MicroCeph

Canonical Group Ltd

Jun 17, 2025

CONTENTS

MicroCeph is the easiest way to get up and running with Ceph.

MicroCeph is a lightweight way of deploying and managing a Ceph cluster. Ceph is a highly scalable, open-source distributed storage system designed to provide excellent performance, reliability, and flexibility for object, block, and file-level storage.

Ceph cluster management is streamlined by simplifying key distribution, service placement, and disk administration for quick, effortless deployment and operations. This applies to clusters that span private clouds, edge clouds, as well as home labs and single workstations.

MicroCeph is focused on providing a modern deployment and management experience to Ceph administrators and storage software developers.

CHAPTER

ONE

IN THIS DOCUMENTATION

Tutorial

Start here: a hands-on introduction to MicroCeph for new users

How-to guides

Step-by-step guides covering key operations and common tasks

Reference

Technical information - specifications, APIs, architecture

Explanation

Discussion and clarification of key topics

CHAPTER

TWO

PROJECT AND COMMUNITY

MicroCeph is a member of the Ubuntu family. It's an open source project that warmly welcomes community projects, contributions, suggestions, fixes and constructive feedback.

- We follow the Ubuntu community Code of conduct
- Contribute to the project on GitHub (documentation contributions go under the `docs` directory)
- GitHub is also used as our bug tracker
- To speak with us, you can find us on Matrix in Ubuntu Ceph or the ~openstack-charms channel on Mattermost

2.1 Tutorials

The pages in this section provide step-by-step instructions for completing common MicroCeph tasks. They do not assume special Linux knowledge nor do they require any deep understanding of Ceph.

2.1.1 Single-node install

This tutorial will show how to install MicroCeph on a single machine, thereby creating a single-node "cluster".

The above will be achieved through the use of loop files placed on the root disk, which is a convenient way for setting up small test and development clusters.

> **Warning**
>
> Using dedicated block devices will result in the best IOPS performance for connecting clients. Basing a Ceph cluster on a single disk also necessarily leads to a common failure domain for all OSDs. For these reasons, loop files should not be used in production environments.

Install the software

Install the most recent stable release of MicroCeph:

```
sudo snap install microceph
```

Next, prevent the software from being auto-updated:

```
sudo snap refresh --hold microceph
```

> **Caution**
>
> Allowing the snap to be auto-updated can lead to unintended consequences. In enterprise environments especially, it is better to research the ramifications of software changes before those changes are implemented.

Initialise the cluster

Begin by initialising the cluster with the **`cluster bootstrap`** command:

```
sudo microceph cluster bootstrap
```

Then look at the status of the cluster with the **`status`** command:

```
sudo microceph status
```

It should look similar to the following:

```
MicroCeph deployment summary:
- node-mees (10.246.114.49)
    Services: mds, mgr, mon
      Disks: 0
```

Here, the machine's hostname of 'node-mees' is given along with its IP address of '10.246.114.49'. The MDS, MGR, and MON services are running but there is not yet any storage available.

Add storage

Three OSDs will be required to form a minimal Ceph cluster. In a production system, typically we would assign a physical block device to an OSD. However for this tutorial, we will make use of file backed OSDs for simplicity.

Add the three file-backed OSDs to the cluster by using the **`disk add`** command. In the example, three 4GiB files are being created:

```
sudo microceph disk add loop,4G,3
```

> **Note**
>
> Although you can adjust the file size and file number to your needs, with a recommended minimum of 2GiB per OSD, there is no obvious benefit to running more than three OSDs via loop files. Be wary that an OSD, whether based on a physical device or a file, is resource intensive.

Recheck status:

```
sudo microceph status
```

The output should now show three disks and the additional presence of the OSD service:

```
MicroCeph deployment summary:
- node-mees (10.246.114.49)
    Services: mds, mgr, mon, osd
      Disks: 3
```

Manage the cluster

Your Ceph cluster is now deployed and can be managed by following the resources found in the *How-to* section.

The cluster can also be managed using native Ceph tooling if snap-level commands are not yet available for a desired task:

```
sudo ceph status
```

The cluster built during this tutorial gives the following output:

```
cluster:
  id:     4c2190cd-9a31-4949-a3e6-8d8f60408278
  health: HEALTH_OK

services:
  mon: 1 daemons, quorum node-mees (age 7d)
  mgr: node-mees(active, since 7d)
  osd: 3 osds: 3 up (since 7d), 3 in (since 7d)

data:
  pools:   1 pools, 1 pgs
  objects: 2 objects, 577 KiB
  usage:   96 MiB used, 2.7 TiB / 2.7 TiB avail
  pgs:     1 active+clean
```

2.1.2 Multi-node install

This tutorial will show how to install MicroCeph on three machines, thereby creating a multi-node cluster. For this tutorial, we will utilise physical block devices for storage.

Ensure storage requirements

Three OSDs will be required to form a minimal Ceph cluster. This means that, on each of the three machines, one entire disk must be allocated for storage.

The disk subsystem can be inspected with the **`lsblk`** command. In this tutorial, the command's output on each machine looks very similar to what's shown below. Any output related to possible loopback devices has been suppressed for the purpose of clarity:

```
lsblk | grep -v loop

NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINTS
vda    252:0    0    40G  0 disk
├─vda1 252:1    0     1M  0 part
└─vda2 252:2    0    40G  0 part /
vdb    252:16   0    20G  0 disk
```

For the example cluster, each machine will use `/dev/vdb` for storage.

Prepare the three machines

On each of the three machines we will need to:

- install the software
- disable auto-updates of the software

Below we'll show these steps explicitly on **node-1**, which we'll call the primary node.

Install the most recent stable release of MicroCeph:

```
sudo snap install microceph
```

Prevent the software from being auto-updated:

```
sudo snap refresh --hold microceph
```

Caution

Allowing the snap to be auto-updated can lead to unintended consequences. In enterprise environments especially, it is better to research the ramifications of software changes before those changes are implemented.

Repeat the above two steps for node-2 and node-3.

Prepare the cluster

On **node-1** we will now:

- initialise the cluster
- create registration tokens

Initialise the cluster with the **`cluster bootstrap`** command:

```
sudo microceph cluster bootstrap
```

Tokens are needed to join the other two nodes to the cluster. Generate these with the **`cluster add`** command.

Token for node-2:

```
sudo microceph cluster add node-2

eyJuYW1lIjoibm9kZS0yIiwic2VjcmV0IjoiYmRjMzZlOWJmNmIzNzhiYzMwY2ZjOWVmMzRjNDM5YzNlZTMzMTlmZDIyZjkxNmJhMTI
```

Token for node-3:

```
sudo microceph cluster add node-3

eyJuYW1lIjoibm9kZS0zIiwic2VjcmV0IjoiYTZjYWJjOTZiNDJkYjg0YTRkZTFiY2MzY2VkYTI1M2Y4MTU1ZTNhYjAwYWUyOWY1MDA
```

Keep these tokens in a safe place. They'll be needed in the next step.

Note

Tokens are randomly generated; each one is unique.

Join the non-primary nodes to the cluster

The **`cluster join`** command is used to join nodes to a cluster.

On **node-2**, add the machine to the cluster using the token assigned to node-2:

```
sudo microceph cluster join␣
↪eyJuYW1lIjoibm9kZS0yIiwic2VjcmV0IjoiYmRjMzZlOWJmNmIzNzhiYzMwY2ZjOWVmMzRjNDM5YzNlZTMzMTlmZDIyZjkxNmJhM
```

On **node-3**, add the machine to the cluster using the token assigned to node-3:

```
sudo microceph cluster join␣
↪eyJuYW1lIjoibm9kZS0zIiwic2VjcmV0IjoiYTZjYWJjOTZiNDJkYjg0YTRkZTFiY2MzY2VkYTI1M2Y4MTU1ZTNhYjAwYWUyOWY1M
```

Add storage

> **Warning**
>
> This step will remove the data found on the target storage disks. Make sure you don't lose data unintentionally.

On **each** of the three machines, use the **`disk add`** command to add storage:

```
sudo microceph disk add /dev/vdb --wipe
```

Adjust the above command per machine according to the storage disks at your disposal.

Check MicroCeph status

On any of the three nodes, the **`status`** command can be invoked to check the status of MicroCeph:

```
sudo microceph status

MicroCeph deployment summary:
- node-01 (10.246.114.11)
  Services: mds, mgr, mon, osd
  Disks: 1
- node-02 (10.246.114.47)
  Services: mds, mgr, mon, osd
  Disks: 1
- node-03 (10.246.115.11)
  Services: mds, mgr, mon, osd
  Disks: 1
```

Machine hostnames are given along with their IP addresses. The MDS, MGR, MON, and OSD services are running and each node is supplying a single disk, as expected.

Manage the cluster

Your Ceph cluster is now deployed and can be managed by following the resources found in the *Howto* section.

The cluster can also be managed using native Ceph tooling if snap-level commands are not yet available for a desired task:

```
ceph status
```

This gives:

```
cluster:
  id:     cf16e5a8-26b2-4f9d-92be-dd3ac9602ebf
```

(continues on next page)

(continued from previous page)

```
  health: HEALTH_OK

services:
  mon: 3 daemons, quorum node-01,node-02,node-03 (age 14m)
  mgr: node-01(active, since 43m), standbys: node-02, node-03
  osd: 3 osds: 3 up (since 4s), 3 in (since 6s)

data:
  pools:   1 pools, 1 pgs
  objects: 0 objects, 0 B
  usage:   336 MiB used, 60 GiB / 60 GiB avail
  pgs:     100.000% pgs unknown
           1 unknown
```

2.2 How-to guides

These how-to guides will cover key operations and processes in MicroCeph.

2.2.1 Configuring network keys

The MicroCeph cluster configuration CLI supports setting, getting, resetting and listing supported config keys mentioned below.

Table 1: Supported Config Keys

Key	Description
cluster_network	Set this key to desired CIDR to configure cluster network
public_network	Set this key to desired CIDR to configure public network

1. Supported config keys can be configured using the 'set' command:

```
$ sudo microceph cluster config set cluster_network 10.5.0.0/16
```

2. Config value for a particular key could be queried using the 'get' command:

```
$ sudo microceph cluster config get cluster_network
+---+-----------------+-------------+
| # |       KEY       |     VALUE   |
+---+-----------------+-------------+
| 0 | cluster_network | 10.5.0.0/16 |
+---+-----------------+-------------+
```

3. A list of all the configured keys can be fetched using the 'list' command:

```
$ sudo microceph cluster config set public_network 10.5.0.0/16
$ sudo microceph cluster config list
+---+-----------------+-------------+
| # |       KEY       |     VALUE   |
+---+-----------------+-------------+
| 0 | cluster_network | 10.5.0.0/16 |
+---+-----------------+-------------+
```

(continues on next page)

(continued from previous page)

```
| 1 | public_network  | 10.5.0.0/16 |
+---+-----------------+-------------+
```

4. Resetting a config key (i.e. setting the key to its default value) can performed using the 'reset' command:

```
$ sudo microceph cluster config reset cluster_network
$ sudo microceph cluster config reset public_network
$ sudo microceph cluster config list
+---+-----+-------+
| # | KEY | VALUE |
+---+-----+-------+
```

For more explanations and implementation details refer to *explanation*

2.2.2 Enabling Prometheus Alertmanager alerts

Pre-Requisite

In order to configure alerts, your MicroCeph deployment must enable metrics collections with Prometheus. Follow *this How-To* if you haven't configured it. Also, Alertmanager is distributed as a separate binary which should be installed and running.

Introduction

Prometheus Alertmanager handles alerts sent by the Prometheus server. It takes care of deduplicating, grouping, and routing them to the correct receiver integration such as email. It also takes care of silencing and inhibition of alerts.

Alerts are configured using Alerting Rules. These rules allows the user to define alert conditions using Prometheus expressions. Ceph is designed to be configurable with Alertmanager, you can use the default set of alerting rules provided below to get basic alerts from your MicroCeph deployments.

The default alert rules can be downloaded from `here`

Configuring Alert rules

Alerting rules and Alertmanager targets are configured in Prometheus using the same config file we used to configure scraping targets.

A simple configuration file with scraping targets, Alertmanager and alerting rules is provided below:

```
# microceph.yaml
global:
    external_labels:
        monitor: 'microceph'

# Scrape Job
scrape_configs:
  - job_name: 'microceph'

    # Ceph's default for scrape_interval is 15s.
    scrape_interval: 15s

    # List of all the ceph-mgr instances along with default (or configured) port.
    static_configs:
    - targets: ['10.245.165.103:9283', '10.245.165.205:9283', '10.245.165.94:9283']
```

(continues on next page)

(continued from previous page)

```
rule_files: # path to alerting rules file.
  - /home/ubuntu/prometheus_alerts.yaml

alerting:
  alertmanagers:
    - static_configs:
      - targets: # Alertmanager <HOST>:<PORT>
        - "10.245.167.132:9093"
```

Start Prometheus with provided configuration file.

```
prometheus --config.file=microceph.yaml
```

Click on the 'Alerts' tab on Prometheus dashboard to view the configured alerts:

Look we already have an active 'CephHealthWarning' alert! (shown in red) while the other configured alerts are inactive (shown in green). Hence, Alertmanager is configured and working.

2.2.3 Enabling metrics collection with Prometheus

Introduction

Metrics play an important role in understanding the operation of your MicroCeph deployment. These metrics or measurements form the basis for analysing and understanding your cluster's behaviour and are essential for providing reliable services.

A popular and mature open-source tool used for scraping and recording metrics over time is Prometheus. Ceph is also designed to be easily integratable with Prometheus. This tutorial documents the procedure and related information for configuring Prometheus to scrape MicroCeph's metrics endpoint.

Setup

The diagram above describes how the metrics endpoint is served by ceph-mgr and scraped by Prometheus on a service level. Another thing to notice is that at any given time only one of the mgr module is active and responsible for receiving MgrReports and serving them i.e. only one instance of ceph-mgr serves the metrics endpoint. As the active Mgr instance can be changing over time, standard practice is to scrape all the mgr instances when monitoring a Ceph cluster.

Enabling Ceph-Mgr Prometheus module

Ceph-Mgr Prometheus module is responsible for serving the metrics endpoint which can then be scraped by Prometheus itself. We can enable the module by executing the following command on a MicroCeph node:

```
ceph mgr module enable prometheus
```

Configuring metrics endpoint

By default, it will accept HTTP requests on port 9283 on all IPv4 and IPv6 addresses on the host. However this can configured using the following ceph-mgr config keys to fine tune to requirements.

```
ceph config set mgr mgr/prometheus/server_addr <addr>
ceph config set mgr mgr/prometheus/port <port>
```

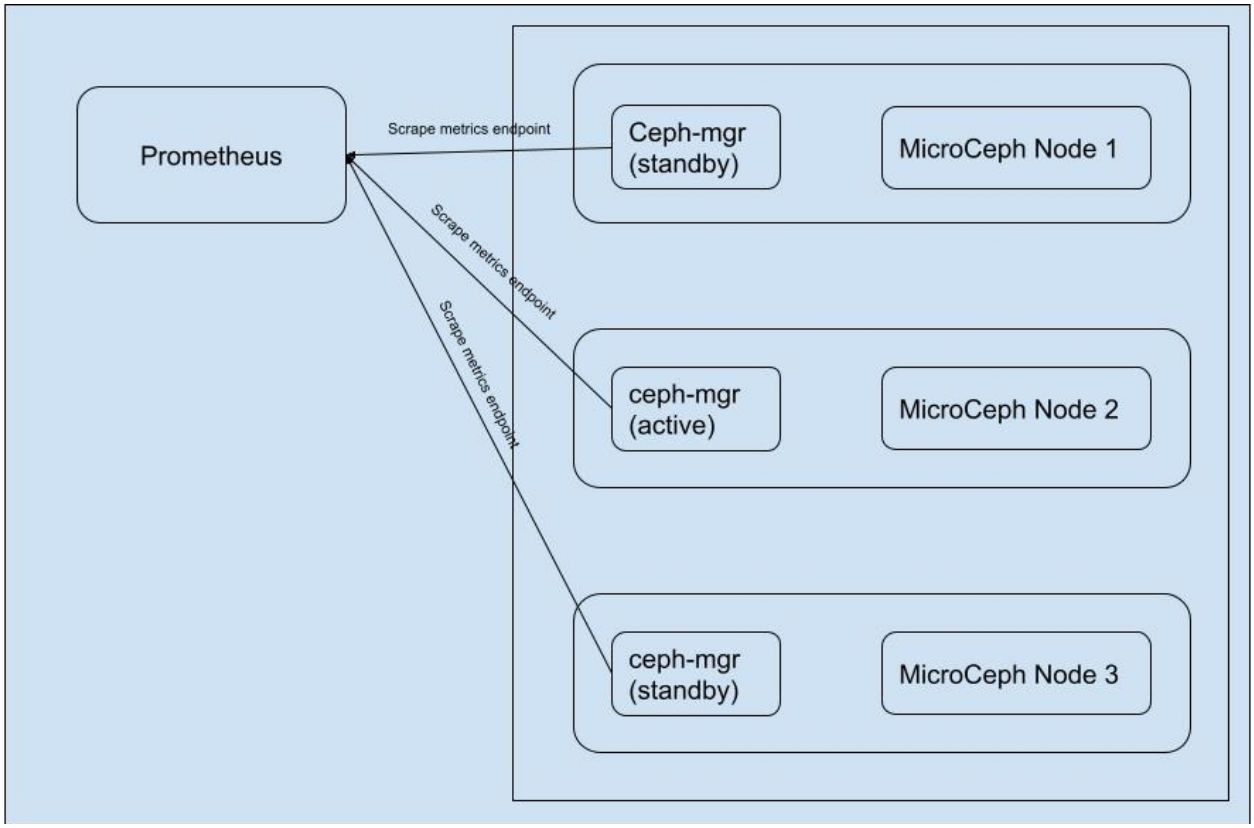

Fig. 1: Prometheus service scraping endpoints of a multi-node MicroCeph cluster.

For details on how metrics endpoint can be further configured visit Ceph Prometheus module

Configuring Prometheus to scrape MicroCeph

Prometheus uses YAML file based configuration of scrape targets. While Prometheus supports an extensive list of configurations that is out of the scope of this document. For details visit Prometheus configuration

A simple configuration file is provided below:

```
# microceph.yaml
global:
    external_labels:
        monitor: 'microceph'

# Scrape Job
scrape_configs:
  - job_name: 'microceph'

    # Ceph's default for scrape_interval is 15s.
    scrape_interval: 15s

    # List of all the ceph-mgr instances along with default (or configured) port.
    static_configs:
    - targets: ['10.245.165.103:9283', '10.245.165.205:9283', '10.245.165.94:9283']
```

Start Prometheus with provided configuration file.

```
prometheus --config.file=microceph.yaml
```

The default port used is 9090 hence collected metrics can be observed at <prometheus_addr>:9090 as:

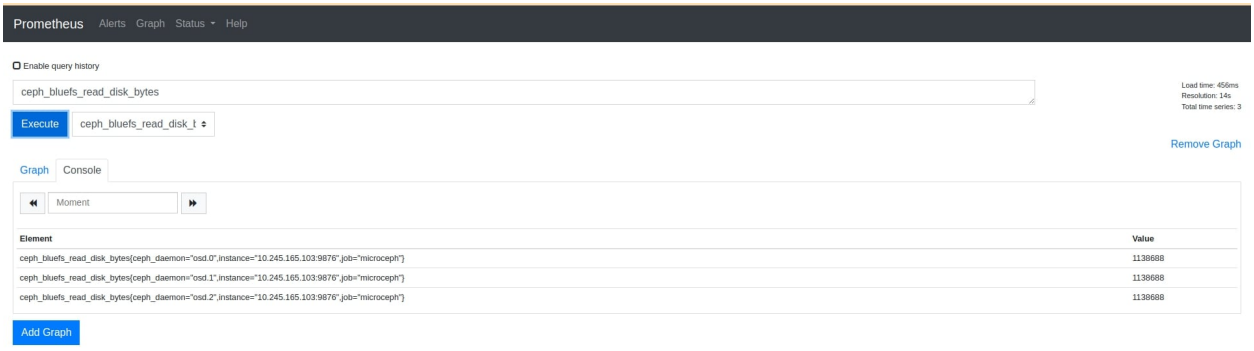

Fig. 2: A Prometheus console displaying scraped metric from MicroCeph cluster.

2.2.4 Enabling additional service instances

To ensure a base level of resiliency, MicroCeph will always try to enable a sufficient number of instances for certain services in the cluster. This number is set to three by default.

The services affected by this include:

- MON (Monitor service)
- MDS (Metadata service)
- MGR (Manager service)

Cluster designs that call for extra service instances, however, can be satisfied by manual means. In addition to the above-listed services, the following service can be added manually to a node:

- RGW (RADOS Gateway service)

This is the purpose of the **enable** command. It manually enables a new instance of a service on a node.

The syntax is:

```
sudo microceph enable <service> --target <destination> ...
```

Where the service value is one of 'mon', 'mds', 'mgr', and 'rgw'. The destination is a node name as discerned by the output of the **status** command:

```
sudo microceph status
```

For a given service, the **enable** command may support extra parameters. These can be discovered by querying for help for the respective service:

```
sudo microceph enable <service> --help
```

Example: enable an RGW service

First check the status of the cluster to get node names and an overview of existing services:

```
sudo microceph status

MicroCeph deployment summary:
- node1-2c3eb41e-14e8-465d-9877-df36f5d80922 (10.111.153.78)
  Services: mds, mgr, mon, osd
  Disks: 3
- workbook (192.168.29.152)
  Services: mds, mgr, mon
  Disks: 0
```

View any possible extra parameters for the RGW service:

```
sudo microceph enable rgw --help
```

To enable the RGW service on node1 and specify a value for extra parameter *port*:

```
sudo microceph enable rgw --target node1 --port 8080
```

Finally, view cluster status again and verify expected changes:

```
sudo microceph status

MicroCeph deployment summary:
- node1 (10.111.153.78)
  Services: mds, mgr, mon, rgw, osd
  Disks: 3
- workbook (192.168.29.152)
  Services: mds, mgr, mon
  Disks: 0
```

2.2.5 Migrating automatically-provisioned services

MicroCeph deploys automatically-provisioned Ceph services when needed. These services include:

- MON - Monitor service
- MDS - Metadata service
- MGR - Manager service

It can however be useful to have the ability to move (or migrate) these services from one node to another. This may be desirable during a maintenance window for instance where these services must remain available.

This is the purpose of the **`cluster migrate`** command. It enables automatically-provisioned services on a target node and disables them on the source node.

The syntax is:

```
sudo microceph cluster migrate <source> <destination>
```

Where the source and destination are node names that are available via the **`status`** command:

```
sudo microceph status
```

Post-migration, the **`status`** command can also be used to verify the distribution of services among nodes.

Notes:

- It's not possible, nor useful, to have more than one instance of an automatically-provisioned service on any given node.
- RADOS Gateway services are not considered to be of the automatically-provisioned type; they are enabled and disabled explicitly on a node.

2.2.6 Configure RBD client cache in MicroCeph

MicroCeph supports setting, resetting, and listing client configurations which are exported to ceph.conf and are used by tools like qemu directly for configuring rbd cache. Below are the supported client configurations.

Table 2: Supported Config Keys

Key	Description
rbd_cache	Enable caching for RADOS Block Device (RBD).
rbd_cache_size	The RBD cache size in bytes.
rbd_cache_writethrough_until_flu	The number of seconds dirty data is in the cache before writeback starts.
rbd_cache_max_dirty	The dirty limit in bytes at which the cache triggers write-back. If 0, uses write-through caching.
rbd_cache_target_dirty	The dirty target before the cache begins writing data to the data storage. Does not block writes to the cache.

1. Supported config keys can be configured using the 'set' command:

```
$ sudo microceph client config set rbd_cache true
$ sudo microceph client config set rbd_cache false --target alpha
$ sudo microceph client config set rbd_cache_size 2048MiB --target beta
```

> **Note**
>
> Host level configuration changes can be made by passing the relevant hostname as the –target parameter.

2. All the client configs can be queried using the 'list' command.

```
$ sudo microceph cluster config list
+---+----------------+---------+----------+
| # |      KEY       |  VALUE  |   HOST   |
+---+----------------+---------+----------+
| 0 | rbd_cache      | true    | beta     |
+---+----------------+---------+----------+
| 1 | rbd_cache      | false   | alpha    |
+---+----------------+---------+----------+
| 2 | rbd_cache_size | 2048MiB | beta     |
+---+----------------+---------+----------+
```

Similarly, all the client configs of a particular host can be queried using the –target parameter.

```
$ sudo microceph cluster config list --target beta
+---+----------------+---------+----------+
| # |      KEY       |  VALUE  |   HOST   |
+---+----------------+---------+----------+
| 0 | rbd_cache      | true    | beta     |
+---+----------------+---------+----------+
| 1 | rbd_cache_size | 2048MiB | beta     |
+---+----------------+---------+----------+
```

3. A particular config key can be queried for using the 'get' command:

```
$ sudo microceph cluster config list
+---+----------------+---------+----------+
| # |      KEY       |  VALUE  |   HOST   |
+---+----------------+---------+----------+
| 0 | rbd_cache      | true    | beta     |
```

(continues on next page)

(continued from previous page)

```
+---+----------------+---------+----------+
| 1 | rbd_cache      | false   | alpha    |
+---+----------------+---------+----------+
```

Similarly, –target parameter can be used with get command to query for a particular config key/hostname pair.

```
$ sudo microceph cluster config rbd_cache --target alpha
+---+----------------+---------+----------+
| # |      KEY       |  VALUE  |   HOST   |
+---+----------------+---------+----------+
| 0 | rbd_cache      | false   | alpha    |
+---+----------------+---------+----------+
```

4. Resetting a config key (i.e. removing the configured key/value) can performed using the 'reset' command:

```
$ sudo microceph cluster config reset rbd_cache_size
$ sudo microceph cluster config list
 +---+----------------+---------+----------+
 | # |      KEY       |  VALUE  |   HOST   |
 +---+----------------+---------+----------+
 | 0 | rbd_cache      | true    | beta     |
 +---+----------------+---------+----------+
 | 1 | rbd_cache      | false   | alpha    |
 +---+----------------+---------+----------+
```

This operation can also be performed for a specific host as follows:

```
$ sudo microceph cluster config reset rbd_cache --target alpha
$ sudo microceph cluster config list
 +---+----------------+---------+----------+
 | # |      KEY       |  VALUE  |   HOST   |
 +---+----------------+---------+----------+
 | 0 | rbd_cache      | true    | beta     |
 +---+----------------+---------+----------+
```

2.2.7 Removing a disk

Overview

There are valid reasons for wanting to remove a disk from a Ceph cluster. A common use case is the need to replace one that has been identified as nearing its shelf life. Another example is the desire to scale down the cluster through the removal of a cluster node (machine).

The following resources provide extra context to the disk removal operation:

- the *Cluster scaling* page
- the *disk* command reference

> **Note**
>
> This feature is currently only supported in channel `latest/edge` of the **`microceph`** snap.

Procedure

First get an overview of the cluster and its OSDs:

```
ceph status
```

Example output:

```
cluster:
  id:     cf16e5a8-26b2-4f9d-92be-dd3ac9602ebf
  health: HEALTH_OK

services:
  mon: 3 daemons, quorum node-01,node-02,node-03 (age 41h)
  mgr: node-01(active, since 41h), standbys: node-02, node-03
  osd: 5 osds: 5 up (since 22h), 5 in (since 22h); 1 remapped pgs

data:
  pools:   1 pools, 1 pgs
  objects: 2 objects, 577 KiB
  usage:   105 MiB used, 1.9 TiB / 1.9 TiB avail
  pgs:     2/6 objects misplaced (33.333%)
           1 active+clean+remapped
```

Then determine the ID of the OSD associated with the disk with the (native Ceph) **`ceph osd tree`** command:

```
ceph osd tree
```

Sample output:

```
ID  CLASS  WEIGHT   TYPE NAME             STATUS  REWEIGHT  PRI-AFF
-1         1.87785  root default
-5         1.81940      host node-mees
 3         0.90970          osd.3             up   1.00000  1.00000
 4         0.90970          osd.4             up   1.00000  1.00000
-2         0.01949      host node-01
 0         0.01949          osd.0             up   1.00000  1.00000
-3         0.01949      host node-02
 1         0.01949          osd.1             up   1.00000  1.00000
-4         0.01949      host node-03
 2         0.01949          osd.2             up   1.00000  1.00000
```

Let's assume that our target disk is on host 'node-mees' and has an associated OSD whose ID is 'osd.4'.

To remove the disk:

```
sudo microceph disk remove osd.4
```

Verify that the OSD has been removed:

```
ceph osd tree
```

Output:

```
ID  CLASS  WEIGHT   TYPE NAME             STATUS  REWEIGHT  PRI-AFF
-1         0.96815  root default
```

(continues on next page)

(continued from previous page)

```
-5         0.90970      host node-mees
 3    hdd  0.90970          osd.3              up   1.00000  1.00000
-2         0.01949      host node-01
 0    hdd  0.01949          osd.0              up   1.00000  1.00000
-3         0.01949      host node-02
 1    hdd  0.01949          osd.1              up   1.00000  1.00000
-4         0.01949      host node-03
 2    hdd  0.01949          osd.2              up   1.00000  1.00000
```

Finally, confirm cluster status and health:

```
ceph status
```

Output:

```
cluster:
  id:     cf16e5a8-26b2-4f9d-92be-dd3ac9602ebf
  health: HEALTH_OK

services:
  mon: 3 daemons, quorum node-01,node-02,node-03 (age 4m)
  mgr: node-01(active, since 4m), standbys: node-02, node-03
  osd: 4 osds: 4 up (since 4m), 4 in (since 4m)

data:
  pools:   1 pools, 1 pgs
  objects: 2 objects, 577 KiB
  usage:   68 MiB used, 991 GiB / 992 GiB avail
  pgs:     1 active+clean
```

2.3 Reference

MicroCeph reference material is divided into several categories.

2.3.1 Commands

List of MicroCeph commands.

client

Manages MicroCeph clients

Usage:

```
microceph client [flags]
microceph client [command]
```

Available commands:

```
config      Manage Ceph Client configs
```

Global options:

```
-d, --debug       Show all debug messages
-h, --help        Print help
    --state-dir   Path to store state information
-v, --verbose     Show all information messages
    --version     Print version number
```

config

Manages Ceph Cluster configs.

Usage:

```
microceph cluster config [flags]
microceph cluster config [command]
```

Available Commands:

```
get         Fetches specified Ceph Client config
list        Lists all configured Ceph Client configs
reset       Removes specified Ceph Client configs
set         Sets specified Ceph Client config
```

config set

Sets specified Ceph Client config

Usage:

```
microceph client config set <Key> <Value> [flags]
```

Flags:

```
--target string   Specify a microceph node the provided config should be applied to.␣
↪(default "*")
--wait            Wait for configs to propagate across the cluster. (default true)
```

config get

Fetches specified Ceph Client config

Usage:

```
microceph client config get <key> [flags]
```

Flags:

```
--target string   Specify a microceph node the provided config should be applied to.␣
↪(default "*")
```

config list

Lists all configured Ceph Client configs

Usage:

```
microceph client config list [flags]
```

Flags:

```
--target string   Specify a microceph node the provided config should be applied to.␣
↪(default "*")
```

config reset

Removes specified Ceph Client configs

Usage:

```
microceph client config reset <key> [flags]
```

Flags:

```
--target string          Specify a microceph node the provided config should be applied␣
↪to. (default "*")
--wait                   Wait for required ceph services to restart post config reset.␣
↪(default true)
--yes-i-really-mean-it   Force microceph to reset all client config records for given␣
↪key.
```

cluster

Manages the MicroCeph cluster.

Usage:

```
microceph cluster [flags]
microceph cluster [command]
```

Available commands:

```
add         Generates a token for a new server
bootstrap   Sets up a new cluster
config      Manage Ceph Cluster configs
join        Joins an existing cluster
list        List servers in the cluster
migrate     Migrate automatic services from one node to another
remove      Removes a server from the cluster
sql         Runs a SQL query against the cluster database
```

Global options:

```
-d, --debug       Show all debug messages
-h, --help        Print help
    --state-dir   Path to store state information
-v, --verbose     Show all information messages
    --version     Print version number
```

add

Generates a token for a new server

Usage:

```
microceph cluster add <NAME> [flags]
```

bootstrap

Sets up a new cluster

Usage:

```
microceph cluster bootstrap [flags]
```

Flags:

```
--mon-ip string   Public address for bootstrapping ceph mon service.
```

config

Manages Ceph Cluster configs.

Usage:

```
microceph cluster config [flags]
microceph cluster config [command]
```

Available Commands:

```
get         Get specified Ceph Cluster config
list        List all set Ceph level configs
reset       Clear specified Ceph Cluster config
set         Set specified Ceph Cluster config
```

config get

Gets specified Ceph Cluster config.

Usage:

```
microceph cluster config get <key> [flags]
```

config list

Lists all set Ceph level configs.

Usage:

```
microceph cluster config list [flags]
```

config reset

Clears specified Ceph Cluster config.

Usage:

```
microceph cluster config reset <key> [flags]
```

Flags:

```
--wait   Wait for required ceph services to restart post config reset.
```

config set

Sets specified Ceph Cluster config.

Usage:

```
microceph cluster config set <Key> <Value> [flags]
```

Flags:

```
--wait   Wait for required ceph services to restart post config set.
```

join

Joins an existing cluster.

Usage:

```
microceph cluster join <TOKEN> [flags]
```

list

Lists servers in the cluster.

Usage:

```
microceph cluster list [flags]
```

migrate

Migrates automatic services from one node to another.

Usage:

```
microceph cluster migrate <SRC> <DST [flags]
```

remove

Removes a server from the cluster.

Syntax:

```
microceph cluster remove <NAME> [flags]
```

Flags:

```
-f, --force   Forcibly remove the cluster member
```

sql

Runs a SQL query against the cluster database.

Usage:

```
microceph cluster sql <query> [flags]
```

disable

Disables a feature on the cluster

Usage:

```
microceph disable [flags]
microceph disable [command]
```

Available Commands:

```
rgw         Disable the RGW service on this node
```

Global flags:

```
-d, --debug       Show all debug messages
-h, --help        Print help
    --state-dir   Path to store state information
-v, --verbose     Show all information messages
    --version     Print version number
```

disk

Manages disks in MicroCeph.

Usage:

```
microceph disk [flags]
microceph disk [command]
```

Available commands:

```
add         Add a Ceph disk (OSD)
list        List servers in the cluster
remove      Remove a Ceph disk (OSD)
```

Global flags:

```
-d, --debug       Show all debug messages
-h, --help        Print help
    --state-dir   Path to store state information
-v, --verbose     Show all information messages
    --version     Print version number
```

add

Adds a new Ceph disk (OSD) to the cluster, alongside optional devices for write-ahead logging and database management.

The command takes a parameter <spec> which is either a path to a block device such as /dev/sdb or a specification for one or more loop files.

The specification for loop files is of the form *loop,<size>,<nr>*

- *size* is an integer with *M*, *G*, or *T* suffixes for megabytes, gigabytes, or terabytes.
- *nr* is the number of file-backed loop OSDs to create.

For instance, a spec of *loop,8G,3* will create 3 file-backed loop OSDs of 8GB each.

Usage:

```
microceph disk add <spec> [flags]
```

Flags:

```
--db-device string    The device used for the DB
--db-encrypt          Encrypt the DB device prior to use
--db-wipe             Wipe the DB device prior to use
--encrypt             Encrypt the disk prior to use (only block devices)
--wal-device string   The device used for WAL
--wal-encrypt         Encrypt the WAL device prior to use
--wal-wipe            Wipe the WAL device prior to use
--wipe                Wipe the disk prior to use
```

> **Note**
>
> Only the data device is mandatory. The WAL and DB devices can improve performance by delegating the management of some subsystems to additional block devices. The WAL block device stores the internal journal whereas the DB one stores metadata. Using either of those should be advantageous as long as they are faster than the data device. WAL should take priority over DB if there isn't enough storage for both.
>
> WAL and DB devices can only be used with data devices that reside on a block device, not with loop files. Loop files do not support encryption.

list

List servers in the cluster

Usage:

```
microceph disk list [flags]
```

remove

Removes a single disk from the cluster.

Usage:

```
microceph disk remove <osd-id> [flags]
```

Flags:

```
--bypass-safety-checks               Bypass safety checks
--confirm-failure-domain-downgrade   Confirm failure domain downgrade if required
--timeout int                        Timeout to wait for safe removal (seconds)␣
↪(default: 300)
```

enable

Enables a feature or service on the cluster.

Usage:

```
microceph enable [flags]
microceph enable [command]
```

Available commands:

```
mds         Enable the MDS service on the --target server (default: this server)
mgr         Enable the MGR service on the --target server (default: this server)
mon         Enable the MON service on the --target server (default: this server)
rgw         Enable the RGW service on the --target server (default: this server)
```

Global flags:

```
-d, --debug       Show all debug messages
-h, --help        Print help
    --state-dir   Path to store state information
-v, --verbose     Show all information messages
    --version     Print version number
```

mds

Enables the MDS service on the –target server (default: this server).

Usage:

```
microceph enable mds [--target <server>] [--wait <bool>] [flags]
```

Flags:

```
--target string   Server hostname (default: this server)
--wait            Wait for mds service to be up. (default true)
```

mgr

Enables the MGR service on the –target server (default: this server).

Usage:

```
microceph enable mgr [--target <server>] [--wait <bool>] [flags]
```

Flags:

```
--target string   Server hostname (default: this server)
--wait            Wait for mgr service to be up. (default true)
```

mon

Enables the MON service on the –target server (default: this server).

Usage:

```
microceph enable mon [--target <server>] [--wait <bool>] [flags]
```

Flags:

```
--target string   Server hostname (default: this server)
--wait            Wait for mon service to be up. (default true)
```

rgw

Enables the RGW service on the –target server (default: this server).

Usage:

```
microceph enable rgw [--port <port>] [--target <server>] [--wait <bool>] [flags]
```

Flags:

```
--port int        Service port (default: 80) (default 80)
--target string   Server hostname (default: this server)
--wait            Wait for rgw service to be up. (default true)
```

help

Help provides help for any command in the application. Simply type microceph help [path to command] for full details.

Usage:

```
microceph help [command] [flags]
```

Global flags:

```
-d, --debug       Show all debug messages
-h, --help        Print help
    --state-dir   Path to store state information
-v, --verbose     Show all information messages
    --version     Print version number
```

init

Initialises MicroCeph (in interactive mode).

Usage:

```
microceph init [flags]
```

Global flags:

```
-d, --debug       Show all debug messages
-h, --help        Print help
    --state-dir   Path to store state information
```

(continues on next page)

(continued from previous page)

```
-v, --verbose     Show all information messages
    --version     Print version number
```

status

Reports the status of the cluster.

Usage:

```
microceph status [flags]
```

Global flags:

```
-d, --debug       Show all debug messages
-h, --help        Print help
    --state-dir   Path to store state information
-v, --verbose     Show all information messages
    --version     Print version number
```

2.4 Explanation

Discussion and clarification of key topics

2.4.1 Cluster network configurations

Overview

Network configuration is critical for building a high performance Ceph Storage Cluster.

Ceph clients make requests directly to Ceph OSD Daemons i.e. Ceph does not perform request routing. The OSD Daemons perform data replication on behalf of clients, which means replication and other factors impose additional loads on Ceph Storage Cluster networks. Therefore, to enhance security and stability, it can be advantageous to split public and cluster network traffic so that client traffic flows on a public net while cluster traffic (for replication and backfilling) utilises a separate net. This helps to prevent malicious or malfunctioning clients from disrupting cluster backend operations.

For more details, refer to Ceph Network Config.

Implementation

MicroCeph cluster config subcommands rely on `ceph config` as the single source of truth for config values and for getting/setting the configs. After updating (setting/resetting) a config value, a restart request is sent to other hosts on the MicroCeph cluster for restarting particular daemons. This is done for the change to take effect.

In a multi-node MicroCeph cluster, restarting the daemons is done cautiously in a synchronous manner to prevent cluster outage. The flow diagram below explains the order of execution.

2.4.2 Cluster scaling

Overview

MicroCeph's scalability is courtesy of its foundation on Ceph, which has excellent scaling capabilities. To scale out, either add machines to the existing cluster nodes or introduce additional disks (OSDs) on the nodes.

Note it is strongly recommended to use uniformly-sized machines, particularly with smaller clusters, to ensure Ceph fully utilises all available disk space.

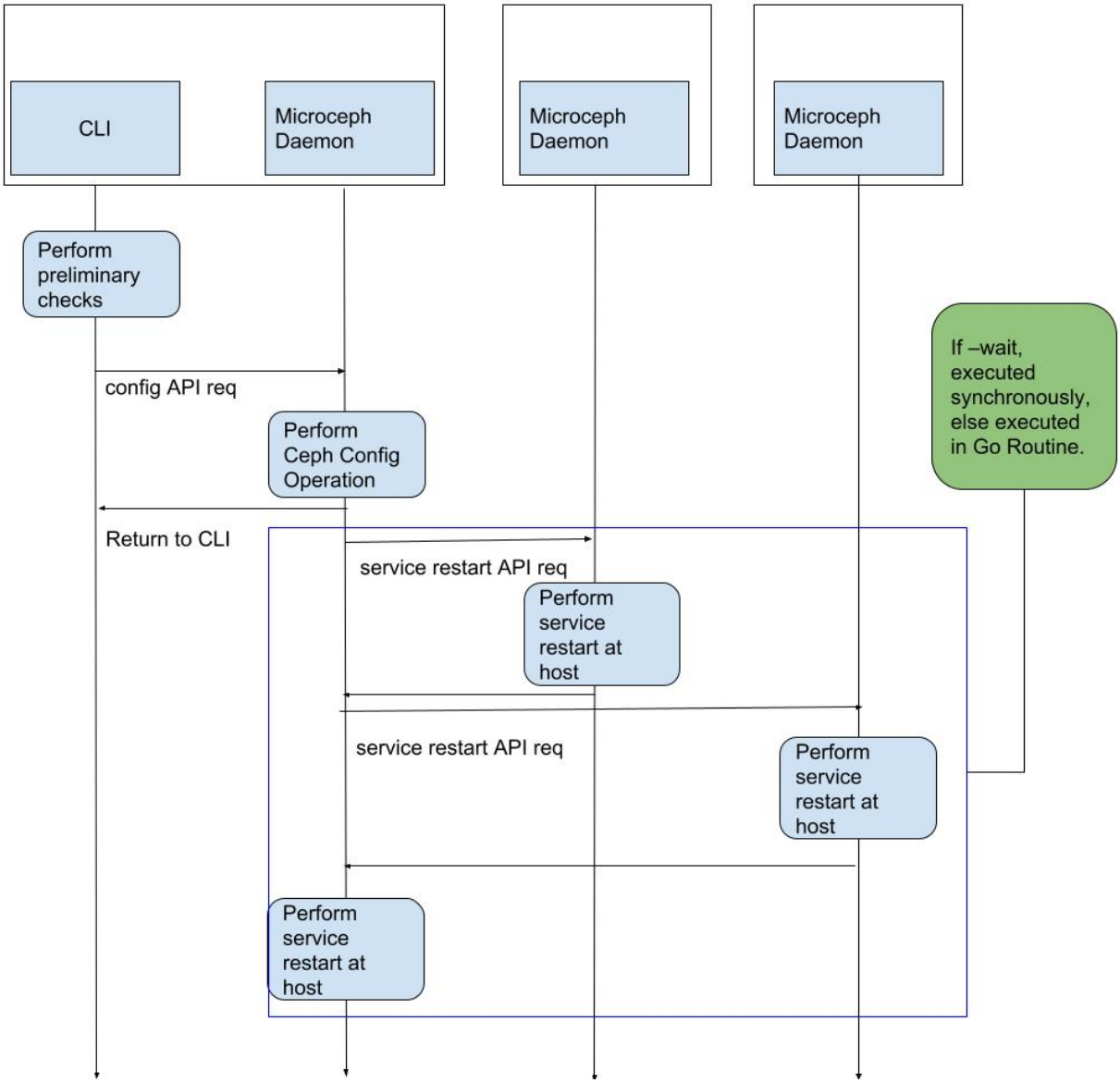

Fig. 3: Execution flow of config set/reset commands in multi-node MicroCeph deployment

Failure Domains

In the realm of Ceph, the concept of failure domains comes into play in order to provide data safety. A failure domain is an entity or a category across which object replicas are spread. This could be OSDs, hosts, racks, or even larger aggregates like rooms or data centres. The key purpose of failure domains is to mitigate the risk of extensive data loss that could occur if a larger aggregate (e.g. machine or rack) crashes or becomes otherwise unavailable.

This spreading of data or objects across various failure domains is managed through the Ceph's Controlled Replication Under Scalable Hashing (CRUSH) rules. The CRUSH algorithm enables Ceph to distribute data replicas over various failure domains efficiently and without any central directory, thus providing consistent performance as you scale.

In simple terms, if one component within a failure domain fails, Ceph's built-in redundancy means your data is still accessible from an alternate location. For instance, with a host-level failure domain, Ceph will ensure that no two replicas are placed on the same host. This prevents loss of more than one replica should a host crash or get disconnected. This extends to higher-level aggregates like racks and rooms as well.

Furthermore, the CRUSH rules ensure that data is automatically re-distributed if parts of the system fail, assuring the resiliency and high availability of your data.

The flipside is that for a given replication factor and failure domain you will need the appropriate number of aggregates. So for the default replication factor of 3 and failure domain at host level you'll need at least 3 hosts (of comparable size); for failure domain rack you'll need at least 3 racks, etc.

Failure Domain Management

MicroCeph implements automatic failure domain management at the OSD and host levels. At the start, CRUSH rules are set for OSD-level failure domain. This makes single-node clusters viable, provided they have at least 3 OSDs.

Scaling Up

As you scale up, the failure domain automatically will be upgraded by MicroCeph. Once the cluster size is increased to 3 nodes having at least one OSD each, the automatic failure domain shifts to the host level to safeguard data even if an entire host fails. This upgrade typically will need some data redistribution which is automatically performed by Ceph.

Scaling Down

Similarly, when scaling down the cluster by removing OSDs or nodes, the automatic failure domain rules will be downgraded, from the host level to the osd level. This is done once a cluster has less than 3 nodes with at least one OSD each. MicroCeph will ask for confirmation if such a downgrade is necessary.

Disk removal

The *disk* command (**`disk remove`**) is used to remove OSDs.

Automatic failure domain downgrades

The removal operation will abort if it would lead to a downgrade in failure domain. In such a case, the command's `--confirm-failure-domain-downgrade` option overrides this behaviour and allows the downgrade to proceed.

Cluster health and safety checks

The removal operation will wait for data to be cleanly redistributed before evicting the OSD. There may be cases however, such as when a cluster is not healthy to begin with, where the redistribution of data is not feasible. In such situations, the command's `--bypass-safety-checks` option disable these safety checks.

> **Warning**
>
> The `--bypass-safety-checks` option is intended as a last resort measure only. Its usage may result in data loss.

Custom Crush Rules

MicroCeph automatically manages two rules, named *microceph_auto_osd* and *microceph_auto_host* respectively; these two rules must not be changed. Users can however freely set custom CRUSH rules anytime. MicroCeph will respect custom rules and not perform any automatic updates for these. Custom CRUSH rules can be useful to implement larger failure domains such as rack- or room-level. At the other end of the spectrum, custom CRUSH rules could be used to enforce OSD-level failure domains for clusters larger than 3 nodes.

Machine Sizing

Maintaining uniformly sized machines is an important aspect of scaling up MicroCeph. This means machines should ideally have a similar number of OSDs and similar disk sizes. This uniformity in machine sizing offers several advantages:

1. Balanced Cluster: Having nodes with a similar configuration drives a balanced distribution of data and load in the cluster. It ensures all nodes are optimally performing and no single node is overstrained, enhancing the cluster's overall efficiency.
2. Space Utilisation: With similar sized machines, Ceph can optimally use all available disk space rather than having some remain underutilised and hence wasted.
3. Easy Management: Uniform machines are simpler to manage as each has similar capabilities and resource needs.

As an example, consider a cluster with 3 nodes with host-level failure domain and replication factor 3, where one of the nodes has significant lower disk space available. That node would effectively bottleneck available disk space, as Ceph needs to ensure one replica of each object is placed on each machine (due to the host-level failure domain).

2.4.3 Full disk encryption

Overview

MicroCeph supports automatic full disk encryption (FDE) on OSDs.

Full disk encryption is a security measure that protects the data on a storage device by encrypting all the information on the disk. FDE helps maintain data confidentiality in case the disk is lost or stolen by rendering the data inaccessible without the correct decryption key or password.

In the event of disk loss or theft, unauthorised individuals are unable to access the encrypted data, as the encryption renders the information unreadable without the proper credentials. This helps prevent data breaches and protects sensitive information from being misused.

FDE also eliminates the need for wiping or physically destroying a disk when it is replaced, as the encrypted data remains secure even if the disk is no longer in use. The data on the disk is effectively rendered useless without the decryption key.

Implementation

Full disk encryption for OSDs has to be requested when adding disks. MicroCeph will then generate a random key, store it in the Ceph cluster configuration, and use it to encrypt the given disk via LUKS/cryptsetup.

Prerequisites

To use FDE, the following prerequisites must be met:

- The installed snapd daemon version must be >= 2.59.1
- The `dm-crypt` kernel module must be available. Note that some cloud-optimised kernels do not ship dm-crypt by default. Check by running `sudo modinfo dm-crypt`
- The snap dm-crypt plug has to be connected, and `microceph.daemon` subsequently restarted:

```
sudo snap connect microceph:dm-crypt
sudo snap restart microceph.daemon
```

Limitations

Warning:

- It is important to note that MicroCeph FDE *only* encompasses OSDs. Other data, such as state information for monitors, logs, configuration etc., will *not* be encrypted by this mechanism.
- Also note that the encryption key will be stored on the Ceph monitors as part of the Ceph key/value store

Usage

FDE for OSDs is activated by passing the optional `--encrypt` flag when adding disks:

```
sudo microceph disk add /dev/sdx --wipe --encrypt
```

Note there is no facility to encrypt an OSD that is already part of the cluster. To enable encryption you will have to take the OSD disk out of the cluster, ensure data is replicated and the cluster converged and is healthy, and then re-introduce the OSD with encryption.

2.4.4 Snap content interface for MicroCeph

Overview

Snap content interfaces enable access to a particular directory from a producer snap. The MicroCeph `ceph-conf` content interface is designed to facilitate access to MicroCeph's configuration and credentials. This interface includes information about MON addresses, enabling a consumer snap to connect to the MicroCeph cluster using this data.

Additionally, the `ceph-conf` content interface also provides version information of the running Ceph software.

Usage

The usage of the `ceph-conf` interface revolves around providing the consuming snap access to necessary configuration details.

Here is how it can be utilised:

- Connect to the `ceph-conf` content interface to gain access to MicroCeph's configuration and credentials.
- The interface exposes a standard `ceph.conf` configuration file as well Ceph keyrings with administrative privileges.
- Use the MON addresses included in the configuration to connect to the MicroCeph cluster.
- The interface provides version information that can be used to set up version-specific clients.

To connect the `ceph-conf` content interface to a consumer snap, use the following command:

```
snap connect <consumer-snap-name>:ceph-conf microceph:ceph-conf
```

Replace <consumer-snap-name> with the name of your consumer snap. Once executed, this command establishes a connection between the consumer snap and the MicroCeph ceph-conf interface.