Decapod Documentation

Release 1.0.0

Sergey Arkhipov

Mar 14, 2017

Contents

CHAPTER 1

Introduction

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

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

Decapod provides the following functionality:

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

However, Decapod does not cover:

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

See also:

- Ceph
- Ansible

- ceph-ansible community project
- Decapod API reference

CHAPTER 2

Install and configure Decapod

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

Prerequisites

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

1. 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 development or production images. In the development version, the SSH private keys, SSL certificate, and configuration file are pre-generated and placed in the `containerization/files` directory of the source code. To build a production version, you need to have your own configuration file, an SSH private key for Ansible, and an SSL certificate for the web front end.
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 the previous version, apply migrations. This operation is idempotent and you may execute it safely at any time. If a migration was applied,

Decapod will not reapply it again. On the first boot, migrations are required to obtain the root user. Otherwise, Decapod will start 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, use:

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

3. Build Decapod depending on your needs:

Important: In case if you do not have an access to private repository to fetch base images, then you need to set upstreams for base images:

```
$ make docker_registry_use_dockerhub
```

If you decide to switch back, do following:

```
$ make docker_registry_use_internal_ci
```

- Development version. To build the development images, run:

```
$ make build_containers_dev
```

- Production version.

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

```
make copy_example_keys
```

  (b) Build the production version:

```
$ make build_containers
```

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

---

4. Move the Docker images to the target node.

Note: In future, it will be possible to run Decapod services on different machines. However, it is assumed that you have only one machine with Docker and Docker Compose. There may be one build machine and another production one. If you have such a diversity, 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 *Decapod configuration files*.
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 the 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.
```

You can get a list of applied migrations with `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
```

And the details of the certain migration with `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 password of user **root**. Please check *Password Reset* for details.

   Example:

```
$ docker-compose exec -T 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
    }
]
$ docker-compose exec -T admin decapod-admin password-reset -p MYNEWPASSWORD␣
↪e6f28a01-ee7f-4ac8-b1ee-a1a21c3eb182
```

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.

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	Port to bind the HTTP endpoint of Decapod.
DECAPOD_HTTPS_PORT	10000	Port to bind the HTTPS endpoint of 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	Version of Decapod to use. This is the image tag and is set in the registry. `latest` means unreleased (developer version).
DECAPOD_SSH_PRIVATE_KEY	$(pwd)/containerization/files/devconfigs/ansible_ssh_keyfile.pem	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, use real environment variables:

```
$ 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:

- *Decapod configuration files*

Decapod configuration files

Decapod supports a number of configuration files you may want to propagate into the container:

ansible_ssh_keyfile.pem

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 a new one, 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

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.

config.yaml

Configuration file for Decapod. Please check *config.yaml file* for details.

mongodb.pem

SSL/TLS pair of certificate and key, concatenated in one file. Required to use secured connection by MongoDB. For information on how to generate this file, refer to the official documentation. To allow SSL/TLS on client side, verify that config file has the `?ssl=true` parameter in URI. For example, `mongodb://database:27017/db` will not use a secured connection, but `mongodb://database:27017/db?ssl=true` will.

Note: To use database authentication, follow the official guide or the community checklist:

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

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

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

Propagation to containers

Here is the list of mentioned files and their placement in containers. For simplicity, this table uses **docker-compose** service names.

Configuration file	api	controller	admin	frontend	database
ansible_ssh_keyfile.pem		/root/.ssh/id_rsa	/root/.ssh/id_rsa		
ssl.key				/ssl/ssl.key	
ssl.crt				/ssl/ssl.crt	
ssl-dhparam.pem				/ssl/dhparam.pem	
config.yaml	/etc/decapod/config.yaml	/etc/decapod/config.yaml	/etc/decapod/config.yaml		
mongodb.pem					/certs/mongodb.pem
mongodb-ca.crt					/certs/mongodb-ca.crt

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

```
---
version: "2"

services:
  database:
    volumes:
      # SSL certificate for MongoDB
      - ./containerization/files/devconfigs/mongodb-ca.crt:/certs/mongodb-ca.crt:ro
      # SSL keys for MongoDB
      - ./containerization/files/devconfigs/mongodb.pem:/certs/mongodb.pem:ro

  api:
    volumes:
      - ./containerization/files/devconfigs/config.yaml:/etc/decapod/config.yaml:ro

  controller:
    volumes:
      - ./containerization/files/devconfigs/config.yaml:/etc/decapod/config.yaml:ro
      - /keys/ansible_ssh_keyfile.pem:/root/.ssh/id_rsa:ro

  cron:
    volumes:
      - ./containerization/files/devconfigs/config.yaml:/etc/decapod/config.yaml:ro
```

In this case, Docker Compose will use the following merged configuration:

```
networks: {}
services:
  api:
    image: docker-prod-virtual.docker.mirantis.net/mirantis/ceph/decapod/api:latest
    links:
    - database
    restart: on-failure:5
```

```
    volumes:
    - /vagrant/containerization/files/devconfigs/config.yaml:/etc/decapod/config.
↪yaml:ro
  controller:
    image: docker-prod-virtual.docker.mirantis.net/mirantis/ceph/decapod/
↪controller:latest
    links:
    - database
    restart: on-failure:5
    volumes:
    - /vagrant/containerization/files/devconfigs/config.yaml:/etc/decapod/config.
↪yaml:ro
    - /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
    volumes:
    - /vagrant/containerization/files/devconfigs/config.yaml:/etc/decapod/config.
↪yaml:ro
  database:
    image: docker-prod-virtual.docker.mirantis.net/mirantis/ceph/decapod/db:latest
    restart: always
    volumes:
    - /vagrant/containerization/files/devconfigs/mongodb-ca.crt:/certs/mongodb-ca.
↪crt:ro
    - /vagrant/containerization/files/devconfigs/mongodb.pem:/certs/mongodb.pem:ro
    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: {}
```

Note: If you have modified the configuration, provide it 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

config.yaml file

Decapod configuration is done with file in YAML format. This guide briefly goes through all configuration sections and describes each setting in details. Also, at the bottom of the page, you can find a list of specific usecases like integrations with different authentication backends.

Here is an example of default configuration 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"
```

Decapod tries to search configuration file in different places in following order:

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

If some configuration file was found and parsed before, other alternatives won't be used. In other words, if you have default configuration file in `/etc/decapod/config.yaml`, then placing configuration in `$XDG_CONFIG_HOME/decapod/config.yaml` will override it completely (check specs on XDG directories).

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

Configuration has several sections. Next section will cite mentioned configuration above and describe purpose and possible variations of these settings with some recommendations.

Settings

common

Common section defines some generic settings for Decapod which are not related to specifics, like API or controller settings.

```
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: ""
```

password This section describes settings for Decapod key derivation function. Decapod do not store user passwords in plaintext, instead it uses key derivation functions to calculate cryptographic secure hash from the password.

To do so, it uses Argon2 key derivation function which is somehow similar to scrypt but has a property to defense against concurrent attacks with GPUs.

Default settings are perfectly fine for most of deployments but is you want to tune them, please check recommendations on application of Argon2 to password hashing functionality.

password_reset_ttl_in_seconds When user resets her password, she gets a secret token. Consuming this token will do actual password reset. This setting sets TTL (Time-To-Live) of such token. Token lives only such amount of seconds and expires after.

email This configuration setting defines how to send emails from Decapod. *host*, *port*, *login* and *password* are self-descriptive settings, *from* means email to set in `From` field. *enabled* is a boolean setting which enables or disabled email sending. If it is disabled, all other fields in this section are ignored.

api

This settings group describes configuration specific to API service only.

```
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
```

Most of the settings propagates to Flask directly. To get descripton of Flask settings, please check official documentation. Please find the table with mapping below.

Decapod setting	Flask setting
debug	DEBUG
testing	TESTING
logger_name	LOGGER_NAME
logger_handler_policy	LOGGER_HANDLER_POLICY
json_sort_keys	JSON_SORT_KEYS
json_as_ascii	JSON_AS_ASCII
jsonify_prettyprint_regular	JSONIFY_PRETTYPRINT_REGULAR

If you not quite sure which setting to set, use default ones, they are reasonable for most of deployments.

The following settings are not Flask ones, but Decapod specific.

pagination_per_page This setting sets a default count of items per page in paginated listings. If amount of items is less than *pagination_per_page*, then less elements would be returned.

server_discovery_token Servers, found during process of server discovery, has to have some authentication token to use to access `POST /v1/server` API endpoint. This is a special token only for such purposes: it does not refer to any certain user and it is possible to access only mentioned API endpoint with it.

This is safe because even after accessing of this endpoint, Ansible has to access remote host to gather facts and verify access.

reset_password_url This is a template of URL which would be used for generating email on password reset. Email, which should be send to the user, will contain this URL. `{reset_token}` will be replaced by Decapod to correct password reset token.

token This setting set has a configuration for authentication tokens. At the time of writing, only one setting is present: *ttl_in_seconds* which defines token TTL in seconds. Please be noticed, that all tokens, which are already generated, won't respected updated setting, only new tokens will be generated.

This section makes sense only if native authentication backend is used. For example, Keystone integration won't respect this setting because Keystone manages its tokens.

In future releases this section can be moved to *auth*.

logging This section defines specific settings for logging in API. This applies settings from *logging* to API only.

auth This section configures authentication backend used by Decapod. Absent section implies native backend with default configuration. Configuration is set like this:

```
auth:
  type: sometype
  parameters:
    - setting1: value1
    - setting2: value2
```

type defines the type of backend to be used and *parameters* is an object to configure it. Please check *Authentication backends* for details on available backends.

controller

Controller defines specific settings for controller serivce. This service manages task queue and runs Ansible for tasks.

```
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"
```

daemon This section defines, shall we run controller as UNIX daemon. If you are using systemd or Docker containers, please set this to `false`.

pidfile If controller is run as daemon, this setting defines PIDFile for daemon to use.

ansible_config Path to default Ansible config to use. Usually, you do not want to change this setting.

worker_threads Controller uses worker pool to manage Ansible executions concurrently. You can set an amount of workers per controller in this setting. *0* has a special meaning: define this number automatically. By default it is `2 * cpu_count`.

graceful_stop Since controller executes a lot of processes, it cannot be stopped at the same moment: processes should be correctly finished. This settings defines the timeout of graceful stopping of those external processes. Initially, controller sends *SIGTERM* to them and if they won't stop after *graceful_stop* of seconds, it kills them with *SIGKILL*.

logging This section defines specific settings for logging in controller. This applies settings from *logging* to controller only.

Please be noticed that some scripts, unreleated to controller directly also uses these settings.

cron

This section defines several cron-like settings. They may or may not be used by cron, depepnding on current implementation.

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

clean_finished_tasks_after_seconds This setting defines TTL for finished tasks. They are going to be purged from database after this amount of seconds. This is related only to finished tasks, that were completed or failed. It does not related to not started ones.

db

These settings are related to MongoDB: how to connect to database and some specifics of db client configuration.

```
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
```

uri This setting describes URI to connect to MongoDB. Please check official docs on connection URIs.

connect This settings defines will Decapod connect to MongoDB immediately after initialization of a client or on the first request. It is suggested to keep this setting as `false`.

socket_timeout Controls how long (in milliseconds) the driver will wait during server monitoring when connecting a new socket to a server before concluding the server is unavailable.

socket_timeout Controls how long (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 The maximum allowable number of concurrent connections to each connected server. Requests to a server will block if there are *pool_size* outstanding connections to the requested server.

gridfs_chunk_size_in_bytes This setting defines a size of file chunk (a part of the file, stored in separate document) for GridFS. 255 kilobytes is a reasonable default.

plugins

This section describes what to do with plugins: disable, enable some etc.

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

As you can see, this section is a mapping of settings itself. All plugins are split in 2 categories: alerts and playbooks.

Alerts plugins are responsible for problem alerting (e.g 500 errors). This section has a list of enabled alerts plugins. Every key except of *enabled* is how to setup each alert plugin.

Playbooks section has only 1 setting: *disabled*. This is a list of plugins which are disabled even if they are installed.

logging

This section defines configuration of Decapod logging.

```
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"
```

The meaning of this section and options is described in official Python documentation on logging: https://docs.python.org/3.5/library/logging.config.html#configuration-dictionary-schema

Authentication backends

Decapod can use several authentication backends. This section enumerates supported variants.

Native authentication backend

Native configuration backend uses Decapod MongoDBs to store authentication tokens. Everytime when user logins into Decapod, it creates new authentication token and stores it in collection. Everytime user logouts, corresponding token is removed. Every token has TTL and wipes out when time is came (this is done by using MongoDB TTL indexes).

To setup native authentication backend, just use following snippet for your *config.yaml file*. Place this snippet in *api* section of the config.

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

This type of backend does not require any configuration. If you omit section *api* → *auth* completely, this will imply.

Keystone authentication backend

Keystone authentication backend uses Keystone for authentication. This is more complex setup than default *Native authentication backend* and involves several steps.

Keystone integration is one-way sync. Since Decapod uses its own role system, only user authentication is used. User delete in Decapod won't affect Keystone and Decapod won't write to Keystone. Keystone is just a read only source of authentication truth.

If Keystone integration is enabled, Decapod will sync user list by Cron every 10 minutes. If user is deleted or disabled in Keystone, it will be deleted in Decapod also. If user is created, it will be created in Decapod. If it is enabled again, user will be restored.

Setup `config.yaml`

To setup Keystone integration, please update your *config.yaml file* file, section *api*. Insert following snippet:

```
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}
```

Please check OpenStack official documentation on a meaning of these parameters. For whole list of options, please check documentation of v3.Password.

Important: Username and password should correspond to the user which can request token for other users and list them.

Initial keystone migration

After you enable Keystone, you immediately get a "chicken or the egg" problem: to access Decapod and set user permissions, you need a user with enough permissions, but this user can be absent from Decapod itself.

The solution is initial sync. After you've setup your `config.yaml`, you need to perform initial sync. This could be done with *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.
```

With this utility you need to set the role name (default is *wheel* which has a biggest number of permissions) and user logins which will have this role. After that sync, you can access Decapod and set roles for required users.

Note: Newely synchronized users from Keystone won't have any role.

Synchronization is done by cron in *admin* service but you can execute it manually after initial sync.

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

CHAPTER 3

Data models

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

The section contains the following topics:

User model

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

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

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

See also:

- *Role model*

Role model

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

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

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

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

Server model

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

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

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

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

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

See also:

- Ohai
- Facter
- The cloud-init documentation

Cluster model

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

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

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

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

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

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

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

See also:

- *Playbook configuration*
- *Playbook execution*

Decapod playbooks

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

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

Playbook configuration

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

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

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

Basically, configuring a playbook includes:

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

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

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

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

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

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

See also:

- *Playbook execution*

Playbook execution

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

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

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

See also:

- *Playbook configuration*

CHAPTER 4

Manage users and roles

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

Manage users

To add a new user:

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

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

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

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

Manage roles

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

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

See also:

- *Role model*

CHAPTER 5

Deploy a cluster

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

The section contains the following topics:

Create a cluster

To create a cluster:

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

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

View servers

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

To view the discovered servers:

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

Create a playbook configuration

To create a playbook configuration:

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

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

Execute a playbook configuration

To execute a playbook configuration:

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

CHAPTER 6

Ceph monitoring

Decapod may support integration with various monitoring systems. It ships with simplistic in-house monitoring tool ceph-monitoring. Also, it is possible to integrate with other tools like Prometheus via Telegraf. In future there will be more choices presented, please check the *list of supported plugins*.

`ceph-monitoring` is a tool which collects statistics on cluster state and monitors performance from time to time, you may consider it as executed by Cron.

admin service serves collected data. To access it, check `DECAPOD_MONITORING_PORT` environment variable (default is **10001**). So, if you access Decapod like **http://10.0.0.10:9999**, docs will be served on **http://10.0.0.10:10001**.

If you do not have information on cluster which was just deployed, please wait ~15 minutes and try again. If data is still not accessible, please check logs of **admin** service. You can get them with following command:

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

CHAPTER 7

Use the Decapod CLI

This section contains the following topics:

Install the Decapod CLI

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

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

To install the Decapod CLI:

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

```
$ make build_eggs
```

2. Install the packages:

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

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

See also:

- *Access the Decapod CLI*

Access the Decapod CLI

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

To access Decapod using CLI:

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

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

   Save this to a file and source when required.

2. Verify that it works:

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

   If you used environment variables, run:

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

Cluster deployment workflow

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

Create a cluster

To create a cluster:

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

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

   Example:

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

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

3. Proceed to *Discover a server*.

Discover a server

To discover a server:

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

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

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

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

See also:

- *Server model*

Create a playbook configuration

To create a playbook configuration:

1. List the existing playbooks:

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

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

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

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

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

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

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

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

     Where `machine_id` is the server ID.

   - Use the `jq` tool mentioned above:

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

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

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

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

Example:

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

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

```
                    "10.10.0.12",
                    "10.10.0.11",
                    "10.10.0.8"
                ],
                "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
```

CHAPTER 8

Backup and restore procedures

Decapod with **`decapod-admin`** tool (a part of *admin* service) allows to create backup. If you have dockerized setup (i.e running with **`docker-compose`**) you need to setup backup procedure manually.

Decapod stores its state in MongoDB and in 99% of cases restoring of DB backups allows to restore all decapod data. Another 1% is internal container state, like a data from ceph-monitoring (check *Ceph monitoring* chapter for details). This data is ok to be lost since Decapod refresh it every 10 minutes by default (for urgent cases, it is even possbible to collect explicitly with `docker-compose exec controller decapod-collect-data` command).

To perform backup, just execute following:

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

And to restore:

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

Note: At the time of writing, it was not possible to use **`docker-compose exec`** to perform restore due to the bug in docker-compose: https://github.com/docker/compose/issues/3352

There are 2 scripts in `./scripts` directory, `backup_db.py` and `restore_db.py` which does backup/restore for you.

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

You can add backup to cron like this:

```
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
```

CHAPTER 9

Deploy an operating system on a Ceph node

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

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

The section contains the following topics:

Generate user data for cloud-init

This section contains the following topics:

Prerequisites

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

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

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

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

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

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

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

Generate user data

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

To generate user data:

Run the following command:

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

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

Deploy OS using MAAS

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

The section contains the following topics:

Prerequisites

MAAS installation has the following requirements:

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

Install MAAS

To install MAAS:

To install MAAS, follow the steps described in:

1. Installing a single node MAAS.
2. Importing the boot images.
3. Logging in.

Deploy an OS using MAAS

To deploy an operating system using MAAS:

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

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

2. Deploy an OS using the required MAAS version.

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

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

```
$ maas mymaas nodes read
```

  (b) Deploy the OS:

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

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

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

```
$ maas prof nodes list
```

  (b) Deploy the OS:

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

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

Note: If you do not want (or cannot) use server discovery for some reason, please check Ansible playbooks which will prepare machine based on generated user-data.

https://github.com/Mirantis/ceph-lcm/tree/master/infrastructure_playbooks/server_discovery_playbook

---

CHAPTER 10

Supported Ceph packages

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

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

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

The following table lists packages provided for upgrades only:

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

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

Note: It is possible and recommended to create and use your own repository. To do so, please check corresponding playbook and follow the instructions.

https://github.com/Mirantis/ceph-lcm/tree/master/infrastructure_playbooks/apt_mirror_playbook

This playbook creates only server, please setup webserver like nginx or caddy to serve static by yourself.

CHAPTER 11

Playbook plugins

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

The section contains the following topics:

Deploy Ceph cluster

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

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

The section contains the following topics:

Overview

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

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

The following table lists the available hints for the plugin:

Hint	Title	Default value	Description
dm-crypt	Use dmcrypted OSDs	False	Defines the dmcrypt usage for OSD devices.
collo-cation	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,<1.0	v1.0.8
>=1.0,<1.1	v2.1.9

Parameters and roles

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

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

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

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

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

`cluster` Defines the cluster name.

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

`cluster_network` Defines the cluster network.

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

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

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

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

`raw_multi_journal` This option is the opposite to `journal_collocation`.

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

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

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

journal_size OSD journal size in megabytes.

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

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

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

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

public_network Defines the public network.

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

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

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

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

The following table lists the possible combinations:

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

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

Configuration example

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

```
{
  "global_vars": {
    "ceph_facts_template": "/usr/local/lib/python3.5/dist-packages/decapod_common/
↪facts/ceph_facts_module.py.j2",
    "ceph_stable": true,
    "ceph_stable_distro_source": "jewel-xenial",
    "ceph_stable_release": "jewel",
    "ceph_stable_repo": "http://eu.mirror.fuel-infra.org/shrimp/ceph/apt",
    "cluster": "ceph",
    "cluster_network": "10.10.0.0/24",
    "copy_admin_key": true,
```

```
    "dmcrypt_dedicated_journal": true,
    "dmcrypt_journal_collocation": false,
    "fsid": "e0b82a0d-b669-4787-8f4d-84f6733e45cd",
    "journal_collocation": false,
    "journal_size": 512,
    "max_open_files": 131072,
    "nfs_file_gw": false,
    "nfs_obj_gw": false,
    "os_tuning_params": [
      {
        "name": "kernel.pid_max",
        "value": 4194303
      },
      {
        "name": "fs.file-max",
        "value": 26234859
      }
    ],
    "public_network": "10.10.0.0/24",
    "raw_multi_journal": false
  },
  "inventory": {
    "_meta": {
      "hostvars": {
        "10.10.0.10": {
          "ansible_user": "ansible",
          "devices": [
            "/dev/vde",
            "/dev/vdb"
          ],
          "monitor_interface": "ens3",
          "raw_journal_devices": [
            "/dev/vdd",
            "/dev/vdc"
          ]
        },
        "10.10.0.11": {
          "ansible_user": "ansible",
          "devices": [
            "/dev/vde",
            "/dev/vdb"
          ],
          "monitor_interface": "ens3",
          "raw_journal_devices": [
            "/dev/vdd",
            "/dev/vdc"
          ]
        },
        "10.10.0.12": {
          "ansible_user": "ansible",
          "devices": [
            "/dev/vde",
            "/dev/vdb"
          ],
          "monitor_interface": "ens3",
          "raw_journal_devices": [
            "/dev/vdd",
            "/dev/vdc"
```

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


Add OSD host

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

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

of supported parameters, see official list.

The section contains the following topics:

Overview

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

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

The following table lists the available hints for the plugin:

Hint	Title	Default value	Description
dm-crypt	Use dmcrypted OSDs	False	Defines the dmcrypt usage for OSD devices.
collo-cation	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,<1.0	v1.0.8
>=1.0,<1.1	v2.1.9

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.

Also, there is a limitation on deployment: for consistency and problem avoidance, Decapod checks version it is going to install. If cluster has inconsistent versions, deployment is stopped: user has to fix versions within a cluster. If version user is going to deploy is newer than installed ones, process is also failing: user has to update cluster packages first.

There are 2 parameters responsible for that:

`ceph_version_verify` This is a boolean setting which checks that strict mode is enabled. Otherwise (it is set to `false`) no verification described above is done.

`ceph_version_verify_packagename` The name of the package to check. Usually, you do not need to touch this setting at all.

Configuration example

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

```
{
    "data": {
        "cluster_id": "1597a71f-6619-4db6-9cda-a153f4f19097",
        "configuration": {
            "global_vars": {
                "ceph_facts_template": "/usr/local/lib/python3.5/dist-packages/shrimp_
↪common/facts/ceph_facts_module.py.j2",
                "ceph_stable": true,
                "ceph_stable_distro_source": "jewel-xenial",
                "ceph_stable_release": "jewel",
                "ceph_stable_repo": "http://eu.mirror.fuel-infra.org/shrimp/ceph/apt",
                "cluster": "ceph",
                "cluster_network": "10.10.0.0/24",
                "copy_admin_key": true,
                "fsid": "1597a71f-6619-4db6-9cda-a153f4f19097",
                "journal_collocation": true,
                "journal_size": 100,
                "max_open_files": 131072,
                "nfs_file_gw": false,
                "nfs_obj_gw": false,
                "os_tuning_params": [
                    {
                        "name": "kernel.pid_max",
                        "value": 4194303
                    },
                    {
                        "name": "fs.file-max",
                        "value": 26234859
                    }
                ],
                "public_network": "10.10.0.0/24"
            },
            "inventory": {
                "_meta": {
                    "hostvars": {
                        "10.10.0.2": {
                            "ansible_user": "ansible",
                            "devices": [
                                "/dev/vdb"
                            ],
                            "monitor_interface": "ens3"
                        },
                        "10.10.0.3": {
                            "ansible_user": "ansible",
                            "devices": [
                                "/dev/vdb"
                            ],
                            "monitor_interface": "ens3"
                        }
                    }
                },
                "mons": [
                    "10.10.0.2"
                ],
                "osds": [
                    "10.10.0.3",
                ],
            }
```

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


Remove OSD host

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

The section contains the following topics:

Overview

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

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

Configuration example

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

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

```
  }
}
```

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

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.

The section contains the following topics:

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
>=1.0,<1.1	v2.1.9

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.

Also, there is a limitation on deployment: for consistency and problem avoidance, Decapod checks version it is going to install. If cluster has inconsistent versions, deployment is stopped: user has to fix versions within a cluster. If version user is going to deploy is newer than installed ones, process is also failing: user has to update cluster packages first.

There are 2 parameters responsible for that:

`ceph_version_verify` This is a boolean setting which checks that strict mode is enabled. Otherwise (it is set to `false`) no verification described above is done.

`ceph_version_verify_packagename` The name of the package to check. Usually, you do not need to touch this setting at all.

Configuration example

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


```
{
  "global_vars": {
    "ceph_facts_template": "/usr/local/lib/python3.5/dist-packages/decapod_common/
↪facts/ceph_facts_module.py.j2",
    "ceph_stable": true,
    "ceph_stable_distro_source": "jewel-xenial",
    "ceph_stable_release": "jewel",
    "ceph_stable_repo": "http://eu.mirror.fuel-infra.org/shrimp/ceph/apt",
    "cluster": "ceph",
    "cluster_network": "10.10.0.0/24",
    "copy_admin_key": true,
    "fsid": "d5069dc9-05d9-4ef2-bc21-04a938917260",
    "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",
          "monitor_interface": "ens3"
        },
        "10.10.0.12": {
          "ansible_user": "ansible",
          "monitor_interface": "ens3"
        },
        "10.10.0.8": {
          "ansible_user": "ansible",
          "monitor_interface": "ens3"
        },
        "10.10.0.9": {
          "ansible_user": "ansible",
          "monitor_interface": "ens3"
        }
      }
    },
    "mons": [
      "10.10.0.10",
      "10.10.0.12",
      "10.10.0.8",
      "10.10.0.9"
    ]
```

```
  }
}
```

Remove montor host

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

The section contains the following topics:

Overview

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

Property	Value
ID	remove_mon
Name	Remove monitor host
Required Server List	Yes

Please pay attention, that you have to 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.

Purge cluster

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

The section contains the following topics:

Overview

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

Property	Value
ID	purge_cluster
Name	Purge Cluster
Required Server List	No

Parameters and roles

The *Purge cluster* plugin has the following parameter:

`cluster` Defines the name of the cluster.

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

Configuration example

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

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

```
      "10.10.0.12",
      "10.10.0.11",
      "10.10.0.8"
    ],
    "restapis": [
      "10.10.0.9"
    ]
  }
}
```

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

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.

The section contains the following topics:

Overview

The following table shows the 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
>=1.0,<1.1	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:

- Telegraf Ceph Input Source.
- Installing and configuring Telegraf <official documentation.

Telegraf purging

While *Telegraf integration* plugin installs and configures Telegraf, this plugin will revert it back uninstalling Telegraf or its managed section from the config.

The section contains the following topics:

Overview

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

Property	Value
ID	purge_telegraf
Name	Telegraf removal
Required Server List	Yes

Hints

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

Configuration example

The following is an example of the *Telegraf purging* 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"
```

```
      ]
   }
}
```

CHAPTER 12

Upgrade Guide

This chapter describes in details how to upgrade Decapod from one version to another. Please backup before starting any upgrade procedure. Also, please pay attention that in most cases (if there are no explicit chapters or recommendations) upgrade is possible only from previous version to current one. If you are going to upgrade from version *X* to version *X+3*, you need to upgrade to *X+1* first, then to *X+2* and only after that to *X+3*.

Contents

Upgrade from 0.1.x to 1.0

This chapter describes upgrade procedure from versions **0.1.x** to **1.0.x**.

The major issues with upgrading from 0.1.x to 1.0 are:

1. During development of 1.0 Decapod changed its name from Shrimp. It means that naming of some defaults was changed from *shrimp* to *decapod*. For example, name of the database in default config was changed.
2. Configuration files are not stored in containers anymore, but injected using volumes. It means that Decapod images are same in testing and production.
3. Usage of shell scripts from `./scripts` directory is deprecated, because Decapod got admin service.
4. MongoDB was updated from *3.2.x* to *3.4.x*.

Changes in existing Ceph deployments are not required.

This guide tries to cover all these obstacles. It is split into 2 parts: action, required to perform on version 0.1 and actions for version 1.0.

Initial Preparations

To start, we need to have latest versions of 1.0 release series. To obtain them, please clone repositories on machine which is used to run Decapod.

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

After that, please create directory where config files and private keys for decapod should be stored. You may choose any directory you like.

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

The last step is to identify the name of the project. If you run Shrimp with explicit project name using docker-compose (e.g `docker-compose -p shrimp up`), then `shrimp` has to be your projectname. If you didn't set any, then you need to fetch it.

Execute next command from the directory where you run **`docker-compose`** to start Shrimp.

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

For simplicity, please assume that the name of the project is `PROJ`.

Note: If you do not want to pass `-p` all the time you use **`docker-compose`**, please use `COMPOSE_PROJECT_NAME` environment variable.

Now let's copy required files in your `~/decapod_runtime` directory.

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

And let's set the path to SSH private key in `.env` file.

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

If you are using the name other than `id_rsa` for private key, use it.

Backup Database

We will create 2 backups:

`pre_upgrade` This will have a backup of data before any other action.

`pre_upgrade_renamed` This will have a backup in case if you want to use default config from 1.0 and do not port existing one.

Create `pre_upgrade` Backup

From the directory where you run Shrimp, please execute following command. Please, pay attention to the fact that **PROJ** is listed as lowercase **proj** here (this is how **`docker-compose`** is converting project name to container name).

```
$ docker exec -i proj_database_1 mongodump --gzip --archive --ssl --
↪sslAllowInvalidCertificates > ~/pre_upgrade
2017-03-01T14:21:52.856+0000    writing shrimp.migration_script to archive on stdout
2017-03-01T14:21:52.857+0000    writing shrimp.role to archive on stdout
2017-03-01T14:21:52.857+0000    writing shrimp.lock to archive on stdout
2017-03-01T14:21:52.857+0000    writing shrimp.user to archive on stdout
2017-03-01T14:21:52.857+0000    done dumping shrimp.migration_script (3 documents)
```

```
2017-03-01T14:21:52.860+0000    writing shrimp.cluster to archive on stdout
2017-03-01T14:21:52.862+0000    done dumping shrimp.cluster (1 document)
2017-03-01T14:21:52.866+0000    writing shrimp.server to archive on stdout
2017-03-01T14:21:52.867+0000    done dumping shrimp.server (0 documents)
2017-03-01T14:21:52.869+0000    done dumping shrimp.user (1 document)
2017-03-01T14:21:52.875+0000    writing shrimp.kv to archive on stdout
2017-03-01T14:21:52.876+0000    writing shrimp.execution_step to archive on stdout
2017-03-01T14:21:52.876+0000    done dumping shrimp.execution_step (0 documents)
2017-03-01T14:21:52.881+0000    writing shrimp.task to archive on stdout
2017-03-01T14:21:52.882+0000    done dumping shrimp.lock (1 document)
2017-03-01T14:21:52.882+0000    done dumping shrimp.kv (0 documents)
2017-03-01T14:21:52.882+0000    done dumping shrimp.task (0 documents)
2017-03-01T14:21:52.887+0000    writing shrimp.execution to archive on stdout
2017-03-01T14:21:52.888+0000    done dumping shrimp.role (1 document)
2017-03-01T14:21:52.889+0000    done dumping shrimp.execution (0 documents)
2017-03-01T14:21:52.891+0000    writing shrimp.token to archive on stdout
2017-03-01T14:21:52.892+0000    writing shrimp.playbook_configuration to archive on␣
↪stdout
2017-03-01T14:21:52.894+0000    done dumping shrimp.token (0 documents)
2017-03-01T14:21:52.894+0000    done dumping shrimp.playbook_configuration (0␣
↪documents)
```

Important: If you want to restore database for any reason, please execute following:

```
$ docker exec -i proj_database_1 mongorestore --drop --gzip --archive --ssl --
↪sslAllowInvalidCertificates < ~/pre_upgrade
2017-03-01T14:26:19.268+0000    creating intents for archive
2017-03-01T14:26:19.309+0000    reading metadata for shrimp.migration_script from␣
↪archive on stdin
2017-03-01T14:26:19.465+0000    restoring shrimp.migration_script from archive on␣
↪stdin
2017-03-01T14:26:19.469+0000    restoring indexes for collection shrimp.migration_
↪script from metadata
2017-03-01T14:26:19.469+0000    finished restoring shrimp.migration_script (3␣
↪documents)
2017-03-01T14:26:19.539+0000    reading metadata for shrimp.cluster from archive on␣
↪stdin
2017-03-01T14:26:19.728+0000    restoring shrimp.cluster from archive on stdin
2017-03-01T14:26:19.735+0000    restoring indexes for collection shrimp.cluster from␣
↪metadata
2017-03-01T14:26:20.010+0000    finished restoring shrimp.cluster (1 document)
2017-03-01T14:26:20.206+0000    reading metadata for shrimp.server from archive on␣
↪stdin
2017-03-01T14:26:20.306+0000    reading metadata for shrimp.user from archive on stdin
2017-03-01T14:26:20.507+0000    restoring shrimp.server from archive on stdin
2017-03-01T14:26:20.509+0000    restoring indexes for collection shrimp.server from␣
↪metadata
2017-03-01T14:26:20.731+0000    restoring shrimp.user from archive on stdin
2017-03-01T14:26:21.580+0000    restoring indexes for collection shrimp.user from␣
↪metadata
2017-03-01T14:26:21.580+0000    finished restoring shrimp.server (0 documents)
2017-03-01T14:26:21.707+0000    reading metadata for shrimp.execution_step from␣
↪archive on stdin
2017-03-01T14:26:21.732+0000    reading metadata for shrimp.lock from archive on stdin
2017-03-01T14:26:22.119+0000    finished restoring shrimp.user (1 document)
2017-03-01T14:26:22.374+0000    restoring shrimp.execution_step from archive on stdin
2017-03-01T14:26:22.376+0000    restoring indexes for collection shrimp.execution_
↪step from metadata
```

```
2017-03-01T14:26:22.579+0000    restoring shrimp.lock from archive on stdin
2017-03-01T14:26:22.666+0000    finished restoring shrimp.execution_step (0 documents)
2017-03-01T14:26:22.724+0000    reading metadata for shrimp.kv from archive on stdin
2017-03-01T14:26:22.724+0000    restoring indexes for collection shrimp.lock from␣
↪metadata
2017-03-01T14:26:22.790+0000    reading metadata for shrimp.task from archive on stdin
2017-03-01T14:26:22.824+0000    reading metadata for shrimp.role from archive on stdin
2017-03-01T14:26:23.016+0000    restoring shrimp.kv from archive on stdin
2017-03-01T14:26:23.018+0000    restoring indexes for collection shrimp.kv from␣
↪metadata
2017-03-01T14:26:23.208+0000    finished restoring shrimp.lock (1 document)
2017-03-01T14:26:23.440+0000    restoring shrimp.task from archive on stdin
2017-03-01T14:26:23.443+0000    restoring indexes for collection shrimp.task from␣
↪metadata
2017-03-01T14:26:23.616+0000    restoring shrimp.role from archive on stdin
2017-03-01T14:26:23.745+0000    finished restoring shrimp.kv (0 documents)
2017-03-01T14:26:23.938+0000    finished restoring shrimp.task (0 documents)
2017-03-01T14:26:24.024+0000    reading metadata for shrimp.execution from archive on␣
↪stdin
2017-03-01T14:26:24.024+0000    restoring indexes for collection shrimp.role from␣
↪metadata
2017-03-01T14:26:24.090+0000    reading metadata for shrimp.token from archive on␣
↪stdin
2017-03-01T14:26:24.146+0000    reading metadata for shrimp.playbook_configuration␣
↪from archive on stdin
2017-03-01T14:26:24.407+0000    restoring shrimp.execution from archive on stdin
2017-03-01T14:26:24.410+0000    restoring indexes for collection shrimp.execution␣
↪from metadata
2017-03-01T14:26:24.782+0000    finished restoring shrimp.role (1 document)
2017-03-01T14:26:24.991+0000    restoring shrimp.token from archive on stdin
2017-03-01T14:26:24.993+0000    restoring indexes for collection shrimp.token from␣
↪metadata
2017-03-01T14:26:25.275+0000    restoring shrimp.playbook_configuration from archive␣
↪on stdin
2017-03-01T14:26:25.277+0000    restoring indexes for collection shrimp.playbook_
↪configuration from metadata
2017-03-01T14:26:25.473+0000    finished restoring shrimp.execution (0 documents)
2017-03-01T14:26:25.584+0000    finished restoring shrimp.token (0 documents)
2017-03-01T14:26:25.852+0000    finished restoring shrimp.playbook_configuration (0␣
↪documents)
2017-03-01T14:26:25.852+0000    done
```

Create `pre_upgrade_renamed` Backup

Since project was renamed from Shrimp to Decapod during development of release 1.0, default database name was also changed from *shrimp* to *decapod*. If you want to use new name and keep running with default config, then please rename it in Mongo DB doing following:

```
$ 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 }
```

The command above will rename database *shrimp* to *decapod* keeping all the data. After that, do new backup as described above:

```
$ docker exec -i proj_database_1 mongodump --gzip --archive --ssl --
↪sslAllowInvalidCertificates > ~/pre_upgrade_renamed
2017-03-01T14:28:36.830+0000    writing decapod.user to archive on stdout
2017-03-01T14:28:36.831+0000    writing decapod.lock to archive on stdout
2017-03-01T14:28:36.831+0000    writing decapod.role to archive on stdout
2017-03-01T14:28:36.832+0000    writing decapod.migration_script to archive on stdout
2017-03-01T14:28:36.833+0000    done dumping decapod.user (1 document)
2017-03-01T14:28:36.845+0000    writing decapod.cluster to archive on stdout
2017-03-01T14:28:36.845+0000    done dumping decapod.cluster (1 document)
2017-03-01T14:28:36.846+0000    done dumping decapod.lock (1 document)
2017-03-01T14:28:36.852+0000    done dumping decapod.role (1 document)
2017-03-01T14:28:36.852+0000    writing decapod.kv to archive on stdout
2017-03-01T14:28:36.853+0000    done dumping decapod.migration_script (3 documents)
2017-03-01T14:28:36.854+0000    writing decapod.execution_step to archive on stdout
2017-03-01T14:28:36.855+0000    done dumping decapod.kv (0 documents)
2017-03-01T14:28:36.859+0000    writing decapod.server to archive on stdout
2017-03-01T14:28:36.862+0000    writing decapod.task to archive on stdout
2017-03-01T14:28:36.862+0000    writing decapod.playbook_configuration to archive on␣
↪stdout
2017-03-01T14:28:36.862+0000    done dumping decapod.execution_step (0 documents)
2017-03-01T14:28:36.862+0000    done dumping decapod.playbook_configuration (0␣
↪documents)
2017-03-01T14:28:36.862+0000    done dumping decapod.server (0 documents)
2017-03-01T14:28:36.869+0000    writing decapod.token to archive on stdout
2017-03-01T14:28:36.869+0000    done dumping decapod.task (0 documents)
2017-03-01T14:28:36.870+0000    done dumping decapod.token (0 documents)
2017-03-01T14:28:36.872+0000    writing decapod.execution to archive on stdout
2017-03-01T14:28:36.873+0000    done dumping decapod.execution (0 documents)
```

Extract Config Files

If you already have a configuration files from old version, please collect them in some directory (e.g `~/decapod_runtime`). Decapod version 1.0.x and newer will have default files stored in containers but you need to mount your own if you've changed some defaults.

If you already have all files, mentioned in documentation on version 0.1 in `~/decapod_runtime`, you can skip this section and proceed to *Stop and Remove Containers for Version 0.1.x*.

Otherwise, execute commands mentioned below to collect required files. These commands should be executed from the same directory which you are using to run Shrimp 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
```

If you do not have generated any files mentioned above by yourself and used defaults, there is not need to copy them: all of them will be stored in correct places in 1.0.x images. If you've modified any of `config.yaml` or `id_rsa` (SSH private key for Ansible), please copy them.

Get Images for Version 1.0

Please follow *Install and configure Decapod* chapter to get new images. And remember that we have required files in `~/decapod_runtime`. Repository for version 1.0 is cloned in `~/decapod` as mentioned in *Initial Preparations*.

Stop and Remove Containers for Version 0.1.x

Since Docker containers are stateless and we have a backup of the state (DB backup), the most easiest and safe method of updating is to drop existing containers and start new ones.

From the directory where you run Shrimp do the following:

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

Run 1.0.x Version

Note: The rest of operations would be performed from `~/decapod_runtime` directory so please **cd** into.

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

The next step is to restore DB:

```
$ docker exec -i $(docker-compose -p PROJ ps -q admin) decapod-admin db restore < ~/
↪pre_upgrade_renamed
2017-03-01T14:32:16.139+0000    preparing collections to restore from
2017-03-01T14:32:16.179+0000    reading metadata for decapod.user from archive on␣
↪stdin
2017-03-01T14:32:16.322+0000    restoring decapod.user from archive on stdin
2017-03-01T14:32:16.325+0000    restoring indexes for collection decapod.user from␣
↪metadata
2017-03-01T14:32:16.781+0000    finished restoring decapod.user (1 document)
2017-03-01T14:32:16.781+0000    reading metadata for decapod.cluster from archive on␣
↪stdin
2017-03-01T14:32:16.931+0000    restoring decapod.cluster from archive on stdin
2017-03-01T14:32:16.934+0000    restoring indexes for collection decapod.cluster from␣
↪metadata
2017-03-01T14:32:16.936+0000    reading metadata for decapod.lock from archive on␣
↪stdin
2017-03-01T14:32:17.217+0000    finished restoring decapod.cluster (1 document)
2017-03-01T14:32:17.406+0000    restoring decapod.lock from archive on stdin
2017-03-01T14:32:17.415+0000    restoring indexes for collection decapod.lock from␣
↪metadata
2017-03-01T14:32:17.417+0000    reading metadata for decapod.role from archive on␣
↪stdin
2017-03-01T14:32:17.629+0000    finished restoring decapod.lock (1 document)
2017-03-01T14:32:17.788+0000    restoring decapod.role from archive on stdin
2017-03-01T14:32:17.790+0000    reading metadata for decapod.migration_script from␣
↪archive on stdin
2017-03-01T14:32:17.922+0000    restoring decapod.migration_script from archive on␣
↪stdin
2017-03-01T14:32:17.923+0000    restoring indexes for collection decapod.role from␣
↪metadata
2017-03-01T14:32:17.925+0000    reading metadata for decapod.kv from archive on stdin
2017-03-01T14:32:18.133+0000    no indexes to restore
2017-03-01T14:32:18.133+0000    finished restoring decapod.migration_script (3␣
↪documents)
2017-03-01T14:32:18.133+0000    finished restoring decapod.role (1 document)
2017-03-01T14:32:18.265+0000    restoring decapod.kv from archive on stdin
2017-03-01T14:32:18.267+0000    restoring indexes for collection decapod.kv from␣
↪metadata
2017-03-01T14:32:18.267+0000    reading metadata for decapod.execution_step from␣
↪archive on stdin
2017-03-01T14:32:18.473+0000    restoring decapod.execution_step from archive on stdin
2017-03-01T14:32:18.476+0000    restoring indexes for collection decapod.execution_
↪step from metadata
2017-03-01T14:32:18.476+0000    reading metadata for decapod.playbook_configuration␣
↪from archive on stdin
2017-03-01T14:32:18.599+0000    finished restoring decapod.kv (0 documents)
2017-03-01T14:32:18.908+0000    restoring decapod.playbook_configuration from archive␣
↪on stdin
2017-03-01T14:32:18.910+0000    restoring indexes for collection decapod.playbook_
↪configuration from metadata
2017-03-01T14:32:18.910+0000    reading metadata for decapod.server from archive on␣
↪stdin
2017-03-01T14:32:18.981+0000    finished restoring decapod.execution_step (0␣
↪documents)
2017-03-01T14:32:19.135+0000    finished restoring decapod.playbook_configuration (0␣
↪documents)
2017-03-01T14:32:19.342+0000    restoring decapod.server from archive on stdin
2017-03-01T14:32:19.344+0000    restoring indexes for collection decapod.server from␣
↪metadata
```

```
2017-03-01T14:32:19.344+0000    reading metadata for decapod.task from archive on␣
↪stdin
2017-03-01T14:32:19.511+0000    restoring decapod.task from archive on stdin
2017-03-01T14:32:19.513+0000    restoring indexes for collection decapod.task from␣
↪metadata
2017-03-01T14:32:19.513+0000    reading metadata for decapod.token from archive on␣
↪stdin
2017-03-01T14:32:20.123+0000    finished restoring decapod.server (0 documents)
2017-03-01T14:32:20.327+0000    finished restoring decapod.task (0 documents)
2017-03-01T14:32:20.494+0000    restoring decapod.token from archive on stdin
2017-03-01T14:32:20.497+0000    restoring indexes for collection decapod.token from␣
↪metadata
2017-03-01T14:32:20.497+0000    reading metadata for decapod.execution from archive␣
↪on stdin
2017-03-01T14:32:20.585+0000    finished restoring decapod.token (0 documents)
2017-03-01T14:32:20.820+0000    restoring decapod.execution from archive on stdin
2017-03-01T14:32:20.823+0000    restoring indexes for collection decapod.execution␣
↪from metadata
2017-03-01T14:32:21.008+0000    finished restoring decapod.execution (0 documents)
2017-03-01T14:32:21.008+0000    done
```

or, if you skip renaming of database:

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

Now we need to apply migrations:

```
$ docker-compose -p PROJ exec admin decapod-admin migration apply
2017-02-14 07:04:12 [DEBUG   ] (        lock.py:118  ): Lock applying_migrations was␣
↪acquire by locker 5ebb8d44-2919-4913-85f8-47e160d02207
2017-02-14 07:04:12 [DEBUG   ] (        lock.py:183  ): Prolong thread for locker␣
↪applying_migrations of lock 5ebb8d44-2919-4913-85f8-47e160d02207 has been started.␣
↪Thread MongoLock prolonger 5ebb8d44-2919-4913-85f8-47e160d02207 for applying_
↪migrations, ident 140234729555712
2017-02-14 07:04:12 [INFO    ] (   migration.py:123  ): Run migration 0003_native_ttl_
↪index.py
2017-02-14 07:04:12 [INFO    ] (   migration.py:198  ): Run /usr/local/lib/python3.5/
↪dist-packages/decapod_admin/migration_scripts/0003_native_ttl_index.py. Pid 40
2017-02-14 07:04:13 [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
2017-02-14 07:04:13 [INFO    ] (   migration.py:277  ): Save result of 0003_native_
↪ttl_index.py migration (result MigrationState.ok)
2017-02-14 07:04:13 [INFO    ] (   migration.py:123  ): Run migration 0004_migrate_to_
↪native_ttls.py
2017-02-14 07:04:13 [INFO    ] (   migration.py:198  ): Run /usr/local/lib/python3.5/
↪dist-packages/decapod_admin/migration_scripts/0004_migrate_to_native_ttls.py. Pid 48
2017-02-14 07:04:14 [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
2017-02-14 07:04:14 [INFO    ] (   migration.py:277  ): Save result of 0004_migrate_
↪to_native_ttls.py migration (result MigrationState.ok)
2017-02-14 07:04:14 [INFO    ] (   migration.py:123  ): Run migration 0005_index_
↪cluster_data.py
2017-02-14 07:04:14 [INFO    ] (   migration.py:198  ): Run /usr/local/lib/python3.5/
↪dist-packages/decapod_admin/migration_scripts/0005_index_cluster_data.py. Pid 56
2017-02-14 07:04:16 [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
```

```
2017-02-14 07:04:16 [INFO    ] (   migration.py:277  ): Save result of 0005_index_
↪cluster_data.py migration (result MigrationState.ok)
2017-02-14 07:04:16 [INFO    ] (   migration.py:123  ): Run migration 0006_create_
↪cluster_data.py
2017-02-14 07:04:16 [INFO    ] (   migration.py:198  ): Run /usr/local/lib/python3.5/
↪dist-packages/decapod_admin/migration_scripts/0006_create_cluster_data.py. Pid 64
2017-02-14 07:04:17 [DEBUG   ] (        lock.py:164  ): Lock applying_migrations was␣
↪proloned by locker 5ebb8d44-2919-4913-85f8-47e160d02207.
2017-02-14 07:04:17 [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
2017-02-14 07:04:17 [INFO    ] (   migration.py:277  ): Save result of 0006_create_
↪cluster_data.py migration (result MigrationState.ok)
2017-02-14 07:04:17 [INFO    ] (   migration.py:123  ): Run migration 0007_add_
↪external_id_to_user.py
2017-02-14 07:04:17 [INFO    ] (   migration.py:198  ): Run /usr/local/lib/python3.5/
↪dist-packages/decapod_admin/migration_scripts/0007_add_external_id_to_user.py. Pid␣
↪72
2017-02-14 07:04:18 [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
2017-02-14 07:04:18 [INFO    ] (   migration.py:277  ): Save result of 0007_add_
↪external_id_to_user.py migration (result MigrationState.ok)
2017-02-14 07:04:18 [DEBUG   ] (        lock.py:202  ): Prolong thread for locker␣
↪applying_migrations of lock 5ebb8d44-2919-4913-85f8-47e160d02207 has been stopped.␣
↪Thread MongoLock prolonger 5ebb8d44-2919-4913-85f8-47e160d02207 for applying_
↪migrations, ident 140234729555712
2017-02-14 07:04:18 [DEBUG   ] (        lock.py:124  ): Try to release lock applying_
↪migrations by locker 5ebb8d44-2919-4913-85f8-47e160d02207.
2017-02-14 07:04:18 [DEBUG   ] (        lock.py:140  ): Lock applying_migrations was␣
↪released by locker 5ebb8d44-2919-4913-85f8-47e160d02207.
```

Set MongoDB Backward Incompatibility (optional)

This is optional part but if you want, you can set MongoDB to be non-backward compatible to previous release. To do that, please execute following:

```
$ 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  }
```

Change `root` Password (optional)

Regular reminder: if you haven't changed password for `root` user, you have to do it. Starting from version 1.0 you can do it using admin service. Please check *Password Reset*.

CHAPTER 13

Debug snapshot

To simplify communication interface between production and development, Decapod has a concept of debug snapshot, similar to Fuel snapshots: snapshot is an archive which contains all information, required to debug and troubleshoot problems.

To generate snapshot, just execute following:

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

or, if you have containers only:

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

If you use last way, please check docs and set correct settings if required:

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

Create 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
```

After execution, you will get snapshot as `snapshot_path.*` (snapshot tool will calculate best compression algorithm available on your platform and use its extension. So you may get `snapshot_path.tar.bz2` or `snapshot_path.tar.xz` depepnding on how your Python was built).

Information, stored in the snapshot:

- Backup of the database
- Logs from services
- `docker-compose.yml` file
- 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

No ansible private keys or user passwords (they are hashed by Argon2) are stored in debug snapshot.

CHAPTER 14

Admin service

Along with ordinary Decapod docker containers, docker-compose runs optional but strongly recommended service, called *admin* service. The main intention of this service is to simplify life of Decapod administrator providing special containers which acts like lightweight VM with configured *CLI interface* and special tool, **`decapod-admin`** for performing maintenance or low level operations on Decapod or cluster.

To access this service, use following command:

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

Note: As usual, `-p` means the name of the project. If you haven't specified it on running docker-compose, do not specify it here.

You will enter container. Default environment allows to run **`decapod`** utility with configured URL and login/password pair `root`/`root`.

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

Also, this server has a bunch of additional utilities to simplify administrator life: `vim`, `nano`, `less`, `jq`, `yaql`, `jmespath-terminal` and `jp`. Vim is configured as default editor.

Basically, it means that you can execute **`decapod`** from such 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
```

Also, admin service runs cron jobs and it means, that keystone synchronization, *monitoring* data collection is performed there.

```
root@7252bfd5947d:/# crontab -l
PATH=/bin:/usr/bin:/usr/local/bin
LC_ALL=C.UTF-8
LANG=C.UTF-8

*/10 * * * * flock -xn /usr/local/bin/decapod-collect-data timeout --foreground -k 3m␣
↪2m /usr/local/bin/decapod-collect-data > /var/log/cron.log 2>&1
*/10 * * * * flock -xn /usr/local/bin/decapod-admin /usr/local/bin/decapod-admin␣
↪keystone sync > /var/log/cron.log 2>&1
```

The most interesting part is **`decapod-admin`** utility which allows to perform a various maintenence and admin routines.

Migrations

Migration concept in Decapod is quite similar to migrations in databases but it does not affect only schema but data also. The main idea of such migration is to adapt existing data to newer version of Decapod.

Overview

```
root@7252bfd5947d:/# decapod-admin migration --help
Usage: decapod-admin migration [OPTIONS] COMMAND [ARGS]...

  Migrations for database.

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

Commands:
  apply  Apply migration script.
  list   List migrations.
  show   Show details on applied migration.

root@7252bfd5947d:/# decapod-admin migration apply --help
Usage: decapod-admin migration apply [OPTIONS] [MIGRATION_NAME]...

  Apply migration script.

  If no parameters are given, then run all not applied migration scripts if
```

```
  correct order.

Options:
  -r, --reapply  Reapply migrations even if them were applied.
  -f, --fake     Do not actual run migration, just mark it as applied.
  -h, --help     Show this message and exit.

root@7252bfd5947d:/# decapod-admin migration list --help
Usage: decapod-admin migration list [OPTIONS] [QUERY]

  List migrations.

  Available query filters are:

      - all (default) - list all migrations;
      - applied       - list only applied migrations;
      - not-applied   - list only not applied migrations.

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

root@7252bfd5947d:/# decapod-admin migration show --help
Usage: decapod-admin migration show [OPTIONS] MIGRATION_NAME

  Show details on applied migration.

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

To get a list of migrations, do following:

```
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
```

To apply migrations:

```
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.
```

Migrations can be applied at any moment: Decapod tracks migrations which were already applied.

To show details on migration:

```
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 config

You can generate user-data config for cloud-init with **decapod** as described in *Generate user data* chapter, but since **decapod-admin** knows about Decapod installation much more than **decapod** is expected to know, you can use it to generate correct config.

Overview

```
root@7252bfd5947d:/# decapod-admin cloud-config --help
Usage: decapod-admin cloud-config [OPTIONS] PUBLIC_URL

  Generate cloud-init user-data config for current installation.

Options:
  -u, --username TEXT    Username which should be used by Ansible  [default:
                         ansible]
  -n, --no-discovery     Generate config with user and packages but no
                         discovery files.
  -t, --timeout INTEGER  Timeout for remote requests.  [default: 20]
  -h, --help             Show this message and exit.
```

So, you need to set only URL accessible by Ceph nodes.

```
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␣
↪AAAAB3NzaC1yc2EAAAADAQABAAACAQC7K9bHPrSu5VHnUOis2Uwc822fMyPTtwjfOkzNi/
↪oVOxmd1QE3DilrO5fJ33pRwEj7r1DfTlJmZWs8XwWaaUXkQ+iyfRPtgt/Ox+X5A/XaLdi/
↪yz7UjnHc8ERDUT/
↪z73RzDwf21KNQOopGRuyhe+gvGZ5mhYDz3bnnYY9IRBNYaGw4bjS0q1AbkPa1PvCo7P5b5UuRjhi4H74zCFkQD4evQsrQOcgev5
↪eEJwnnd1TcYG7dS6FqMWpFX1gqcKjFIuqNTZLYzJu9U8mxxKmGOQSI6KWfP0etBw1YRHRIfdZmdaqSKHh0ZhUUHjbf8Hb5Vqv1F
↪KOm/eXCm4yQg0ImXHUmsDoW+6W6akI/
↪fSCAn8r9GK2QBBJPeTPA95WlOSXtICnrsqgb74yKPEsslzfrTUIiyoXBuuR9o5OoPXghKrazqcTeK/
↪Vdl7w4nZ00O4jllHMTrS1xyubN0QeBd+3D8Hy2bN5h7WjiJsZ2XhlKR0Z1i5AbgCR9hfQl84aFIXRARz+6uuDDHe2ONXujcS9jK
  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'
```

Database maintenence

decapod-admin performs backup and restore of MongoDB, main storage system used by Decapod. Archive format, created by this tool is native MongoDB archive, compressed by default.

Overview

```
root@7252bfd5947d:/# decapod-admin -h
Usage: decapod-admin [OPTIONS] COMMAND [ARGS]...

  Decapod Admin commandline tool.

  With this CLI admin/operator can perform low-level maintenence of Decapod.
  This tool is not intended to be used by anyone but administrators. End-
  users should not use it at all.

Options:
  -d, --debug  Run in debug mode.
  --version    Show the version and exit.
  -h, --help   Show this message and exit.

Commands:
  ceph-version    Commands related to fetching of Ceph version.
```

```
  cloud-config    Generate cloud-init user-data config for...
  db              Database commands.
  keystone        Keystone related commands.
  locked-servers  Commands to manage locked servers.
  migration       Migrations for database.
  pdsh            PDSH for decapod-admin.
  restore         Restores entity.
  ssh             Connect to remote machine by SSH.

root@7252bfd5947d:/# decapod-admin db --help
Usage: decapod-admin db [OPTIONS] COMMAND [ARGS]...

  Database commands.

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

Commands:
  backup   Backup database.
  restore  Restores database.

root@7252bfd5947d:/# decapod-admin db backup --help
Usage: decapod-admin db backup [OPTIONS]

  Backup database.

  This backup will use native MongoDB stream archive format already gzipped
  so please redirect to required file.

Options:
  -r, --no-compress  Do not gzip archive format.
  -h, --help         Show this message and exit.

root@7252bfd5947d:/# decapod-admin db restore --help
Usage: decapod-admin db restore [OPTIONS]

  Restores database.

  Backup is native MongoDB stream archive format, created by mongodump
  --archive or 'backup' subcommand

Options:
  -r, --no-compress  Do not gzip archive format.
  -h, --help         Show this message and exit.
```

Result of execution `decapod-admin db backup` is identical to output of `mongodump --archive --gzip`. Result of execution of `decapod-admin db restore` is identical to `mongorestore --archive --gzip`. `decapod-admin` uses `/etc/decapod/config.yaml` for reading Decapod's MongoDB settings and correctly constructs commandline respecting SSL settings.

To perform backup, do following

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

And to restore:

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

If you do not want to compress, use `-r` flag. It literally means, that **`mongodump`** and **`mongorestore`** won't use `--gzip` flag.

See also:

- Archiving and Compression in MongoDB Tools

SSH to Ceph hosts

It is possible to SSH on remote host with the same user as used by Ansible using **`decapod-admin`** only.

Overview

```
root@7252bfd5947d:/# decapod-admin ssh --help
Usage: decapod-admin ssh [OPTIONS] COMMAND [ARGS]...

  Connect to remote machine by SSH.

Options:
  -o, --ssh-args STRING         SSH arguments to pass to OpenSSH client (in a
                                form of '-o Compression=yes -o
                                CompressionLevel=9', single option)
  -i, --identity-file FILENAME  Path to the private key file.  [default:
                                /root/.ssh/id_rsa]
  -h, --help                    Show this message and exit.

Commands:
  server-id  Connect to remote machine by IP address.
  server-ip  Connect to remote machine by IP address.

root@7252bfd5947d:/# decapod-admin ssh server-id --help
Usage: decapod-admin ssh server-id [OPTIONS] SERVER_ID

  Connect to remote machine by IP address.

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

root@7252bfd5947d:/# decapod-admin ssh server-ip --help
Usage: decapod-admin ssh server-ip [OPTIONS] IP_ADDRESS

  Connect to remote machine by IP address.

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

So if you know server-id or IP, you can execute interactive SSH session with it. For example, if I want to connect to server `8cf8af12-89a0-477d-85e7-ce6cbe5f8a07`:

```
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
```

Parallel SSH executions

Sometimes it is required to execute some commands on remote hosts in parallel. As a rule, pdsh is used for that purposes but **decapod-admin** provides its own implementation, integrated with Decapod.

Using this implementation, you can execute command on multiple hosts in parallel, upload files and download them from remote hosts. Please check help messages from the tool to get details.

Overview

```
root@7252bfd5947d:/# decapod-admin pdsh --help
Usage: decapod-admin pdsh [OPTIONS] COMMAND [ARGS]...

  PDSH for decapod-admin.

  pdsh allows user to execute commands on host batches in parallel using SSH
  connection.

  Please be noticed that -w flag is priority one, all other filters just
  won't work at all.

  If filter is not set, then it means, that all items in the scope will be
  processed (if no role is set, then all roles will be processed etc.)

Options:
  -b, --batch-size INTEGER      By default, command won't connect to all
                                servers simultaneously, it is trying to
                                process servers in batches. Negative number or
                                0 means connect to all hosts  [default: 20]
  -i, --identity-file FILENAME  Path to the private key file  [default:
                                /root/.ssh/id_rsa]
  -w, --server-id TEXT          Servers IDs to connect to. You can set this
                                option multiple times.
  -r, --role-name TEXT          Role name in cluster. You can set this option
                                multiple times. This option works only if you
                                set cluster-id.
  -c, --cluster-id TEXT         Cluster ID to process. You can set this option
                                multiple times.
  -h, --help                    Show this message and exit.

Commands:
  download  Download files from remote host.
  exec      Execute command on remote machines.
  upload    Upload files to remote host.

root@7252bfd5947d:/# decapod-admin pdsh download --help
Usage: decapod-admin pdsh download [OPTIONS] REMOTE_PATH... LOCAL_PATH
```

```
  Download files from remote host.

  When downloading a single file or directory, the local path can be either
  the full path to download data into or the path to an existing directory
  where the data should be placed. In the latter case, the base file name
  from the remote path will be used as the local name.

  Local path must refer to an existing directory.

  If --flat is not set, then directories with server ID and server IP will
  be created (server ID directory will be symlink to server IP).

Options:
  --no-follow-symlinks  Do not process symbolic links
  --no-recursive        The remote path points at a directory, the entire
                        subtree under that directory is not processed
  --no-preserve         The access and modification times and permissions of
                        the original file are not set on the processed file.
  --flat                Do not create directory with server ID and IP on
                        download
  --glob-pattern        Consider remote paths as globs.
  -h, --help            Show this message and exit.

root@7252bfd5947d:/# decapod-admin pdsh exec --help
Usage: decapod-admin pdsh exec [OPTIONS] COMMAND...

  Execute command on remote machines.

Options:
  -s, --sudo  Run command as sudo user.
  -h, --help  Show this message and exit.

root@7252bfd5947d:/# decapod-admin pdsh upload --help
Usage: decapod-admin pdsh upload [OPTIONS] LOCAL_PATH... REMOTE_PATH

  Upload files to remote host.

  When uploading a single file or directory, the remote path can be either
  the full path to upload data into or the path to an existing directory
  where the data should be placed. In the latter case, the base file name
  from the local path will be used as the remote name.

  When uploading multiple files, the remote path must refer to an existing
  directory.

  Local path could be glob.

Options:
  --no-follow-symlinks  Do not process symbolic links
  --no-recursive        The remote path points at a directory, the entire
                        subtree under that directory is not processed
  --no-preserve         The access and modification times and permissions of
                        the original file are not set on the processed file.
  -y, --yes             Do not ask about confirmation.
  -h, --help            Show this message and exit.
```

Example

```
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 deleted entities

Sometimes admin requires to restore some items which were deleted explicitly or accidentaly. To do so, you can use **`decapod-admin`**.

Overview

```
root@7252bfd5947d:/# decapod-admin restore -h
Usage: decapod-admin restore [OPTIONS] ITEM_TYPE ITEM_ID

  Restores entity.

  User selects type of entity (e.g cluster or server) and its ID, this
  command 'undeletes' it in database.

  Valid item types are:

      - cluster
      - execution
      - playbook-configuration
      - role
      - user
      - server

Options:
  -y, --yes   Do not ask about confirmation.
  -h, --help  Show this message and exit.
```

For example, you want to restore user with ID `6805075b-e40d-4800-8520-8569dd7327bd`.

```
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
}
```

Unlock servers

All playbook executions lock servers they use. This is done to eliminate situation when concurrent execution will cause unexpected problems. But sometimes bugs happen and you need to unlock servers manually. Of course, you have to be really cautious on that but as a last resort, you can break lock with **decapod-admin**.

Overview

```
root@7252bfd5947d:/# decapod-admin locked-servers get-all -h
Usage: decapod-admin locked-servers get-all [OPTIONS]

  List locked servers

Options:
  -f, --output-format [json|csv]  Format of the output  [default: json]
  -h, --help                      Show this message and exit.
root@7252bfd5947d:/# decapod-admin locked-servers --help
Usage: decapod-admin locked-servers [OPTIONS] COMMAND [ARGS]...

  Commands to manage locked servers.

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

Commands:
  get-all  List locked servers
  unlock   Unlock servers.

root@7252bfd5947d:/# decapod-admin locked-servers get-all --help
Usage: decapod-admin locked-servers get-all [OPTIONS]

  List locked servers

Options:
  -f, --output-format [json|csv]  Format of the output  [default: json]
  -h, --help                      Show this message and exit.

root@7252bfd5947d:/# decapod-admin locked-servers unlock --help
Usage: decapod-admin locked-servers unlock [OPTIONS] SERVER_ID...

  Unlock servers.

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

Password Reset

Sometimes it is required to reset a password for a user. Of course, there is well defined procedure of user password resetting but sometimes you just have to change the password bypassing official procedure (e.g user has obsolete, not working email).

Or if you want to change default login/password pair from `root`/`root` to something more secure.

Overview

```
$ decapod-admin password-reset -h
Usage: decapod-admin password-reset [OPTIONS] USER_ID

  Explicitly reset user password.

  Despite the fact that user can request password reset by himself,
  sometimes it is necessary to reset password manually, explicitly and get
  new one immediately.

  Or you may want to change password for user without working email (e.g
  default root user).

Options:
  -p, -password TEXT  New password to use. Empty value means generate password
                      and print after.
  -h, --help          Show this message and exit.
```

If you do not pass new password in commandline, **`decapod-admin`** will bring a prompt and asks you to enter new password.

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

If you do not pass any password, tool will generate one for you and output on the stdout.

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

Also, *admin* service serves documentation you are reading so Decapod has bundled documentation within container. To access documentation, check `DECAPOD_DOCS_PORT` environment variable (default is **9998**). So, if you access Decapod like **http://10.0.0.10:9999**, docs will be served on **http://10.0.0.10:9998**.

See also:

- jq
- yaql
- jmespath-terminal
- jp

CHAPTER 15

Decapod API

This document covers Decapod API internals.

Contents

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 *API 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*, ***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.

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_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"
        ]
    }
]
```

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_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"
            ]
        },
        {
            "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_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_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