and playbook according to best practices. It just proposes a good config, user always may update it.

JSON Schema

```
{
    "name": {"$ref": "#/definitions/non_empty_string"},
    "playbook_id": {"$ref": "#/definitions/non_empty_string"},
    "cluster_id": {"$ref": "#/definitions/uuid4"},
    "configuration": {"type": "object"}
}
```

Real-world Example

```
{
    "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"
                        },
                        "10.10.0.4": {
                            "ansible_user": "ansible",
                            "devices": [
                                "/dev/vdb"
                            ],
                            "monitor_interface": "ens3"
                        },
                        "10.10.0.7": {
                            "ansible_user": "ansible",
                            "devices": [
                                "/dev/vdd",
                                "/dev/vde",
                                "/dev/vdc",
                                "/dev/vdb"
                            ],
                            "monitor_interface": "ens3"
                        },
                        "10.10.0.8": {
                            "ansible_user": "ansible",
                            "devices": [
                                "/dev/vdd",
                                "/dev/vde",
                                "/dev/vdc",
                                "/dev/vdb"
                            ],
                            "monitor_interface": "ens3"
                        }
                    }
                },
                "clients": [],
                "iscsi_gw": [],
                "mdss": [],
                "mons": [
                    "10.10.0.2"
                ],
                "nfss": [],
                "osds": [
                    "10.10.0.7",
```

```
                    "10.10.0.8",
                    "10.10.0.3",
                    "10.10.0.4"
                ],
                "rbdmirrors": [],
                "restapis": [
                    "10.10.0.2"
                ],
                "rgws": []
            }
        },
        "name": "deploy",
        "playbook_id": "cluster_deploy"
    },
    "id": "fd76cea9-3efa-4432-854c-fee30ca79ddb",
    "initiator_id": "9d010f3f-2ec0-4079-ae8c-f46415e2530c",
    "model": "playbook_configuration",
    "time_deleted": 0,
    "time_updated": 1478174220,
    "version": 2
}
```


Field description

Field	Description
cluster_id	ID of the cluster to deploy.
configuration	Configuration of the playbook.
name	Name of the playbook configuration.
playbook_id	ID of the playbook to use.

Configuration differs from one playbook to another. Please check documentation on playbook plugins (TODO) to get a meaning of each configuration option.

Execution

Execution is the model, which incapsulates data about execution of certain playbook configuration on the cluster. You may consider it as a run of **`ansible-playbook`**.

JSON Schema

```
{
    "playbook_configuration": {
        "type": "object",
        "additionalProperties": false,
        "required": ["id", "version"],
        "properties": {
            "id": {"$ref": "#/definitions/uuid4"},
            "version": {"$ref": "#/definitions/positive_integer"}
        }
    },
    "state": {"$ref": "#/definitions/non_empty_string"}
}
```

Real-world Example


```
{
    "data": {
        "playbook_configuration": {
            "id": "fd76cea9-3efa-4432-854c-fee30ca79ddb",
            "version": 2
        },
        "state": "failed"
    },
    "id": "6f016e18-97c4-4069-9e99-70862d98e46a",
    "initiator_id": null,
    "model": "execution",
    "time_deleted": 0,
    "time_updated": 1478175025,
    "version": 3
}
```


Field description

Field	Description
playbook_configuration	Information about ID and version of used playbook configuration.
state	State of execution (failed, completed etc)

Possible states:

State	Description
created	Execution was created but not started yet.
started	Execution is in progress.
completed	Execution was completed successfuly.
failed	Execution was failed.
canceling	Canceling of execution is in progress.
canceled	Execution has been canceled.

Execution Step

This is a model of step of playbook execution. Step is a granular task of configuration management system.

JSON Schema


```
{
    "error": {"type": "object"},
    "execution_id": {"$ref": "#/definitions/uuid4"},
    "name": {"$ref": "#/definitions/non_empty_string"},
    "result": {"$ref": "#/definitions/non_empty_string"},
    "role": {"$ref": "#/definitions/non_empty_string"},
    "server_id": {"$ref": "#/definitions/uuid4"},
    "time_started": {"$ref": "#/definitions/positive_integer"},
    "time_finished": {"$ref": "#/definitions/positive_integer"}
}
```

Real-world Example


```
{
    "data": {
        "error": {},
        "execution_id": "6f016e18-97c4-4069-9e99-70862d98e46a",
        "name": "set config and keys paths",
        "result": "skipped",
        "role": "ceph-restapi",
        "server_id": "3ee25709-215d-4f51-8348-20b4e7390fdb",
        "time_finished": 1478175019,
        "time_started": 1478175019
    },
    "id": "581b292b3ceda10087ab8d41",
    "initiator_id": "6f016e18-97c4-4069-9e99-70862d98e46a",
    "model": "execution_step",
    "time_deleted": 0,
    "time_updated": 1478175019,
    "version": 1
}
```


Field description

Field	Description
error	Error data from Ansible
execution_id	ID of execution made
name	Name of the task which was executed
result	Result of the task execution (failed, ok, ...).
role	Role which task belongs to.
server_id	ID of the server where task was performed.
time_started	UNIX timestamp when task was started.
time_finished	UNIX timestamp when task was finished.

Possible states:

State	Description
ok	Task was executed without any problems.
skipped	Task execution was skipped.
failed	Task was failed.
unreachable	Task was not executed because remote host is unreachable.

Token

Token model presents an authentication token. Token is a string which should be put in **Authorization** header of every request and Decapod API uses it as an authentication mean for operations.

`version` is rudimentary field here and kept for consistency. Do not rely on this field, it always equals 1.

JSON Schema


```
{
    "user": {"type": "User Model"}
```

```
    "expires_at": {"$ref": "#/definitions/positive_integer"}
}
```

Real-world Example

```
{
    "data":{
        "expires_at":1479455919,
        "user":{
            "data":{
                "email":"noreply@example.com",
                "full_name":"Root User",
                "login":"root",
                "role_id":"4f96f3b0-85e5-4735-8c97-34fbef157c9d"
            },
            "id":"ee3944e8-758e-45dc-8e9e-e220478e442c",
            "initiator_id":null,
            "model":"user",
            "time_deleted":0,
            "time_updated":1479295535,
            "version":1
        }
    },
    "id":"cc6cf706-2f26-4975-9885-0d9c234491b2",
    "initiator_id":"ee3944e8-758e-45dc-8e9e-e220478e442c",
    "model":"token",
    "time_deleted":0,
    "time_updated":1479454119,
    "version":1
}
```

Field description

Field	Description
expires_at	UNIX timestamp of moment when this token will be considered as expired.
user	Expanded model of user logged in.

Usage example

As mentioned in *decapodlib API*, Decapod provides RPC client for interaction with remote API. This communication is done using HTTP/HTTPS protocols and client, mostly, works as a thin layer between API and your code. RPC client uses Requests library to manage keep-alive connection to API and does transparent authentication so you do not need to worry about explicit session objects or explicit loggin in/loggin out from API.

This is short a short tutorial which shows you complete workflow: from creating new user and role to deployment of Ceph cluster.

Before doing so, let's do some assumptions:

1. You have Decapod library up and running
2. You already have a bunch of future Ceph nodes registered in Decapod

Let's assume, that all those requirements were fullfiled:

1. Decapod API is placed on IP **10.10.0.1**. HTTP endpoint of Decapod is placed on port **8080**, HTTPS - **8081**.
2. Default account is created. Login is **root**, password is **root**.

Installation

Installation of decapod API library can be done in 2 ways: using wheel and from source code directly.

To install from wheel, do following:

```
$ pip install decapodlib-*-py2.py3-none-any.whl
```

Note: Please be noticed that naming is following to **PEP 0425** which is mandatory for wheel format (**PEP 0427**). This means, that this package is universal for both Python2 and Python3 (the same is true for CLI package) also.

decapodlib and decapodcli are both support Python >= 2.7 and Python >= 3.3.

To install from source code, please do following:

```
$ git clone --recursive --depth 1 https://github.com/Mirantis/ceph-lcm.git
$ cd ceph-lcm/decapodlib
$ python setup.py install
```

Initialize client

Decapod uses versioning for its API. Current up to date version is 1.

Every client version is defined in *decapodlib.client* module. If you want to use version 1, just pick *decapodlib.client.V1Client*. If you want latest and greatest one, just pick *decapodlib.Client* - this is an alias to the latest version.

If you want to use HTTP, just initialize client like this:

```
>>> client = decapodlib.Client("http://10.10.0.1:8080", "root", "root")
```

and if you want HTTPS:

```
>>> client = decapodlib.Client("https://10.10.0.1:8081", "root", "root")
```

Note: If you use HTTPS with self-signed certificate, please use verify option to define certificate verification strategy (by default verification is *enabled*):

```
>>> client = decapodlib.Client("http://10.10.0.1:8081", "root", "root", verify=False)
```

Please refer to documentation of *decapodlib.client.V1Client* to get details about options on client initialization.

Create new user

Now let's create new user with new role. If you already have a role to assign, you can do it on user creation, but to have this tutorial complete, let's do it in several steps.

To please check signature of *decapodlib.client.V1Client.create_user()* method.

```
>>> user = client.create_user("mylogin", "myemail@mydomain.com", "Jane Doe")
>>> print(user["id"])
... "b6631e30-94c8-44dd-b990-1662f3e28788"
>>> print(user["data"]["login"])
... "mylogin"
... print(user["data"]["role_id"])
... None
```

So, new user is created. To get example of the user model, please check *User*.

Please be noticed, that no password is set on user create. User will get his password in his email after creating of user. If she wants, she may change it later, resetting the password.

Let's assume, that user's password is `mypassword`.

Note: As mentioned in models, decapod API returns JSONs and client works with parsed JSONs. No models or similar datastructures are used, just parsed JSONs, so except to get lists and dicts from RPC client responses.

Create new role

You may consider Role as a named set of permissions. To get a list of permissions, please use `decapodlib.V1Client.get_permissions()` method.

```
>>> permissions = client.get_permissions()
>>> print({perm["name"] for perm in permissions["items"]})
... {"api", "permissions"}
```

Let's create role, which can only view items, but cannot do any active actions:

```
>>> playbook_permissions = []
>>>
>>> api_permissions = []
>>> api_permissions.append("view_cluster")
>>> api_permissions.append("view_cluster_versions")
>>> api_permissions.append("view_cluster_versions")
>>> api.permissions.append("view_execution")
>>> api.permissions.append("view_execution_steps")
>>> api.permissions.append("view_execution_version")
>>> api.permissions.append("view_playbook_configuration")
>>> api.permissions.append("view_playbook_configuration_version")
>>> api.permissions.append("view_role")
>>> api.permissions.append("view_role_versions")
>>> api.permissions.append("view_server")
>>> api.permissions.append("view_server_versions")
>>> api.permissions.append("view_user")
>>> api.permissions.append("view_user_versions")
>>>
>>> our_permissions = {"playbook": playbook_permissions, "api": api_permissions}
>>>
>>> new_role = client.new_role("viewer", our_permissions)
>>> print(new_role["id"])
... "ea33fc23-8679-4d57-af53-dff960da7021"
```

Assign user with role

To assign our *viewer* role to *mylogin* user, we need to update her. Updating in decapod is slightly different to update process in other libraries. Decapod does not do any update in place, it creates new version of the same entity. So updates and deletes doing progression of the same value and it is possible to access any versions were made in Decapod using API.

Important: To update model, we need to update its *data* fieldset (please check *Basic model* for details). Do not update any field except of *data*, you will get *400 Bad Request* on such attempt.

```
>>> user["data"]["role_id"] = new_role["id"]
>>> updated_user = client.update_user(user)
>>> print(user["version"])
... 1
>>> print(updated_user["version"])
... 2
>>> print(updated_user["data"]["role_id"] == new_role["id"])
... True
```

Delete user

Now it is a time to delete this user because it was created for illustrative purposes only.

```
>>> deleted_user = client.delete_user(user["id"])
>>> print(deleted_user["version"])
... 3
>>> print(deleted_user["time_deleted"])
... 1479379541
```

The thing is: as mentioned before, no actual *deletion* is done in Decapod, user is archived but not removed from database. It is marked with tombstone, **time_deleted** which is UNIX timestamp, when deletion was made. If user is active, then **time_deleted** is **0**, otherwise it equals to timestamp when deletion was made.

If user model was deleted, it is not possible to login as such user, his access tokens are revoked. It is also not possible to create any modification with such model. Deleted is deleted.

Since deletion does not do any removing from DB, you may consider that process as a combination of archivation and sealing.

Deploy Ceph cluster

Now it is a time to deploy actual Ceph cluster. So, we need to do following:

1. Create new cluster model
2. Create new playbook configuration to deploy that cluster
3. Run execution of that playbook configuration.

Create new cluster model

To deploy new cluster, first we have to create model for that. You may interpret cluster as a named holder for actual Ceph configuration.

```
>>> cluster = client.create_cluster("ceph")
```

Also, it is possible to delete cluster right now with *decapodlib.client.V1Client.delete_cluster* because it has no assigned servers. If cluster has servers assigned, it is not possible to delete it.

Create new playbook configuration

Playbook configuration is a settings for playbook to be executed on given set of servers. To get playbooks, execute *decapodlib.client.V1Client.get_playbooks()* (please check method documentation for example of results).

```
>>> playbooks = client.get_playbooks()
```

For now, we are interested in `cluster_deploy` playbook. It states that it requires the server list. Some playbooks require explicit server list, some - don't. This is context dependend. For example, if you want to purge whole cluster with `purge_cluster` playbook, it makes no sense to specify all servers: purginig cluster affects all servers in this cluster, so playbook configuration will be created for all servers in such cluster.

To deploy clusters, we have to specify servers. To get a list of active servers, just use appropriate `decapodlib.V1Client.get_servers()` method:

```
>>> servers = client.get_servers()
```

To run playbooks, we need only IDs of servers. For simplicity of tutorial, let's assume that we want to use all known servers for that cluster.

```
>>> server_ids = [server["id"] for server in servers]
```

Not everything is ready for creating our playbook configuration.

```
>>> config = client.create_playbook_configuration("cephdeploy", cluster["id"],
↪"cluster_deploy", server_ids)
```

Done, configuration is created. Please check *Playbook Configuration* to get description of configuration options. If you want to modify something (e.g. add another servers as monitors), use *decapodlib.client.V1Client.update_playbook_configuration()* method.

Execute playbook configuration

After you have good enough playbook configuration, it is a time to execute it.

```
>>> execution = client.create_execution(config["id"], config["version"])
```

Note: Please pay attention that you need both playbook configuration ID and version. This is done intentionally because you may want to execute another version of configuration.

When execution is created, it does not start immediately. API service creates task for controller service in UNIX spooling style and controller starts to execute it if it is possible. Decapod uses server locking to avoid collisions in playbook executions, so execution will start only when locks for all required servers can be acquired.

You can check status of execution by requesting model again.

```
>>> execution = client.get_execution(execution["id"])
>>> print(execution["data"]["state"])
>>> "started"
```

When execution is started, you can track it's steps using *decapodlib.client.V1Client.get_execution_steps()* method.

```
>>> steps = client.get_execution_steps(execution["id"])
```

This will return user a models of execution steps for a following execution. When execution is finished, it is also possible to request whole log of execution in plain text (basically, it is just an stdout on **ansible-playbook**).

```
>>> log = client.get_execution_log(execution["id"])
```

Execution is completed when its state either `completed` or `failed`. Completed means that everything is OK, failed - something went wrong.

decapodlib API

decapodlib is a library to work with Decapod API. Also, it provides user with a bunch of additional convenient utilities.

Usage example

API

decapodlib	Library to work with Decapod API.
decapodlib.client	This module contains implementation of RPC client for Decapod API.
decapodlib.auth	This module contains implementation of authorization for Decapod API.
decapodlib.exceptions	Exceptions raised in decapodlib.
decapodlib.cloud_config	This module has routines to help user to build user-data configs for cloud-init.

decapodlib

Library to work with Decapod API.

Top level module provides a list of shortcuts to use with Decapod. Right now, it has only current *decapodlib.Client* implementation as *decapodlib.Client*.

`decapodlib.`**`Client`**

An actual version of JSON client for Decapod.

alias of `V1Client`

decapodlib.client

This module contains implementation of RPC client for Decapod API.

Decapod client *Client* is a simple RPC client and thin wrapper for the requests library which allows end user to work with remote API without worrying about connections and endpoints.

RPC client itself manages authorization (therefore you have to supply it with user/password pair on initialization) so there is no need in explicit session objects but if you do not like that way, you may always relogin explicitly.

Usage example:

```
client = Client(url="http://localhost", login="root", password="root")
```

This will initialize new client. Initialization does not imply immediate login, login would be occured thread-safely on the first real method execution.

```
users = client.get_users()
```

This will return end user a list with active users in Decapod.

```
[
    {
        "data": {
            "email": "noreply@example.com",
            "full_name": "Root User",
            "login": "root",
            "role_id": "37fb532f-2620-4e0d-80e6-b68ed6988a6d"
        },
        "id": "6567c2ab-54cc-40b7-a811-6147a3f3ea83",
        "initiator_id": null,
        "model": "user",
        "time_deleted": 0,
        "time_updated": 1478865388,
        "version": 1
    }
]
```

Incoming JSON will be parsed. If it is not possible, *decapodlib.exceptions.DecapodError* will be raised.

class decapodlib.client.**Client**(*url*, *login*, *password*, *timeout=None*, *verify=True*, *certificate_file=None*)

A base RPC client model.

Parameters

- **url** (*str*) – URL of Decapod API (*without* version prefix like /v1).
- **login** (*str*) – Login of user in Decapod.
- **password** (*str*) – Password of user in Decapod.
- **timeout** (int or None) – Timeout for remote requests. If None is set, default socket timeout (e.g which is set by socket.setdefaulttimeout()) will be used.
- **verify** (*bool*) – If remote URL implies SSL, then using this option client will check SSL certificate for validity.
- **certificate_file** (str or None) – If SSL works with client certificate, this option sets the path to such certificate. If None is set, then it implies that no client certificate should be used.

AUTH_CLASS = None

Base class for authenication.

class decapodlib.client.**V1Client**(*url*, *login*, *password*, *timeout=None*, *verify=True*, *certificate_file=None*)

Implemetation of *decapodlib.client.Client* which works with API version 1.

Please check parameters for *decapodlib.client.Client* class.

Note: All `**kwargs` keyword arguments here are the same as `requests.Session.request()` takes.

AUTH_CLASS

alias of `V1Auth`

cancel_execution(*execution_id*, ***kwargs*)

This method cancels existing execution.

This method does `DELETE /v1/execution/` endpoint call.

Parameters **execution_id** (*str*) – UUID4 (**RFC 4122**) in string form of execution's ID.

Returns Canceled execution model.

Return type dict

Raises

- ***decapodlib.exceptions.DecapodError*** – if not possible to connect to API.
- ***decapodlib.exceptions.DecapodAPIError*** – if API returns error response.

create_cluster(*name*, ***kwargs*)

This method creates new cluster model.

This method does `POST /v1/cluster/` endpoint call.

Parameters **name** (*str*) – Name of the cluster.

Returns New cluster model.

Return type dict

Raises

- ***decapodlib.exceptions.DecapodError*** – if not possible to connect to API.
- ***decapodlib.exceptions.DecapodAPIError*** – if API returns error response.

create_execution(*playbook_configuration_id*, *playbook_configuration_version*, ***kwargs*)

This method creates new execution model.

This method does `POST /v1/execution/` endpoint call.

Parameters

- **playbook_configuration_id** (*str*) – UUID4 (**RFC 4122**) in string form of playbook configuration's ID.
- **playbook_configuration_version** (*int*) – Version of playbook configuration model.

Returns New execution model.

Return type dict

Raises

- ***decapodlib.exceptions.DecapodError*** – if not possible to connect to API.
- ***decapodlib.exceptions.DecapodAPIError*** – if API returns error response.

create_playbook_configuration(*name*, *cluster_id*, *playbook_id*, *server_ids*, *hints=None*, *run_after=False*, ***kwargs*)

This method creates new playbook configuration model.

This method does `POST /v1/playbook_configuration/` endpoint call.

Hints for playbook configuration are the list of optional parameters for creating playbook configuration. It has to be the list key/value parameters obtained from *decapodlib.client.V1Client.get_playbooks()*.

```
[
    {
        "id": "dmcrypt",
        "value": true
    }
]
```

Parameters

- **name** (*str*) – Name of the playbook configuration.
- **cluster_id** (*str*) – UUID4 (**RFC 4122**) in string form of cluster's ID
- **playbook_id** (*str*) – ID of playbook to use.
- **server_ids** ([str, ...]) – List of server UUID4 (**RFC 4122**) in string form of server model IDs.
- **hints** (*list*) – List of hints for playbook configuration.
- **run_after** (*bool*) – Run playbook configuration after create.

Returns New cluster model.

Return type dict

Raises

- ***decapodlib.exceptions.DecapodError*** – if not possible to connect to API.
- ***decapodlib.exceptions.DecapodAPIError*** – if API returns error response.

create_role(*name*, *permissions*, ***kwargs*)

This method creates new role model.

This method does `POST /v1/role` endpoint call.

This method accepts parameter `permissions`. This is a list of permissions like that:

```
[
    {
        "name": "playbook",
        "permissions": [
            "add_osd",
            "cluster_deploy",
            "hello_world",
            "purge_cluster",
            "remove_osd"
        ]
    },
    {
        "name": "api",
        "permissions": [
```

```
            "create_cluster",
            "create_execution",
            "create_playbook_configuration",
            "create_role",
            "create_server",
            "create_user",
            "delete_cluster",
            "delete_execution",
            "delete_playbook_configuration",
            "delete_role",
            "delete_server",
            "delete_user",
            "edit_cluster",
            "edit_playbook_configuration",
            "edit_role",
            "edit_server",
            "edit_user",
            "view_cluster",
            "view_cluster_versions",
            "view_execution",
            "view_execution_steps",
            "view_execution_version",
            "view_playbook_configuration",
            "view_playbook_configuration_version",
            "view_role",
            "view_role_versions",
            "view_server",
            "view_server_versions",
            "view_user",
            "view_user_versions"
        ]
    }
]
```

So, each element is a dict with `name` and `permissions` field.

Parameters

- **name** (*str*) – Name of the role.
- **permissions** (*list*) – A list of permissions. Please check example above.

Returns New role model.

Return type dict

Raises

- ***decapodlib.exceptions.DecapodError*** – if not possible to connect to API.
- ***decapodlib.exceptions.DecapodAPIError*** – if API returns error response.

create_server(*server_id*, *host*, *username*, ***kwargs*)

This method creates new server model.

This method does `POST /v1/server/` endpoint call.

> **Warning:** You should avoid to use this method manually. Servers must be discovered using cloud-init based discovery mechanism.

Parameters

- **server_id** (*str*) – Unique ID of server.
- **host** (*str*) – Hostname of the server (should be accessible by Decapod). It is better to have FQDN here.
- **username** (*str*) – The name of the user for Ansible on this server. Decapod will use Ansible which SSH to machine with hostname given in `host` parameter and that username.

Returns New server model.

Return type dict

Raises

- ***decapodlib.exceptions.DecapodError*** – if not possible to connect to API.
- ***decapodlib.exceptions.DecapodAPIError*** – if API returns error response.

create_user(*login*, *email*, *full_name=''*, *role_id=None*, ***kwargs*)

This method creates new user model.

This method does `POST /v1/user/` endpoint call.

Parameters **name** (*str*) – Name of the user.

Returns New user model.

Return type dict

Raises

- ***decapodlib.exceptions.DecapodError*** – if not possible to connect to API.
- ***decapodlib.exceptions.DecapodAPIError*** – if API returns error response.

delete_cluster(*cluster_id*, ***kwargs*)

This methods deletes cluster model.

Please be noticed that no real delete is performed, cluster model is marked as deleted (`time_deleted > 0`) and model will be skipped from listing, updates are forbidden.

This method does `DELETE /v1/cluster/` endpoint call.

Parameters **cluster_id** (*str*) – UUID4 (**RFC 4122**) in string form of cluster's ID

Returns Deleted cluster model.

Return type dict

Raises

- ***decapodlib.exceptions.DecapodError*** – if not possible to connect to API.
- ***decapodlib.exceptions.DecapodAPIError*** – if API returns error response.

delete_playbook_configuration(*playbook_configuration_id*, ***kwargs*)

This method deletes playbook configuration model.

Please be noticed that no real delete is performed, playbook configuration model is marked as deleted (`time_deleted > 0`) and model will be skipped from listing, updates are forbidden.

This method does `DELETE /v1/playbook_configuration/` endpoint call.

Parameters **playbook_configuration_id** (*str*) – UUID4 (**RFC 4122**) in string form of playbook configuration's ID

Returns Deleted playbook configuration model.

Return type dict

Raises

- ***decapodlib.exceptions.DecapodError*** – if not possible to connect to API.
- ***decapodlib.exceptions.DecapodAPIError*** – if API returns error response.

delete_role(*role_id*, ***kwargs*)
This methods deletes role model.

Please be noticed that no real delete is performed, role model is marked as deleted (`time_deleted > 0`) and model will be skipped from listing, updates are forbidden.

This method does `DELETE /v1/role/` endpoint call.

Parameters **role_id** (*str*) – UUID4 (**RFC 4122**) in string form of role's ID

Returns Deleted role model.

Return type dict

Raises

- ***decapodlib.exceptions.DecapodError*** – if not possible to connect to API.
- ***decapodlib.exceptions.DecapodAPIError*** – if API returns error response.

delete_server(*server_id*, ***kwargs*)
This methods deletes server model.

Please be noticed that no real delete is performed, server model is marked as deleted (`time_deleted > 0`) and model will be skipped from listing, updates are forbidden.

This method does `DELETE /v1/server/` endpoint call.

Parameters **server_id** (*str*) – UUID4 (**RFC 4122**) in string form of server's ID

Returns Deleted server model.

Return type dict

Raises

- ***decapodlib.exceptions.DecapodError*** – if not possible to connect to API.
- ***decapodlib.exceptions.DecapodAPIError*** – if API returns error response.

delete_user(*user_id*, ***kwargs*)
This methods deletes user model.

Please be noticed that no real delete is performed, user model is marked as deleted (`time_deleted > 0`) and model will be skipped from listing, updates are forbidden.

This method does `DELETE /v1/user/` endpoint call.

Parameters **user_id** (*str*) – UUID4 (**RFC 4122**) in string form of user's ID

Returns Deleted user model.

Return type dict

Raises

- ***decapodlib.exceptions.DecapodError*** – if not possible to connect to API.
- ***decapodlib.exceptions.DecapodAPIError*** – if API returns error response.

get_cinder_integration(*cluster_id*, *root='/etc/ceph'*, ***kwargs*)
This method fetches data for integration with Cinder.

This method does `GET /v1/cinder_integration/{cluster_id}` endpoint call.

Parameters

- **cluster_id** (*str*) – UUID4 (RFC 4122) in string form of cluster's ID
- **root** (*str*) – Root on file system where files should be stored.

Returns Integration data

Return type dict

Raises

- ***decapodlib.exceptions.DecapodError*** – if not possible to connect to API.
- ***decapodlib.exceptions.DecapodAPIError*** – if API returns error response.

get_cluster(*cluster_id*, ***kwargs*)
This method fetches a single cluster model (latest version) from API.

This method does `GET /v1/cluster/{cluster_id}` endpoint call.

Parameters **cluster_id** (*str*) – UUID4 (RFC 4122) in string form of cluster's ID

Returns Cluster model of latest available version

Return type dict

Raises

- ***decapodlib.exceptions.DecapodError*** – if not possible to connect to API.
- ***decapodlib.exceptions.DecapodAPIError*** – if API returns error response.

get_cluster_version(*cluster_id*, *version*, ***kwargs*)
This method fetches a certain version of particular cluster model.

This method does `GET /v1/cluster/{cluster_id}/version/{version}` endpoint call.

Parameters

- **cluster_id** (*str*) – UUID4 (RFC 4122) in string form of cluster's ID
- **version** (*int*) – The number of version to fetch.

Returns Cluster model of certain version.

Return type dict

Raises

- ***decapodlib.exceptions.DecapodError*** – if not possible to connect to API.
- ***decapodlib.exceptions.DecapodAPIError*** – if API returns error response.

get_cluster_versions(*cluster_id*, *query_params*, ***kwargs*)
This method fetches a list of all versions for a certain cluster model.

This method does `GET /v1/cluster/{cluster_id}/version/` endpoint call.

Parameters **cluster_id** (*str*) – UUID4 (RFC 4122) in string form of cluster's ID

Returns List of cluster versions for cluster with ID `cluster_id`.

Return type list

Raises

- ***decapodlib.exceptions.DecapodError*** – if not possible to connect to API.
- ***decapodlib.exceptions.DecapodAPIError*** – if API returns error response.

get_clusters(*query_params*, ***kwargs*)
This method fetches a list of latest cluster models from API.

By default, only active clusters will be listed.

This method does `GET /v1/cluster` endpoint call.

Returns List of latest cluster models.

Return type list

Raises

- ***decapodlib.exceptions.DecapodError*** – if not possible to connect to API.
- ***decapodlib.exceptions.DecapodAPIError*** – if API returns error response.

get_execution(*execution_id*, ***kwargs*)
This method fetches a single execution model (latest version) from API.

This method does `GET /v1/execution/{execution_id}` endpoint call.

Parameters **execution_id** (*str*) – UUID4 (**RFC 4122**) in string form of execution's ID

Returns Execution model of latest available version

Return type dict

Raises

- ***decapodlib.exceptions.DecapodError*** – if not possible to connect to API.
- ***decapodlib.exceptions.DecapodAPIError*** – if API returns error response.

get_execution_log(*execution_id*, ***kwargs*)
This method fetches text execution log for a certain execution.

Execution log is a raw Ansible execution log, that one, which is generated by **ansible-playbook** program.

This method does `GET /v1/execution/{execution_id}/log` endpoint call.

Parameters **execution_id** (*str*) – UUID4 (**RFC 4122**) in string form of execution's ID.

Returns List of execution steps.

Return type list

Raises

- ***decapodlib.exceptions.DecapodError*** – if not possible to connect to API.
- ***decapodlib.exceptions.DecapodAPIError*** – if API returns error response.

get_execution_steps(*execution_id*, *query_params*, ***kwargs*)
This method fetches step models of the execution.

This method does `GET /v1/execution/{execution_id}/steps` endpoint call.

Parameters **execution_id** (*str*) – UUID4 (**RFC 4122**) in string form of execution's ID.

Returns List of execution steps.

Return type list

Raises

- ***decapodlib.exceptions.DecapodError*** – if not possible to connect to API.
- ***decapodlib.exceptions.DecapodAPIError*** – if API returns error response.

get_execution_version(*execution_id*, *version*, ***kwargs*)

This method fetches a certain version of particular execution model.

This method does `GET /v1/execution/{execution_id}/version/{version}` endpoint call.

Parameters

- **execution_id** (*str*) – UUID4 (**RFC 4122**) in string form of execution's ID
- **version** (*int*) – The number of version to fetch.

Returns Execution model of certain version.

Return type dict

Raises

- ***decapodlib.exceptions.DecapodError*** – if not possible to connect to API.
- ***decapodlib.exceptions.DecapodAPIError*** – if API returns error response.

get_execution_versions(*execution_id*, *query_params*, ***kwargs*)

This method fetches a list of all versions for a certain execution model.

This method does `GET /v1/execution/{execution_id}/version/` endpoint call.

Parameters **execution_id** (*str*) – UUID4 (**RFC 4122**) in string form of execution's ID

Returns List of execution versions for execution with ID `execution_id`.

Return type list

Raises

- ***decapodlib.exceptions.DecapodError*** – if not possible to connect to API.
- ***decapodlib.exceptions.DecapodAPIError*** – if API returns error response.

get_executions(*query_params*, ***kwargs*)

This method fetches a list of latest execution models from API.

This method does `GET /v1/execution` endpoint call.

Returns List of latest execution models.

Return type list

Raises

- ***decapodlib.exceptions.DecapodError*** – if not possible to connect to API.
- ***decapodlib.exceptions.DecapodAPIError*** – if API returns error response.

get_info(***kwargs*)

This method fetches basic data from Decapod API.

It makes no sense to use this method for anything, it is just a healthcheck that service actually works.

Example of result:

```
{
    "time": {
        "local": "2016-11-16T12:46:55.868153",
        "unix": 1479300415,
        "utc": "2016-11-16T12:46:55.868220"
    },
    "version": "0.1.0"
}
```


Important: This method is basically the only one you may access being not logged in.

Returns Something

Return type dict

Raises

- ***decapodlib.exceptions.DecapodError*** – if not possible to connect to API.
- ***decapodlib.exceptions.DecapodAPIError*** – if API returns error response.

get_permissions(***kwargs*)

This method lists exisiting permissions in system. Not those, which available for current user, but overall ones. This is mostly required if you compose new role.

This method does `GET /v1/permission` endpoint call.

Example of result:


```
{
    "items": [
        {
            "name": "api",
            "permissions": [
                "create_cluster",
                "create_execution",
                "create_playbook_configuration",
                "create_role",
                "create_server",
                "create_user",
                "delete_cluster",
                "delete_execution",
                "delete_playbook_configuration",
                "delete_role",
                "delete_server",
                "delete_user",
                "edit_cluster",
                "edit_playbook_configuration",
                "edit_role",
                "edit_server",
                "edit_user",
                "view_cluster",
                "view_cluster_versions",
                "view_execution",
                "view_execution_steps",
                "view_execution_version",
                "view_playbook_configuration",
```

```
                "view_playbook_configuration_version",
                "view_role",
                "view_role_versions",
                "view_server",
                "view_server_versions",
                "view_user",
                "view_user_versions"
            ]
        },
        {
            "name": "playbook",
            "permissions": [
                "add_osd",
                "cluster_deploy",
                "hello_world",
                "purge_cluster",
                "remove_osd"
            ]
        }
    ]
}
```

Note: As you can see, there are 2 types of permissions in Decapod:

1.api

2.playbook

api permissions are responsible for accessing API endpoints. If user wants to access some API endpoint, he has to have appropriate permission in his role. Some endpoints require several permissions and rule of thumb here is common sense: is user wants to *update* role, he has to have a permission to *view* it.

playbook permissions are slightly different beasts. Each permission allows user to execute a certain playbook.

Returns A list of premissions like those mentioned above

Return type list

Raises

- ***decapodlib.exceptions.DecapodError*** – if not possible to connect to API.
- ***decapodlib.exceptions.DecapodAPIError*** – if API returns error response.

get_playbook_configuration(*playbook_configuration_id*, ***kwargs*)

This method fetches a single playbook configuration model (latest version) from API.

This method does `GET /v1/playbook_configuration/{playbook_configuration_id}` endpoint call.

Parameters **playbook_configuration_id** (*str*) – UUID4 (**RFC 4122**) in string form of playbook configuration's ID.

Returns Playbook configuration model of latest available version.

Return type dict

Raises

- **decapodlib.exceptions.DecapodError** – if not possible to connect to API.
- **decapodlib.exceptions.DecapodAPIError** – if API returns error response.

get_playbook_configuration_version (*playbook_configuration_id*, *version*, ***kwargs*)
This method fetches a certain version of particular playbook configuration model.

This method does `GET /v1/playbook_configuration/{playbook_configuration_id}/version/{version}` endpoint call.

Parameters

- **playbook_configuration_id** (*str*) – UUID4 (**RFC 4122**) in string form of playbook configuration's ID
- **version** (*int*) – The number of version to fetch.

Returns Playbook configuration model of certain version.

Return type dict

Raises

- **decapodlib.exceptions.DecapodError** – if not possible to connect to API.
- **decapodlib.exceptions.DecapodAPIError** – if API returns error response.

get_playbook_configuration_versions (*playbook_configuration_id*, *query_params*, ***kwargs*)
This method fetches a list of all versions for a certain playbook configuration model.

This method does `GET /v1/playbook_configuration/{playbook_configuration_id}/version/` endpoint call.

Parameters **playbook_configuration_id** (*str*) – UUID4 (**RFC 4122**) in string form of playbook configuration's ID.

Returns List of playbook configuration versions for playbook configuration with ID `playbook_configuration_id`.

Return type list

Raises

- **decapodlib.exceptions.DecapodError** – if not possible to connect to API.
- **decapodlib.exceptions.DecapodAPIError** – if API returns error response.

get_playbook_configurations (*query_params*, ***kwargs*)
This method fetches a list of latest playbook configuration models from API.

By default, only active playbook configurations will be listed.

This method does `GET /v1/playbook_configuration` endpoint call.

Returns List of latest playbook configuration models.

Return type list

Raises

- **decapodlib.exceptions.DecapodError** – if not possible to connect to API.
- **decapodlib.exceptions.DecapodAPIError** – if API returns error response.

get_playbooks (***kwargs*)
This method returns a list of playbooks avaialble for execution.

This method does `GET /v1/playbook` endpoint call.

Example of result:


```
{
    "items": [
        {
            "description": "Adding new OSD to the cluster.",
            "id": "add_osd",
            "name": "Add OSD to Ceph cluster",
            "required_server_list": true,
            "hints": []
        },
        {
            "description": "Ceph cluster deployment playbook.",
            "id": "cluster_deploy",
            "name": "Deploy Ceph cluster",
            "required_server_list": true,
            "hints": [
                {
                    "description": "Setup OSDs with dmcrypt",
                    "id": "dmcrypt",
                    "type": "boolean",
                    "values": []
                }
            ]
        },
        {
            "description": "Example plugin for playbook.",
            "id": "hello_world",
            "name": "Hello World",
            "required_server_list": false
            "hints": []
        },
        {
            "description": "Purge whole Ceph cluster.",
            "id": "purge_cluster",
            "name": "Purge cluster",
            "required_server_list": false,
            "hints": []
        },
        {
            "description": "Remove OSD host from cluster.",
            "id": "remove_osd",
            "name": "Remove OSD host from Ceph cluster",
            "required_server_list": true,
            "hints": []
        }
    ]
}
```


Note: Please remember that `playbook` parameter in `POST /v1/playbook_configuration` is `id` field here.

Returns A list of playbook data.

Return type list

Raises

- **decapodlib.exceptions.DecapodError** – if not possible to connect to API.
- **decapodlib.exceptions.DecapodAPIError** – if API returns error response.

get_role(*role_id*, ***kwargs*)

This method fetches a single role model (latest version) from API.

This method does `GET /v1/role/{role_id}` endpoint call.

Parameters **role_id** (*str*) – UUID4 (**RFC 4122**) in string form of role's ID

Returns Role model of latest available version

Return type dict

Raises

- **decapodlib.exceptions.DecapodError** – if not possible to connect to API.
- **decapodlib.exceptions.DecapodAPIError** – if API returns error response.

get_role_self(***kwargs*)

This methods requests model of role of current user.

This method does `GET /v1/role/self/` endpoint call.

Returns Role model of current user.

Return type dict

Raises

- **decapodlib.exceptions.DecapodError** – if not possible to connect to API.
- **decapodlib.exceptions.DecapodAPIError** – if API returns error response.

get_role_version(*role_id*, *version*, ***kwargs*)

This method fetches a certain version of particular role model.

This method does `GET /v1/role/{role_id}/version/{version}` endpoint call.

Parameters

- **role_id** (*str*) – UUID4 (**RFC 4122**) in string form of role's ID
- **version** (*int*) – The number of version to fetch.

Returns Role model of certain version.

Return type dict

Raises

- **decapodlib.exceptions.DecapodError** – if not possible to connect to API.
- **decapodlib.exceptions.DecapodAPIError** – if API returns error response.

get_role_versions(*role_id*, *query_params*, ***kwargs*)

This method fetches a list of all versions for a certain role model.

This method does `GET /v1/role/{role_id}/version/` endpoint call.

Parameters **role_id** (*str*) – UUID4 (**RFC 4122**) in string form of role's ID

Returns List of role versions for role with ID `role_id`.

Return type list

Raises

- **decapodlib.exceptions.DecapodError** – if not possible to connect to API.
- **decapodlib.exceptions.DecapodAPIError** – if API returns error response.

get_roles(*query_params*, ***kwargs*)
This method fetches a list of latest role models from API.

By default, only active roles will be listed.

This method does `GET /v1/role` endpoint call.

Returns List of latest role models.

Return type list

Raises

- **decapodlib.exceptions.DecapodError** – if not possible to connect to API.
- **decapodlib.exceptions.DecapodAPIError** – if API returns error response.

get_server(*server_id*, ***kwargs*)
This method fetches a single server model (latest version) from API.

This method does `GET /v1/server/{server_id}` endpoint call.

Parameters **server_id** (*str*) – UUID4 (**RFC 4122**) in string form of server's ID

Returns Server model of latest available version

Return type dict

Raises

- **decapodlib.exceptions.DecapodError** – if not possible to connect to API.
- **decapodlib.exceptions.DecapodAPIError** – if API returns error response.

get_server_version(*server_id*, *version*, ***kwargs*)
This method fetches a certain version of particular server model.

This method does `GET /v1/server/{server_id}/version/{version}` endpoint call.

Parameters

- **server_id** (*str*) – UUID4 (**RFC 4122**) in string form of server's ID
- **version** (*int*) – The number of version to fetch.

Returns Server model of certain version.

Return type dict

Raises

- **decapodlib.exceptions.DecapodError** – if not possible to connect to API.
- **decapodlib.exceptions.DecapodAPIError** – if API returns error response.

get_server_versions(*server_id*, *query_params*, ***kwargs*)
This method fetches a list of all versions for a certain server model.

This method does `GET /v1/server/{server_id}/version/` endpoint call.

Parameters **server_id** (*str*) – UUID4 (**RFC 4122**) in string form of server's ID

Returns List of server versions for server with ID `server_id`.

Return type list

Raises

- ***decapodlib.exceptions.DecapodError*** – if not possible to connect to API.
- ***decapodlib.exceptions.DecapodAPIError*** – if API returns error response.

get_servers(*query_params*, ***kwargs*)

This method fetches a list of latest server models from API.

By default, only active servers will be listed.

This method does `GET /v1/server` endpoint call.

Returns List of latest server models.

Return type list

Raises

- ***decapodlib.exceptions.DecapodError*** – if not possible to connect to API.
- ***decapodlib.exceptions.DecapodAPIError*** – if API returns error response.

get_user(*user_id*, ***kwargs*)

This method fetches a single user model (latest version) from API.

This method does `GET /v1/user/{user_id}` endpoint call.

Parameters **user_id** (*str*) – UUID4 (**RFC 4122**) in string form of user's ID

Returns User model of latest available version

Return type dict

Raises

- ***decapodlib.exceptions.DecapodError*** – if not possible to connect to API.
- ***decapodlib.exceptions.DecapodAPIError*** – if API returns error response.

get_user_self(***kwargs*)

This methods requests model of current user.

This method does `GET /v1/user/self/` endpoint call.

Returns User model of current user.

Return type dict

Raises

- ***decapodlib.exceptions.DecapodError*** – if not possible to connect to API.
- ***decapodlib.exceptions.DecapodAPIError*** – if API returns error response.

get_user_version(*user_id*, *version*, ***kwargs*)

This method fetches a certain version of particular user model.

This method does `GET /v1/user/{user_id}/version/{version}` endpoint call.

Parameters

- **user_id** (*str*) – UUID4 (**RFC 4122**) in string form of user's ID
- **version** (*int*) – The number of version to fetch.

Returns User model of certain version.

Return type dict

Raises

- ***decapodlib.exceptions.DecapodError*** – if not possible to connect to API.
- ***decapodlib.exceptions.DecapodAPIError*** – if API returns error response.

get_user_versions(*user_id*, *query_params*, ***kwargs*)

This method fetches a list of all versions for a certain user model.

This method does `GET /v1/user/{user_id}/version/` endpoint call.

Parameters **user_id** (*str*) – UUID4 (**RFC 4122**) in string form of user's ID

Returns List of user versions for user with ID `user_id`.

Return type list

Raises

- ***decapodlib.exceptions.DecapodError*** – if not possible to connect to API.
- ***decapodlib.exceptions.DecapodAPIError*** – if API returns error response.

get_users(*query_params*, ***kwargs*)

This method fetches a list of latest user models from API.

By default, only active users will be listed.

This method does `GET /v1/user` endpoint call.

Returns List of latest user models.

Return type list

Raises

- ***decapodlib.exceptions.DecapodError*** – if not possible to connect to API.
- ***decapodlib.exceptions.DecapodAPIError*** – if API returns error response.

login(***kwargs*)

This methods logins users into API.

Basically, you do not need to execute this method by yourself, client will implicitly execute it when needed.

This method does `POST /v1/auth` endpoint call.

Returns Model of the Token.

Return type dict

Raises

- ***decapodlib.exceptions.DecapodError*** – if not possible to connect to API.
- ***decapodlib.exceptions.DecapodAPIError*** – if API returns error response.

logout(***kwargs*)

This method logouts users from API (after that security token will be deleted).

Basically, you do not need to execute this method by yourself, client will implicitly execute it when needed.

This method does `DELETE /v1/auth` endpoint call.

Raises

- ***decapodlib.exceptions.DecapodError*** – if not possible to connect to API.
- ***decapodlib.exceptions.DecapodAPIError*** – if API returns error response.

peek_password_reset(*reset_token*, ***kwargs*)

This method checks if password reset with given token is still requested. It does not consume token, it just checks if it is possible or not.

Example of result:

```
{
    "message": "Password reset was requested."
}
```

Parameters **reset_token** (*str*) – Password reset token from email.

Returns A message that password reset was requested.

Return type dict

Raises

- ***decapodlib.exceptions.DecapodError*** – if not possible to connect to API.
- ***decapodlib.exceptions.DecapodAPIError*** – if API returns error response.

put_server(*model_data*, ***kwargs*)

This methods updates server model.

Please be noticed that no real update is performed, just a new version of the same server is created.

This method does `PUT /v1/server/` endpoint call.

Parameters **model_data** (*dict*) – Updated model of the server.

Returns Updated server model.

Return type dict

Raises

- ***decapodlib.exceptions.DecapodError*** – if not possible to connect to API.
- ***decapodlib.exceptions.DecapodAPIError*** – if API returns error response.

request_password_reset(*login*, ***kwargs*)

This method requests password resetting for a user.

Please be noticed that no real password resetting is occured, it just *requesting* password reset. After that, user will receive secret link on his email. If user will proceed that link, he can *actually* reset her password.

This method does `POST /v1/password_reset` endpoint call.

Example of result:

```
{
    "message": "Password reset was requested."
}
```

Parameters **login** (*str*) – Login of user who is required to reset password.

Returns A message that password reset was requested.

Return type dict

Raises

- ***decapodlib.exceptions.DecapodError*** – if not possible to connect to API.

- **decapodlib.exceptions.DecapodAPIError** – if API returns error response.

reset_password(*reset_token*, *new_password*, ***kwargs*)
This method does actual password resetting.

Example of result:

```
{
    "message": "Password has been reset."
}
```

Parameters

- **reset_token** (*str*) – Password reset token from email.
- **new_password** (*str*) – New password for user.

Returns A message that password was reset.

Return type dict

Raises

- **decapodlib.exceptions.DecapodError** – if not possible to connect to API.
- **decapodlib.exceptions.DecapodAPIError** – if API returns error response.

update_cluster(*model_data*, ***kwargs*)
This methods updates cluster model.

Please be noticed that no real update is performed, just a new version of the same cluster is created.

This method does `PUT /v1/cluster/` endpoint call.

Parameters **model_data** (*dict*) – Updated model of the cluster.

Returns Updated cluster model.

Return type dict

Raises

- **decapodlib.exceptions.DecapodError** – if not possible to connect to API.
- **decapodlib.exceptions.DecapodAPIError** – if API returns error response.

update_playbook_configuration(*model_data*, ***kwargs*)
This method updates playbook configuration model.

Please be noticed that no real update is performed, just a new version of the same playbook configuration is created.

This method does `PUT /v1/playbook_configuration/` endpoint call.

Parameters **model_data** (*dict*) – Updated model of the playbook configuration.

Returns Updated playbook configuration model.

Return type dict

Raises

- **decapodlib.exceptions.DecapodError** – if not possible to connect to API.
- **decapodlib.exceptions.DecapodAPIError** – if API returns error response.

update_role(*model_data*, ***kwargs*)
This methods updates role model.

Please be noticed that no real update is performed, just a new version of the same role is created.

This method does `PUT /v1/role/` endpoint call.

Parameters **model_data** (*dict*) – Updated model of the role.

Returns Updated role model.

Return type dict

Raises

- ***decapodlib.exceptions.DecapodError*** – if not possible to connect to API.
- ***decapodlib.exceptions.DecapodAPIError*** – if API returns error response.

update_user(*model_data*, ***kwargs*)
This methods updates user model.

Please be noticed that no real update is performed, just a new version of the same user is created.

This method does `PUT /v1/user/` endpoint call.

Parameters **model_data** (*dict*) – Updated model of the user.

Returns Updated user model.

Return type dict

Raises

- ***decapodlib.exceptions.DecapodError*** – if not possible to connect to API.
- ***decapodlib.exceptions.DecapodAPIError*** – if API returns error response.

decapodlib.auth

This module contains implementation of authorization for Decapod API.

Decapod client uses requests library to access its API so authentication is done using requests's classes. Please check official guide for details.

class decapodlib.auth.**V1Auth**(*client*)
Request authentication provider for Decapod API V1.

The idea of that provider is really simple: it stores authentication token from Decapod API and injects it into proper header on every request. If no token is defined, it will authorize for you transparently using *decapodlib.client.Client* login method.

AUTH_URL = '/v1/auth/'
URL of authentication.

revoke_token()
Resets information about known token.

decapodlib.auth.**no_auth**(*request*)
Trivial authenticator which does no authentication for a request.

decapodlib.exceptions

Exceptions raised in decapodlib.

Please be noticed, that all exception raised from *decapodlib* will be wrapped in *decapodlib.exceptions.DecapodError* or its subclasses.

exception decapodlib.exceptions.**DecapodAPIError**(*response*)
Common error in API.

Parameters **response** (*requests.Response*) – Original response which is a base of that exception.

json
Return this error as parsed JSON.

Example of result:

```
{
    "code": 403,
    "error": "Forbidden",
    "description": "Access was forbidden!"
}
```

exception decapodlib.exceptions.**DecapodError**(*exc*)
Basic error raised in decapodlib.

Parameters **exc** (*Exception*) – Original exception, wrapped in this class.

Original exception is stored in `exception` field.

decapodlib.cloud_config

This module has routines to help user to build user-data configs for cloud-init.

Decapod uses cloud-init to implement server discovery. On each server boot user-data will be executed (you may consider cloud-init as rc.local on steroids).

Basically, it creates several files on the host system and put their execution into host rc.local.

decapodlib.cloud_config.**generate_cloud_config**(*url*, *server_discovery_token*, *public_key*, *username*, *timeout=20*, *no_discovery=False*)
This function generates user-data config (or cloud config) for cloud-init.

Parameters

- **url** (*str*) – URL of Decapod API. This URL should be accessible from remote machine.
- **server_discovery_token** (*str*) – Server discovery token from Decapod config.
- **public_key** (*str*) – SSH public key of Ansible. This key will be placed in `~username/.ssh/authorized_keys`.
- **username** (*str*) – Username of the user, which Ansible will use to access this host.
- **timeout** (*int*) – Timeout of connection to Decapod API.
- **no_discovery** (*bool*) – Generate config with user and packages but no discovery files. It can be used if user wants to add servers manually.

Returns Generated user-data in YAML format.

Return type str

Python Module Index

d

Index

J

L

N

P

R

U

V