MicroCeph

Canonical Group Ltd

May 21, 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

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*
- Optionally enable Ubuntu Pro on your Ceph nodes. This is a service that provides the Livepatch Service and the Expanded Security Maintenance (ESM) program.

2.1 Get started

This tutorial will guide you through your first steps with MicroCeph. We will deploy a Ceph cluster on a single node using MicroCeph and store a JPEG image in an S3 bucket managed by MicroCeph.

2.1.1 How you'll do It

You will install MicroCeph, initialise the cluster, and add storage. Then, you will enable the S3-compatible Ceph Object Gateway (RGW) on your node and create an S3 bucket. Finally, you will upload an image to the bucket, consuming the storage via RGW.

As we progress, you will also interact with your cluster by checking its health, adding disks, and enabling RGW.

By the end of this tutorial, after successfully using MicroCeph to store an image, you will have a foundational understanding of how MicroCeph works, and be ready to explore more advanced use cases.

2.1.2 What you'll need

- The latest Ubuntu LTS version. Find Ubuntu release information here.
- 2 CPU cores
- 4 GiB RAM
- 12GiB disk space
- An Internet connection

2.1.3 Install MicroCeph

First, install MicroCeph as a snap package from the Snap Store:

```
sudo snap install microceph
```

Disable the default automatic Snap upgrades to prevent MicroCeph from being updated automatically:

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

Caution

Failing to set this option may result in unintended upgrades, which could critically impact your deployed cluster. To prevent this, all subsequent MicroCeph upgrades must be performed manually.

2.1.4 Initialise your cluster

Next, bootstrap your new Ceph storage cluster:

```
sudo microceph cluster bootstrap
```

This process takes 3 to 5 seconds.

Check the cluster status:

```
sudo microceph status
```

The output should look somewhat as shown below:

`MicroCeph deployment summary:- ubuntu (10.246.114.49) Services:  mds, mgr, mon Disks:  0` Your cluster deployment summary contains your node's hostname (IP address). In our case, it's `ubuntu` (`10.246.114.49`), along with information about the services running and available storage. You'll notice that the cluster is healthy with one node and three services running, but no storage has been allocated yet.

Now that the cluster is initialised, we'll add some storage to the node.

2.1.5 Add storage

Let's add storage disk devices to the node.

We will use loop files, which are file-backed Object Storage Daemons (OSDs) convenient for setting up small test and development clusters. Three OSDs are required to form a minimal Ceph cluster.

Execute the following command:

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

`+-----------+---------+| PATH | STATUS |+-----------+---------+| loop,4G,3 | Success |+-----------+---------+` Success! You have added three OSDs with 4GiB storage to your node.

Recheck the cluster status:

```
sudo microceph status
```

`MicroCeph deployment summary:- ubuntu (10.246.114.49)Services:  mds, mgr, mon, osdDisks: 3` You have successfully deployed a Ceph cluster on a single node.

Remember that we had three services running when the cluster was bootstrapped. Note that we now have four services running, including the newly added `osd` service.

2.1.6 Enable RGW

As mentioned before, we will use the Ceph Object Gateway to interact with the object storage cluster we just deployed.

Enable the RGW daemon on your node

```
sudo microceph enable rgw
```

Note

By default, the `rgw` service uses port 80, which may not always be available. If port 80 is occupied, you can specify an alternative port, such as 8080, by adding the `--port <port-number>` parameter.

Run the status check again to confirm that the `rgw` service is reflected in the status output.

```
sudo microceph status
```

```
MicroCeph deployment summary:- ubuntu (10.246.114.49)Services:  mds, mgr, mon, rgw,
osdDisks:  3
```

Create an RGW user

MicroCeph is packaged with the standard `radosgw-admin` tool that manages the `rgw` service and users. We will now use this tool to create an RGW user called `user`, with the display name `user`.

```
sudo radosgw-admin user create --uid=user --display-name=user
```

The output should include user details as shown below, with auto-generated access and secret keys.

```
 {"user_id":  "user","display_name":  "user","email":  "","suspended":  0,"max_buckets":
1000,"subusers":  [],"keys":  [ { "user":  "user", "access_key":  "NJ7YZ3LYI45M4Q1A08OS",
"secret_key":  "H7OTclVbZIwhd2o0NLPu0D7Ass8ouSKmtSewuYwK", "active":  true,
"create_date":  "2024-11-28T13:07:41.561437Z" }],...
```

Set user secrets

Let's define secrets for this user, setting `access_key` to `foo`, and `--secret-key` to `bar`.

```
sudo radosgw-admin key create --uid=user --key-type=s3 --access-key=foo --secret-key=bar
```

```
 ... [ { "user":  "user", "access_key":  "NJ7YZ3LYI45M4Q1A08OS", "secret_key":
"H7OTclVbZIwhd2o0NLPu0D7Ass8ouSKmtSewuYwK", "active":  true, "create_date":
"2024-11-28T13:07:41.561437Z" }, { "user":  "user", "access_key":  "foo", "secret_key":
"bar", "active":  true, "create_date":  "2024-11-28T13:54:36.065214Z" } ],...
```

2.1.7 Consuming the storage

Access RGW

Before attempting to consume the object storage in the cluster, validate that you can access RGW by running **`curl`** on your node.

Find the IP address of the node running the `rgw` service:

```
sudo microceph status
```

```
MicroCeph deployment summary:- ubuntu (10.246.114.49)Services:  mds, mgr, mon, rgw,
osdDisks:  3
```

Then, run **curl** from this node.

```
curl http://10.246.114.49
```

```
<?xml version="1.0" encoding="UTF-8"?><ListAllMyBucketsResult xmlns="http://s3.amazonaws.
com/doc/2006-03-01/"><Owner><ID>anonymous</ID></Owner><Buckets></Bucket
```

Create an S3 bucket

You have verified that your cluster is accessible via RGW. To interact with S3, we need to make sure that the `s3cmd` utility is installed and configured.

Install and configure `s3cmd`

To install `s3cmd`, run the following command:

```
sudo apt-get install s3cmd
```

Configure the `s3cmd` tool:

```
s3cmd --configure
```

This will invoke an interactive configuration session, and later create a file named `.s3cfg` in your home directory with all the settings chosen in the interactive session.

Remember that we had set secrets for our user earlier; we will use those when prompted to provide a secret key and access key. We'll set our host name (`ubuntu`) as the S3 endpoint, and use the default [US] region.

```
Enter new values or accept defaults in brackets with Enter.Refer to user manual for
detailed description of all options. Access key and Secret key are your identifiers
for Amazon S3. Leave them empty for using the env variables.Access Key:  fooSecret
Key:  barDefault Region [US]: Use "s3.amazonaws.com" for S3 Endpoint and not modify
it to the target Amazon S3.S3 Endpoint [s3.amazonaws.com]:  ubuntu Use "%(bucket)s.
s3.amazonaws.com" to the target Amazon S3. "%(bucket)s" and "%(location)s" vars can be
usedif the target S3 system supports dns based buckets.DNS-style bucket+hostname:port
template for accessing a bucket [%(bucket)s.s3.amazonaws.com]:  Encryption password
is used to protect your files from readingby unauthorized persons while in transfer
to S3Encryption password:  Ubuntu-passPath to GPG program [/usr/bin/gpg]:  When using
secure HTTPS protocol all communication with Amazon S3servers is protected from 3rd
party eavesdropping. This method isslower than plain HTTP, and can only be proxied
with Python 2.7 or newerUse HTTPS protocol [Yes]:  No On some networks all internet
access must go through a HTTP proxy.Try setting it here if you can't connect to S3
directlyHTTP Proxy server name:  New settings:Access Key:  fooSecret Key:  barDefault
Region:  USS3 Endpoint:  ubuntuDNS-style bucket+hostname:port template for accessing a
bucket:  %(bucket)s.s3.amazonaws.comEncryption password:  Ubuntu-passPath to GPG program:
/usr/bin/gpgUse HTTPS protocol:  FalseHTTP Proxy server name:HTTP Proxy server port:
0 Test access with supplied credentials? [Y/n] yPlease wait, attempting to list all
buckets...Success. Your access key and secret key worked fine :-) Now verifying that
encryption works...Success. Encryption and decryption worked fine :-) Save settings? [y/
N] yConfiguration saved to '/home/ubuntu/.s3cfg'
```

We have successfully configured `s3cmd`. To see the full configuration, inspect the config file.

```
cat ~/.s3cfg
```

Create a bucket

You have verified that your cluster is accessible via RGW. Now, let's create a bucket using the `s3cmd` tool:

```
s3cmd mb -P s3://mybucket
```

Note

The `-P` flag ensures that the bucket is publicly visible, enabling you to access stored objects easily via a public URL.

`Bucket 's3://mybucket/' created` Our bucket is successfully created.

Upload an image into the bucket

```
s3cmd put -P image.jpg s3://mybucket
```

`upload:  'image.jpg' -> 's3://mybucket/image.jpg' [1 of 1]66565 of 66565 100% in 0s 4.52 MB/s donePublic URL of the object is:  http://ubuntu/mybucket/image.jpg` The output shows that your image is stored in a publicly accessible S3 bucket. You can now click on the public object URL in the output to view the image in your browser.

2.1.8 Cleaning up resources

If you want to remove MicroCeph, you can purge the snap from your machine using:

```
sudo snap remove microceph --purge
```

This command stops all running services and removes the MicroCeph snap, along with your cluster and all its contained resources.

Note

Note: the `--purge` flag will remove all persistent state associated with MicroCeph.

The `--purge` flag deletes all files associated with the MicroCeph package, meaning it will remove the MicroCeph snap without saving any data snapshots. Running the command without this flag will not fully remove MicroCeph; the persistent state will remain intact.

Tip

Skipping the **`purge`** option is useful if you intend to re-install MicroCeph, or move your configuration to a different system.

`2024-11-28T19:44:29+03:00 INFO Waiting for "snap.microceph.rgw.service" to stop. 2024-11-28T19:45:00+03:00 INFO Waiting for "snap.microceph.mds.service" to stop.microceph removed`

2.1.9 Next steps

You have deployed a healthy Ceph cluster on a single-node and enabled RGW on it. Even better, you have consumed the storage in that cluster by creating a bucket and storing an image object in it. Curious to see what else you can do with MicroCeph?

See our *how-to guides*, packed with instructions to help you achieve specific goals with MicroCeph.

Or, explore our *Explanation* and *Reference* sections for additional information and quick references.

2.2 How-to guides

Our *how-to* guides give directions on how perform key operations and processes in MicroCeph.

2.2.1 Installing and initialising MicroCeph cluster

The guides in this section are helpful in the installation and initialisation of both single-node and multi-node clusters.

How to install MicroCeph on a single node

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

This installation 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
```

(continues on next page)

(continued from previous page)

```
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
```

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. You may also provide multiple disks as space separated arguments.

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

Or use the **–all-available** flag to enlist all physical devices available on the machine.

```
sudo microceph disk add --all-available --wipe
```

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
  health: HEALTH_OK
```

(continues on next page)

(continued from previous page)

```
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
```

2.2.2 Configuring your cluster

See these guides for client and network configurations, authentication service integration, and configuration of metrics, alerts and other service instances.

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 1: 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     |
```

(continues on next page)

(continued from previous page)

```
+---+----------------+---------+----------+
| 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     |
+---+----------------+---------+----------+
| 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     |
 +---+----------------+---------+----------+
```

Configure Openstack Keystone Auth in MicroCeph RGW

Ceph Object Gateway (RGW) can be configured to use Openstack Keystone for providing user authentication service. A Keystone authorised user to the gateway will also be automatically created on the Ceph Object Gateway. A token that Keystone validates will be considered as valid by the gateway.

MicroCeph supports setting the following Keystone config keys:

Table 2: Supported Config Keys

Key	Description
rgw_s3_auth_use_keystone	Whether to use keystone auth for the S3 endpoints.
rgw_keystone_url	Keystone server address in {url:port} format
rgw_keystone_admin_token	Keystone admin token (not recommended in production)
rgw_keystone_admin_token_path	Path to Keystone admin token (recommended for production)
rgw_keystone_admin_user	Keystone service tenant user name
rgw_keystone_admin_password	Keystone service tenant user password
rgw_keystone_admin_password_p	Path to Keystone service tenant user password file
rgw_keystone_admin_project	Keystone admin project name
rgw_keystone_admin_domain	Keystone admin domain name
rgw_keystone_service_token_ena	Whether to allow expired tokens with service token in requests
rgw_keystone_service_token_acc	Specify user roles accepted as service roles
rgw_keystone_expired_token_cac	Cache expiration period for an expired token allowed with a service token
rgw_keystone_api_version	Keystone API version
rgw_keystone_accepted_roles	Accepted user roles for Keystone users
rgw_keystone_accepted_admin_r	List of roles allowing user to gain admin privileges
rgw_keystone_token_cache_size	The maximum number of entries in each Keystone token cache
rgw_keystone_verify_ssl	Whether to verify SSL certificates while making token requests to Keystone
rgw_keystone_implicit_tenants	Whether to create new users in their own tenants of the same name
rgw_swift_account_in_url	Whether the Swift account is encoded in the URL path
rgw_swift_versioning_enabled	Enables object versioning
rgw_swift_enforce_content_lengt	Whether content length header is needed when listing containers
rgw_swift_custom_header	Enable swift custom header

A user can set/get/list/reset the above mentioned config keys as follows:

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

```
$ sudo microceph cluster config set rgw_swift_account_in_url true
```

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

```
$ sudo microceph cluster config get rgw_swift_account_in_url
+---+--------------------------+-------+
| # |           KEY            | VALUE |
```

(continues on next page)

(continued from previous page)

```
+---+-------------------------+-------+
| 0 | rgw_swift_account_in_url | true  |
+---+-------------------------+-------+
```

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

```
$ sudo microceph cluster config list
+---+--------------------------+-------+
| # |           KEY            | VALUE |
+---+--------------------------+-------+
| 0 | rgw_swift_account_in_url | true  |
+---+--------------------------+-------+
```

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 rgw_swift_account_in_url
$ sudo microceph cluster config list
+---+-----+-------+
| # | KEY | VALUE |
+---+-----+-------+
```

For detailed documentation of what keys should be configured, visit Ceph Docs

Configuring Cluster network

If you configure a cluster network, OSDs will route heartbeat, object replication and recovery traffic over the cluster network. This may improve performance compared to using a single network.

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

Table 3: Supported Config Keys

Key	Description
cluster_network	Set this key to desired CIDR to configure cluster 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 list
+---+-----------------+-------------+
| # |       KEY       |    VALUE    |
```

(continues on next page)

(continued from previous page)

```
+---+-----------------+-------------+
| 0 | cluster_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 list
+---+-----+-------+
| # | KEY | VALUE |
+---+-----+-------+
```

For more explanations and implementation details refer to *explanation*

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>
```

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

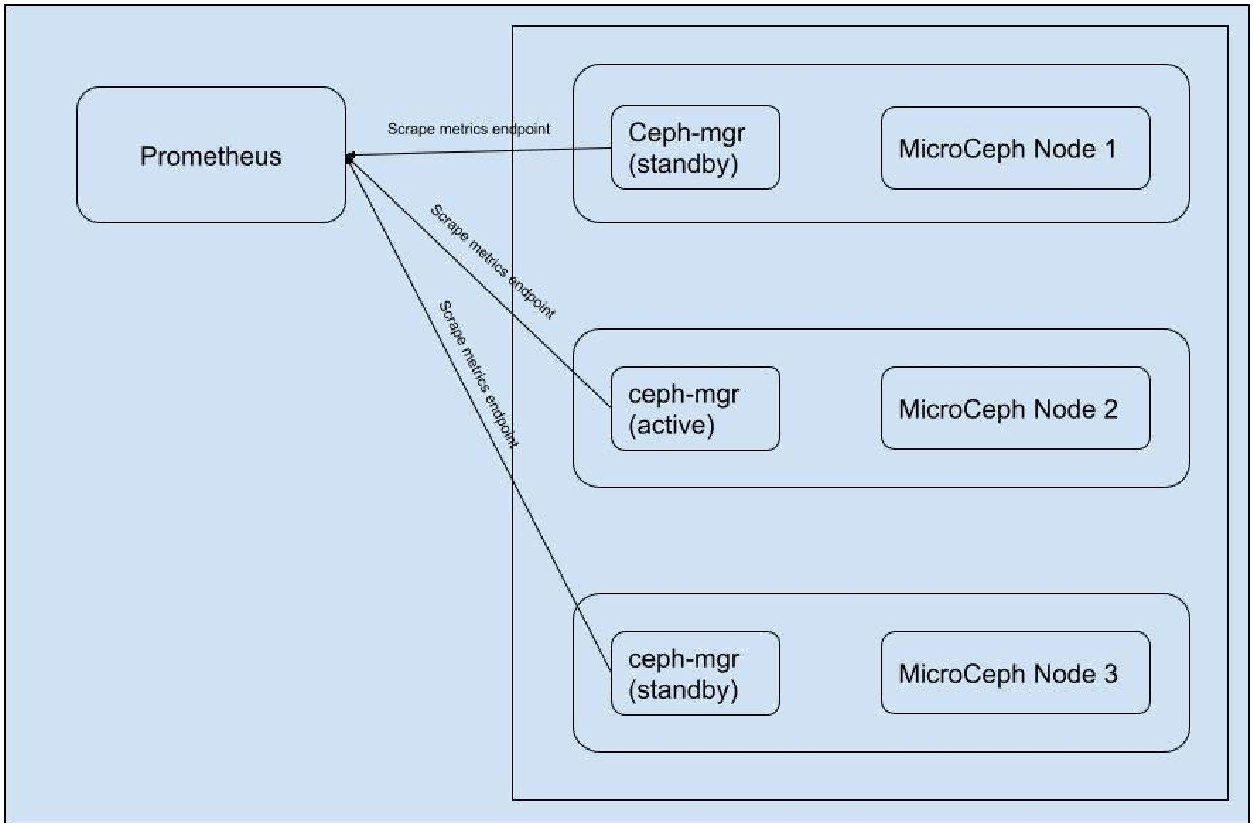


Fig. 1: Prometheus service scraping endpoints of a multi-node MicroCeph cluster.

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

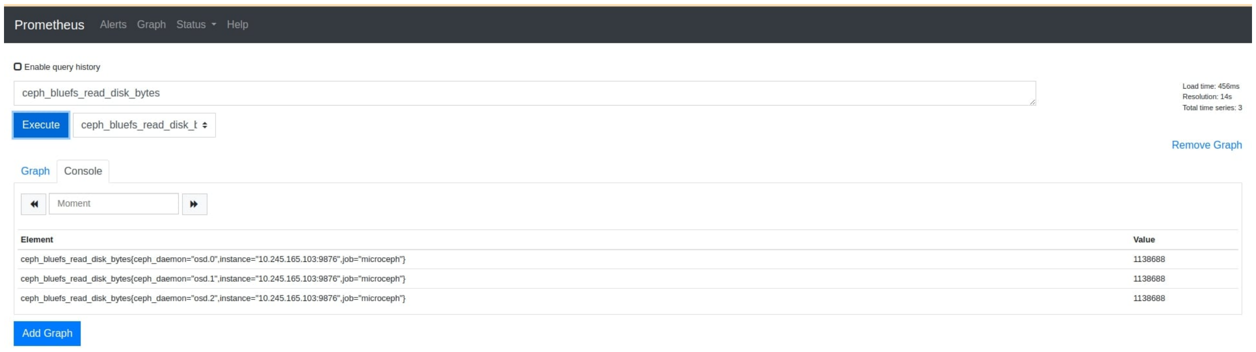
A simple configuration file is provided below:

```yaml
# 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:



ceph_bluefs_read_disk_bytes

Load time: 456ms
Resolution: 14s
Total time series: 3

Element	Value
ceph_bluefs_read_disk_bytes{ceph_daemon="osd.0",instance="10.245.165.103:9876",job="microceph"}	1138688
ceph_bluefs_read_disk_bytes{ceph_daemon="osd.1",instance="10.245.165.103:9876",job="microceph"}	1138688
ceph_bluefs_read_disk_bytes{ceph_daemon="osd.2",instance="10.245.165.103:9876",job="microceph"}	1138688

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

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']

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.

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
```

2.2.3 Interacting with your cluster

Manage your cluster: find steps on how to configure the log level,remove disks, migrate services and more.

Changing the log level

By default, the MicroCeph daemon runs with the log level set to DEBUG. While that is the desirable behaviour for a good number of use cases, there are instances when this level is far too high - for example, embedded devices where storage is much more limited. For these reasons, the MicroCeph daemon exposes a way to both get and set the log level.

Configuring the log level

MicroCeph includes the command `log`, with the sub-commands `set-level` and `get-level`. When setting, we support both string and integer formats for the log level. For example:

```
sudo microceph log set-level warning
sudo microceph log set-level 3
```

Both commands are equivalent. The mapping from integer to string can be consulted by querying the help for the `set-level` sub-command. Note that any changes made to the log level take effect immediately, and need no restarts.

On the other hand, the `get-level` sub-command takes no arguments and returns an integer level only. Any value returned by `get-level` can be used for `set-level`.

For example, after setting the level as shown in the example, we can verify in this way:

```
sudo microceph log get-level
3
```

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.

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:
```

(continues on next page)

(continued from previous page)

```
  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
-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
```

Perform cluster maintenance

Overview

MicroCeph provides a simple and consistent workflow to support maintenance activity.

Before proceeding, please refer to the *Cluster maintenance* to understand its functionality and impact.

Enabling Cluster Maintenance

To review the action plan for enabling maintenance mode, run

```
microceph cluster maintenance enter <node> --dry-run
```

If you only want to verify if the node is ready for maintenance operations, run

```
microceph cluster maintenance enter <node> --check-only
```

By default, noout is set when entering maintenance mode. To disable noout to enable data migration during maintenance, run

```
microceph cluster maintenance enter <node> --set-noout=False
```

By default, OSDs on the node are not stopped during maintenance mode, To stop the OSD service on the node during maintenance, run

```
microceph cluster maintenance enter <node> --stop-osds
```

You can also forcibly bring a node into maintenance mode or ignore the safety checks if you know what you are doing, but it’s generally not recommended as it’s not guaranteed the node is ready for maintenance operations.

```
# Forcibly enter maintenance mode
microceph cluster maintenance enter <node> --force

# Ignore safety checks when entering maintenance mode
microceph cluster maintenance enter <node> --ignore-check
```

Disabling Cluster Maintenance

To review the action plan for disabling maintenance mode, run

```
microceph cluster maintenance exit <node> --dry-run
```

To bring a node out of maintenance, run

```
microceph cluster maintenance exit <node>
```

2.2.4 Managing a remote cluster

Make MicroCeph aware of a remote cluster and configure replication for RBD pools and images.

Import a remote MicroCeph cluster

MicroCeph supports adding secondary MicroCeph clusters as remote clusters. This creates `$remote.conf/$remote.keyring` files in the snap's config directory allowing users (and microceph) to perform ceph operations on the remote clusters.

This also enables capabilities like replication to remote clusters by exposing required remote cluster details to MicroCeph and Ceph.

Working with remote MicroCeph clusters

Assume Primary cluster (named magical) and Secondary cluster (named simple). An operator can generate the cluster token at the secondary cluster as follows:

```
sudo microceph cluster export magical
eyJmc2lkIjoiN2FiZmMwYmItNjIwNC00M2FmLTg4NDQtMjg3NDg2OGNiYTc0Iiwia2V5cmluZy5jbGllbnQubWFnaWNhbCI6IkFRQ0h
```

At the primary cluster, this token can be imported to create the remote record.

```
sudo microceph remote import simple␣
↪eyJmc2lkIjoiN2FiZmMwYmItNjIwNC00M2FmLTg4NDQtMjg3NDg2OGNiYTc0Iiwia2V5cmluZy5jbGllbnQubWFnaWNhbCI6IkFRQ
↪--local-name magical
```

This will create the required $simple.conf and $simple.keyring files. Note: Importing a remote cluster is a unidirectional operation. For symmetric relations both clusters should be added as remotes at each other.

Check remote ceph cluster status

```
 sudo ceph -s --cluster simple --id magical
 cluster:
  id:     7abfc0bb-6204-43af-8844-2874868cba74
  health: HEALTH_OK

services:
  mon: 1 daemons, quorum simple-reindeer (age 18m)
  mgr: simple-reindeer(active, since 18m)
  osd: 3 osds: 3 up (since 17m), 3 in (since 17m)

data:
  pools:   4 pools, 97 pgs
  objects: 4 objects, 449 KiB
```

(continues on next page)

(continued from previous page)

```
  usage:   81 MiB used, 15 GiB / 15 GiB avail
  pgs:     97 active+clean
```

Note: Ceph commands can be invoked on the remote cluster by providing the necessary $cluster and $client.id names.

Similarly, configured remote clusters can be queried as follows

```
sudo microceph remote list
ID  REMOTE NAME  LOCAL NAME
 1  simple       magical
```

and can be removed as

```
sudo microceph remote remove simple
```

Configure RBD replication

MicroCeph supports asynchronously replicating (mirroring) RBD images to a remote cluster.

An operator can enable this on any rbd image, or a whole pool. Enabling it on a pool enables it for all the images in the pool.

Prerequisites

1. A primary and a secondary MicroCeph cluster, for example named "primary_cluster" and "secondary_cluster"
2. primary_cluster has imported configurations from secondary_cluster and vice versa. refer to *import remote*
3. Both clusters have 2 rbd pools: pool_one and pool_two.
4. Both pools at cluster "primary_cluster" have 2 images each (image_one and image_two) while the pools at cluster "secondary_cluster" are empty.

Enable RBD replication

An operator can enable replication for a given rbd pool which is present at both clusters as

```
sudo microceph replication enable rbd pool_one --remote secondary_cluster
```

Here, pool_one is the name of the rbd pool and it is expected to be present at both the clusters.

Check RBD replication status

The above command will enable replication for ALL the images inside pool_one, it can be checked as:

```
sudo microceph replication status rbd pool_one
+------------------------+----------------------+
|         SUMMARY        |        HEALTH        |
+-------------+----------+-------------+--------+
| Name        | pool_one | Replication | OK     |
| Mode        | pool     | Daemon      | OK     |
| Image Count | 2        | Image       | OK     |
+-------------+----------+-------------+--------+


+-------------------+-----------+---------------------------------------+
```

(continues on next page)

(continued from previous page)

```
|    REMOTE NAME    | DIRECTION | UUID                                 |
+-------------------+-----------+--------------------------------------+
| secondary_cluster | rx-tx     | f25af3c3-f405-4159-a5c4-220c01d27507 |
+-------------------+-----------+--------------------------------------+
```

The status shows that there are 2 images in the pool which are enabled for mirroring.

Listing all RBD replication images

An operator can list all the images that have replication (mirroring) enabled as follows:

```
sudo microceph replication list rbd
+-----------+------------+------------+---------------------+
| POOL NAME | IMAGE NAME | IS PRIMARY |  LAST LOCAL UPDATE  |
+-----------+------------+------------+---------------------+
| pool_one  | image_one  |    true    | 2024-10-08 13:54:49 |
| pool_one  | image_two  |    true    | 2024-10-08 13:55:19 |
| pool_two  | image_one  |    true    | 2024-10-08 13:55:12 |
| pool_two  | image_two  |    true    | 2024-10-08 13:55:07 |
+-----------+------------+------------+---------------------+
```

Disabling RBD replication

In some cases, it may be desired to disable replication. A single image ($pool/$image) or a whole pool ($pool) can be disabled in a single command as follows:

Disable Pool replication: .. code-block:: none

> sudo microceph replication disable rbd pool_one sudo microceph replication list rbd +———–+————+————+———————+ | POOL NAME | IMAGE NAME | IS PRIMARY | LAST LOCAL UPDATE | +———–+————+————+———————+ | pool_two | image_one | true | 2024-10-08 13:55:12 | | pool_two | image_two | true | 2024-10-08 13:55:07 | +———–+————+————+———————+

Disable Image replication: .. code-block:: none

> sudo microceph replication disable rbd pool_two/image_two sudo microceph replication list rbd +———–+————+————+———————+ | POOL NAME | IMAGE NAME | IS PRIMARY | LAST LOCAL UPDATE | +———–+————+————+———————+ | pool_two | image_one | true | 2024-10-08 13:55:12 | +———–+————+————+———————+

Perform failover for replicated RBD resources

In case of a disaster, all replicated RBD pools can be failed over to a non-primary remote.

An operator can perform promotion on a non-primary cluster, this will in turn promote all replicated rbd images in all rbd pools and make them primary. This enables them to be consumed by vms and other workloads.

Prerequisites

1. A primary and a secondary MicroCeph cluster, for example named “primary_cluster” and “secondary_cluster”
2. primary_cluster has imported configurations from secondary_cluster and vice versa. refer to *import remote*
3. RBD replication is configured for at least 1 rbd image. refer to *configure rbd replication*

Failover to a non-primary remote cluster

List all the resources on ‘secondary_cluster’ to check primary status.

```
sudo microceph replication list rbd
+-----------+------------+------------+---------------------+
| POOL NAME | IMAGE NAME | IS PRIMARY | LAST LOCAL UPDATE   |
+-----------+------------+------------+---------------------+
| pool_one  | image_one  | false      | 2024-10-14 09:03:17 |
| pool_one  | image_two  | false      | 2024-10-14 09:03:17 |
+-----------+------------+------------+---------------------+
```

An operator can perform cluster wide promotion as follows:

```
sudo microceph replication promote --remote primary_cluster --yes-i-really-mean-it
```

Here, <remote> parameter helps microceph filter the resources to promote. Since promotion of secondary_cluster may cause a split-brain condition in future, it is necessary to pass –yes-i-really-mean-it flag.

Verify RBD replication primary status

List all the resources on ‘secondary_cluster’ again to check primary status.

```
sudo microceph replication status rbd pool_one
+-----------+------------+------------+---------------------+
| POOL NAME | IMAGE NAME | IS PRIMARY | LAST LOCAL UPDATE   |
+-----------+------------+------------+---------------------+
| pool_one  | image_one  | true       | 2024-10-14 09:06:12 |
| pool_one  | image_two  | true       | 2024-10-14 09:06:12 |
+-----------+------------+------------+---------------------+
```

The status shows that there are 2 replicated images and both of them are now primary.

Failback to old primary

Once the disaster struck cluster (primary_cluster) is back online the RBD resources can be failed back to it, but, by this time the RBD images at the current primary (secondary_cluster) would have diverged from primary_cluster. Thus, to have a clean sync, the operator must decide which cluster would be demoted to the non-primary status. This cluster will then receive the RBD mirror updates from the standing primary.

Note: Demotion can cause data loss and hence can only be performed with the ‘yes-i-really-mean-it’ flag.

At primary_cluster (was primary before disaster), perform demotion. .. code-block:: none

> sudo microceph replication demote –remote secondary_cluster failed to process demote_replication request for rbd: demotion may cause data loss on this cluster. If you understand the *RISK* and you’re *ABSOLUTELY CERTAIN* that is what you want, pass –yes-i-really-mean-it.

Now, again at the ‘primary_cluster’, perform demotion with –yes-i-really-mean-it flag. .. code-block:: none

> sudo microceph replication demote –remote secondary_cluster –yes-i-really-mean-it

Note: MicroCeph with demote the primary pools and will issue a resync for all the mirroring images, hence it may cause data loss at the old primary cluster.

2.2.5 Upgrading your cluster

Follow these steps carefully to perform a major upgrade.

Major Upgrades

Overview

This guide provides step-by-step instructions on how to upgrade your MicroCeph cluster to a new major release.

Follow these steps carefully to prevent to ensure a smooth transition.

In the code examples below an upgrade to the Squid stable release is shown. The procedure should apply to any major release upgrade in a similar way however.

Procedure

Prerequisites

Firstly, before initiating the upgrade, ensure that the cluster is healthy. Use the below command to check the cluster health:

```
sudo ceph -s
```

Note: Do not start the upgrade if the cluster is unhealthy.

Secondly, review the *release notes* to check for any version-specific information.

Optional but Recommended: Preparation Steps

Carry out these precautionary steps before initiating the upgrade:

1. **Back up your data**: as a general precaution, it is recommended to take a backup of your data (such as stored S3 objects, RBD volumes, or cephfs filesystems).
2. **Prevent OSDs from dropping out of the cluster**: Run the following command to avoid OSDs from unintentionally dropping out of the cluster during the upgrade process:

```
sudo ceph osd set noout
```

Upgrading Each Cluster Node

If your cluster is healthy, proceed with the upgrade by refreshing the snap on each node using the following command:

```
sudo snap refresh microceph --channel squid/stable
```

Be sure to perform the refresh on every node in the cluster.

Verifying the Upgrade

Once the upgrade process is done, verify that all components have been upgraded correctly. Use the following command to check:

```
sudo ceph versions
```

Unsetting Noout

If you had previously set noout, unset it with this command:

```
sudo ceph osd unset noout
```

You have now successfully upgraded your Ceph cluster.

2.2.6 Consuming cluster storage

Follow these guides to learn how to make use of the storage provided by your cluster.

Mount MicroCeph backed Block Devices

Ceph RBD (RADOS Block Device) are virtual block devices backed by the Ceph storage cluster. This tutorial will guide you with mounting Block devices using MicroCeph.

The above will be achieved by creating an rbd image on the MicroCeph deployed Ceph cluster, mapping it on the client machine, and then mounting it.

> **Warning**
>
> MicroCeph as an isolated snap cannot perform certain elevated operations like mapping the rbd image to the host. Therefore, it is recommended to use the client tools as described in this documentation, even if the client machine is the MicroCeph node itself.

MicroCeph Operations:

Check Ceph cluster's status:

```
$ sudo ceph -s
cluster:
    id:     90457806-a798-47f2-aca1-a8a93739941a
    health: HEALTH_OK

services:
    mon: 1 daemons, quorum workbook (age 36m)
    mgr: workbook(active, since 50m)
    osd: 3 osds: 3 up (since 17m), 3 in (since 47m)

data:
    pools:   2 pools, 33 pgs
    objects: 21 objects, 13 MiB
    usage:   94 MiB used, 12 GiB / 12 GiB avail
    pgs:     33 active+clean
```

Create a pool for RBD images:

```
$ sudo ceph osd pool create block_pool
pool 'block_pool' created

$ sudo ceph osd lspools
1 .mgr
2 block_pool
```

(continues on next page)

(continued from previous page)

```
$ sudo rbd pool init block_pool
```

Create RBD image:

```
$ sudo rbd create bd_foo --size 8192 --image-feature layering -p block_pool
$ sudo rbd list -p block_pool
bd_foo
```

Client Operations:

Download 'ceph-common' package:

```
$ sudo apt install ceph-common
```

This step is required even if the client machine is a MicroCeph node itself.

Fetch the `ceph.conf` and `ceph.keyring` file :

Ideally, a keyring file for any CephX user which has access to RBD devices will work. For the sake of simplicity, we are using admin keys in this example.

```
$ cat /var/snap/microceph/current/conf/ceph.conf
# # Generated by MicroCeph, DO NOT EDIT.
[global]
run dir = /var/snap/microceph/1039/run
fsid = 90457806-a798-47f2-aca1-a8a93739941a
mon host = 192.168.X.Y
public_network = 192.168.X.Y/24
auth allow insecure global id reclaim = false
ms bind ipv4 = true
ms bind ipv6 = false

$ cat /var/snap/microceph/current/conf/ceph.keyring
# Generated by MicroCeph, DO NOT EDIT.
[client.admin]
    key = AQCNTXlmohDfDRAAe3epjquyZGrKATDhL8p3og==
```

The files are located at the paths shown above on any MicroCeph node. Moving forward, we will assume that these files are located at mentioned path.

Map the RBD image on client:

```
$ sudo rbd map \
    --image bd_foo \
    --name client.admin \
    -m 192.168.29.152 \
    -k /var/snap/microceph/current/conf/ceph.keyring \
    -c /var/snap/microceph/current/conf/ceph.conf \
    -p block_pool \
    /dev/rbd0

$ sudo mkfs.ext4 -m0 /dev/rbd0
mke2fs 1.46.5 (30-Dec-2021)
```

(continues on next page)

(continued from previous page)

```
Discarding device blocks: done
Creating filesystem with 2097152 4k blocks and 524288 inodes
Filesystem UUID: 1deeef7b-ceaf-4882-a07a-07a28b5b2590
Superblock backups stored on blocks:
    32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632

Allocating group tables: done
Writing inode tables: done
Creating journal (16384 blocks): done
Writing superblocks and filesystem accounting information: done
```

Mount the device on a suitable path:

```
$ sudo mkdir /mnt/new-mount
$ sudo mount /dev/rbd0 /mnt/new-mount
$ cd /mnt/new-mount
```

With this, you now have a block device mounted at `/mnt/new-mount` on your client machine that you can perform IO to.

Perform IO and observe the ceph cluster:

Write a file on the mounted device:

```
$ sudo dd if=/dev/zero of=random.img count=1 bs=10M
...
10485760 bytes (10 MB, 10 MiB) copied, 0.0176554 s, 594 MB/s

$ ll
...
-rw-r--r-- 1 root root 10485760 Jun 24 17:02 random.img
```

Ceph cluster state post IO:

```
$ sudo ceph -s
cluster:
    id:     90457806-a798-47f2-aca1-a8a93739941a
    health: HEALTH_OK

services:
    mon: 1 daemons, quorum workbook (age 37m)
    mgr: workbook(active, since 51m)
    osd: 3 osds: 3 up (since 17m), 3 in (since 48m)

data:
    pools:   2 pools, 33 pgs
    objects: 24 objects, 23 MiB
    usage:   124 MiB used, 12 GiB / 12 GiB avail
    pgs:     33 active+clean
```

Comparing the ceph status output before and after writing the file shows that the MicroCeph cluster has grown by 30MiB which is thrice the size of the file we wrote (10MiB). This is because MicroCeph configures 3 way replication by default.

Mount MicroCeph backed CephFs shares

CephFs (Ceph Filesystem) are filesystem shares backed by the Ceph storage cluster. This tutorial will guide you with mounting CephFs shares using MicroCeph.

The above will be achieved by creating an fs on the MicroCeph deployed Ceph cluster, and then mounting it using the kernel driver.

MicroCeph Operations:

Check Ceph cluster's status:

```
$ sudo ceph -s
cluster:
    id:     90457806-a798-47f2-aca1-a8a93739941a
    health: HEALTH_OK

services:
    mon: 1 daemons, quorum workbook (age 6h)
    mgr: workbook(active, since 6h)
    osd: 3 osds: 3 up (since 6h), 3 in (since 23h)

data:
    pools:   4 pools, 97 pgs
    objects: 46 objects, 23 MiB
    usage:   137 MiB used, 12 GiB / 12 GiB avail
    pgs:     97 active+clean
```

Create data/metadata pools for CephFs:

```
$ sudo ceph osd pool create cephfs_meta
$ sudo ceph osd pool create cephfs_data
```

Create CephFs share:

```
$ sudo ceph fs new newFs cephfs_meta cephfs_data
new fs with metadata pool 4 and data pool 3
$ sudo ceph fs ls
name: newFs, metadata pool: cephfs_meta, data pools: [cephfs_data ]
```

Client Operations:

Download 'ceph-common' package:

```
$ sudo apt install ceph-common
```

This step is required for `mount.ceph` i.e. making mount aware of ceph device type.

Fetch the `ceph.conf` and `ceph.keyring` file :

Ideally, a keyring file for any CephX user which has access to CephFs will work. For the sake of simplicity, we are using admin keys in this example.

```
$ pwd
/var/snap/microceph/current/conf
```

(continues on next page)

(continued from previous page)

```
$ ls
ceph.client.admin.keyring  ceph.conf  ceph.keyring  metadata.yaml
```

The files are located at the paths shown above on any MicroCeph node. The kernel driver, by-default looks into `/etc/ceph` so we will create symbolic links to that folder.

```
$ sudo ln -s /var/snap/microceph/current/conf/ceph.keyring /etc/ceph/ceph.keyring
$ sudo ln -s /var/snap/microceph/current/conf/ceph.conf /etc/ceph/ceph.conf
$ ll /etc/ceph/
...
lrwxrwxrwx   1 root root    42 Jun 25 16:28 ceph.conf -> /var/snap/microceph/current/
↪conf/ceph.conf
lrwxrwxrwx   1 root root    45 Jun 25 16:28 ceph.keyring -> /var/snap/microceph/current/
↪conf/ceph.keyring
```

Mount the filesystem:

```
$ sudo mkdir /mnt/mycephfs
$ sudo mount -t ceph :/ /mnt/mycephfs/ -o name=admin,fs=newFs
```

Here, we provide the CephX user (admin in our example) and the fs created earlier (newFs).

With this, you now have a CephFs mounted at `/mnt/mycephfs` on your client machine that you can perform IO to.

Perform IO and observe the ceph cluster:

Write a file:

```
$ cd /mnt/mycephfs
$ sudo dd if=/dev/zero of=random.img count=1 bs=50M
52428800 bytes (52 MB, 50 MiB) copied, 0.0491968 s, 1.1 GB/s

$ ll
...
-rw-r--r-- 1 root root 52428800 Jun 25 16:04 random.img
```

Ceph cluster state post IO:

```
$ sudo ceph -s
cluster:
    id:     90457806-a798-47f2-aca1-a8a93739941a
    health: HEALTH_OK

services:
    mon: 1 daemons, quorum workbook (age 8h)
    mgr: workbook(active, since 8h)
    mds: 1/1 daemons up
    osd: 3 osds: 3 up (since 8h), 3 in (since 25h)

data:
    volumes: 1/1 healthy
    pools:   4 pools, 97 pgs
    objects: 59 objects, 73 MiB
```

(continues on next page)

(continued from previous page)

```
    usage:   287 MiB used, 12 GiB / 12 GiB avail
    pgs:     97 active+clean
```

We observe that the cluster usage grew by 150 MiB which is thrice the size of the file written to the mounted share. This is because MicroCeph configures 3 way replication by default.

2.3 Reference

Our Reference section provides technical details about MicroCeph, such as reference information about the command line interface and notes on major MicroCeph releases.

2.3.1 CLI Commands

MicroCeph has a command line interface that can be used to manage a client and the cluster, as well as query the status of any current deployment. Each command is documented separately, or use the help argument from the command line to learn more about the commands while working with MicroCeph, with `microceph help`.

MicroCeph CLI Commands

Use these commands to initialise, deploy and manage your MicroCeph cluster.

`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
export      Generates cluster token for given Remote cluster
join        Joins an existing cluster
list        List servers in the cluster
maintenance Enter or exit the maintenance mode.
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:

```
--microceph-ip    string Network address microceph daemon binds to.
--mon-ip          string Public address for bootstrapping ceph mon service.
--public-network  string Public network Ceph daemons bind to.
--cluster-network string Cluster network Ceph daemons bind to.
```

`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.
--skip-restart   Don't perform the daemon restart for current config.
```

`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.
--skip-restart   Don't perform the daemon restart for current config.
```

`export`

Generates cluster token for Remote cluster with given name.

Usage:

```
microceph cluster export <remote-name> [flags]
```

Flags:

```
--json   output as json string
```

`join`

Joins an existing cluster.

Usage:

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

Flags:

```
--microceph-ip    string Network address microceph daemon binds to.
```

`list`

Lists servers in the cluster.

Usage:

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

`maintenance`

Enter or exit the maintenance mode.

Usage:

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

Available Commands:

```
enter       Enter maintenance mode.
exit        Exit maintenance mode.
```

maintenance enter

Enter maintenance mode.

Usage:

```
microceph cluster maintenance enter <NODE_NAME> [flags]
```

Flags:

```
--check-only     Only run the preflight checks (mutually exclusive with --ignore-check).
--dry-run        Dry run the command.
--force          Force to enter maintenance mode.
--ignore-check   Ignore the the preflight checks (mutually exclusive with --check-only).
--set-noout      Stop CRUSH from rebalancing the cluster. (default true)
--stop-osds      Stop the OSDS when entering maintenance mode.
```

maintenance exit

Exit maintenance mode.

Usage:

```
microceph cluster maintenance exit <NODE_NAME> [flags]
```

Flags:

```
--check-only     Only run the preflight checks (mutually exclusive with --ignore-check).
--dry-run        Dry run the command.
--ignore-check   Ignore the the preflight checks (mutually exclusive with --check-only).
```

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 one or more new Ceph disks (OSDs) to the cluster, alongside optional devices for write-ahead logging and database management. The command takes arguments which is either one or more paths to block devices such as /dev/sdb, or a specification for loop files.

For block devices, add a space separated list of paths, e.g. "/dev/sda /dev/sdb …". You may also add WAL and DB devices, but doing this is mutually exclusive with adding more than one OSD block device at a time.

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 OSDs, 8GB each.

Note that loop files can't be used with encryption nor WAL/DB devices.

Usage:

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

Flags:

```
--all-available       add all available devices as OSDs
--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>] [--ssl-port <port>] [--ssl-certificate <certificate␣
↪material>] [--ssl-private-key <private key material>] [--target <server>] [--wait
↪<bool>] [flags]
```

Flags:

```
--port int                Service non-SSL port (default: 80) (default 80)
--ssl-port int            Service SSL port (default: 443) (default 443)
--ssl-certificate string  base64 encoded SSL certificate
--ssl-private-key string  base64 encoded SSL private key
--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
-v, --verbose     Show all information messages
    --version     Print version number
```

pool

Manages pools in MicroCeph.

Usage:

```
microceph pool [command]
```

Available commands:

```
set-rf      Set the replication factor for pools
```

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
```

set-rf

Sets the replication factor for one or more pools in the cluster. The command takes two arguments: The pool specification (a string) and the replication factor (an integer).

The pool specification can take one of three forms: Either a list of pools, separated by a space, in which case the replication factor is applied only to those pools (provided they exist). It can also be an asterisk ('*') in which case the process is applied to all existing pools; or an empty string (''), which sets the default pool size, but doesn't change any existing pools.

Usage:

```
microceph pool set-rf <pool-spec> <replication-factor>
```

remote

Manage MicroCeph remotes.

Usage:

```
microceph remote [flags]
microceph remote [command]
```

Available commands:

```
import      Import external MicroCeph cluster as a remote
list        List all configured remotes for the site
remove      Remove configured remote
```

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
```

import

Import external MicroCeph cluster as a remote

Usage:

```
microceph remote import <name> <token> [flags]
```

Flags:

```
--local-name string   friendly local name for cluster
```

list

List all configured remotes for the site

Usage:

```
microceph remote list [flags]
```

Flags:

```
--json   output as json string
```

remove

Remove configured remote

Usage:

```
microceph remote remove <name> [flags]
```

replication

Usage:

```
microceph replication [command]
```

Available commands:

```
configure   Configure replication parameters for RBD resource (Pool or Image)
disable     Disable replication for RBD resource (Pool or Image)
enable      Enable replication for RBD resource (Pool or Image)
list        List all configured replications.
status      Show RBD resource (Pool or Image) replication status
```

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
```

`enable`

Enable replication for RBD resource (Pool or Image)

Usage:

```
microceph replication enable rbd <resource> [flags]
```

Flags:

```
--remote string      remote MicroCeph cluster name
--schedule string    snapshot schedule in days, hours, or minutes using d, h, m suffix␣
↪respectively
--skip-auto-enable   do not auto enable rbd mirroring for all images in the pool.
--type string        'journal' or 'snapshot', defaults to journal (default "journal")
```

`status`

Show RBD resource (Pool or Image) replication status

Usage:

```
microceph replication status rbd <resource> [flags]
```

Flags:

```
--json   output as json string
```

`list`

List all configured remotes replication pairs.

Usage:

```
microceph replication list rbd [flags]
```

```
--json          output as json string
--pool string   RBD pool name
```

disable

Disable replication for RBD resource (Pool or Image)

Usage:

```
microceph replication disable rbd <resource> [flags]
```

```
--force   forcefully disable replication for rbd resource
```

promote

Promote local cluster to primary

```
microceph replication promote [flags]
```

```
--remote         remote MicroCeph cluster name
--force          forcefully promote site to primary
```

demote

Demote local cluster to secondary

Usage:

```
microceph replication demote [flags]
```

```
--remote         remote MicroCeph cluster name
```

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
```

2.3.2 Release Notes

The release notes section provides details on major MicroCeph releases.

Overview

The following provides details on major MicroCeph releases, beginning with the MicroCeph squid release.

MicroCeph Squid

The Ceph team is happy to announce the release of MicroCeph 19.2.0 (squid). This is the first stable release in the Squid series of releases.

The MicroCeph squid release can be installed from the squid/stable track.

Highlights

- Uses Ceph 19.2.0 (squid)
- Support for RBD remote replication
- OSD support for many additional block device types such as NVMe, partitions, LVM volumes
- Improved ipv6 support
- Updated dependencies, based off of Ubuntu 24.04
- Various fixes and documentation improvements

Important Changes

For added security, MicroCeph now checks hostnames upon cluster joining. This means that the name used when running *microceph cluster add <name>* must match the hostname of the node where *microceph cluster join* is being run. If the hostname does not match joining the node will fail, and log a message *Joining server certificate SAN does not contain join token name* to syslog.

Known Issues

iSCSI users are advised that the upstream developers of Ceph encountered a bug during an upgrade from Ceph 19.1.1 to Ceph 19.2.0. Read Tracker Issue 68215 before attempting an upgrade to 19.2.0.

List of pull requests

- #467: Fix: increase timings for osd release
- #466: Adjust ‘verify_health’ iterations
- #464: Test: upgrade update
- #463: Fix: add python3-packaging
- #462: Test: upgrade reef to local build
- #461: Test: add reef to squid upgrade test
- #460: Improve require-osd-release
- #459: Set the ‘require-osd-release’ option on startup
- #458: Updated readme.md
- #457: Modify post-refresh hook to set OSD-release
- #456: Make remote replication CLI conformant to CLI guidelines
- #454: Pin LXD and use microcluster with dqlite LTS
- #447: Update mods, build from noble

- #443: Bootstrap: wait for daemon
- #441: Build from noble-proposed
- #440: Remove tutorial section
- #438: MicroCeph Remote Replication (3/3): Site Failover/Failback
- #437: MicroCeph Remote Replication (2/3): RBD Mirroring
- #433: Docs: fix indexes
- #432: Use square brackets around IPv6 in ceph.conf
- #430: Adds support for RO cluster configs
- #429: Move mounting CephFS shares tutorial to how-to section
- #428: Move mounting RBD tutorial to how-to section
- #427: Move multi-node tutorial to how-to section
- #426: Move multi-node tutorial to how-to section
- #422: Change tutorial landing page
- #419: Change explanation landing page
- #418: Add CephFS to wordlist
- #417: Move MicroCeph charm to explanation section
- #416: Fix reference landing page
- #415: Move single-node tutorial to how-to section
- #409: Fetch current OSD pool configuration over the API
- #407: Add interfaces: rbd kernel module and support
- #405: MicroCeph Remote Replication (1/3): Remote Awareness
- #401: doc: remove `woke-install` as prereq for building the docs
- #400: doc: remove `woke-install` as prereq for building the docs
- #398: MicroCeph Remote Replication (2/3): RBD Mirroring
- #395: Use LTS microcluster

2.4 Explanation

The explanatory and conceptual guides in this section provide a better understanding of MicroCeph. They enable you to expand your knowledge and become better at configuring, encrypting, managing, deploying and backing up your workloads.

2.4.1 Working with MicroCeph

Understand the steps to take to successfully deploy and manage your Ceph clusters quickly.

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.

Maintenance Mode

Overview

Cluster maintenance is important for keeping the Ceph Storage Cluster at a healthy state.

MicroCeph provides a simple and consistent workflow to support maintenance activity. Before executing any high-risk maintenance operations on a node, operators are strongly recommended to enable maintenance mode to minimise the impact and ensure system stability. For more information on how to enable maintenance mode in MicroCeph, please refer to *Perform cluster maintenance*.

Strategy

Bringing a node into and out of maintenance mode generally follows check-and-apply pattern. We first verify if the node is ready for maintenance operations, then run the steps to bring the node into or out of maintenance mode if the verification passes. The strategy is idempotent, you can repeatedly run the steps without any issue.

The strategy is defined as follows:

Enabling maintenance mode

- Check if OSDs on the node are `ok-to-stop` to ensure sufficient redundancy to tolerate the loss of OSDs on the node.
- Check if the number of running services is greater than the minimum (3 MON, 1 MDS, 1 MGR) required for quorum.
- *(Optional)* Apply noout flag to prevent data migration from triggering during the planned maintenance slot. (default=True)
- *(Optional)* Bring the OSDs down and disable the service (Default=False)

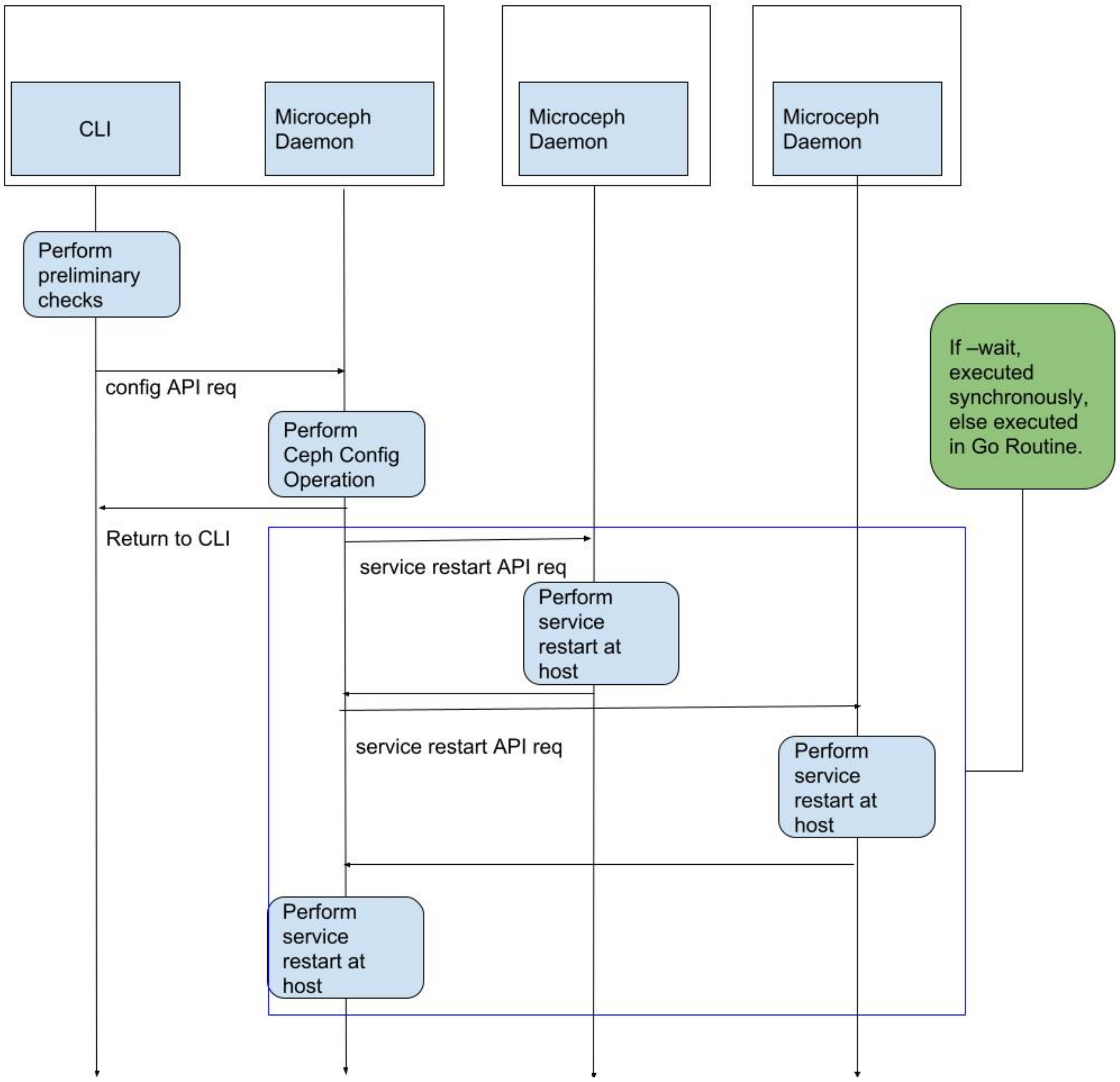

CLI
Microceph Daemon
Microceph Daemon
Microceph Daemon
Perform preliminary checks
config API req
Perform Ceph Config Operation
Return to CLI
service restart API req
Perform service restart at host
service restart API req
Perform service restart at host
Perform service restart at host
If –wait, executed synchronously, else executed in Go Routine.

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

Disabling maintenance mode

- Remove noout flag to allow data migration from triggering after the planned maintenance slot.
- Bring the OSDs up and enable the service

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.

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).

Taking Backups for your Workload

The MicroCeph deployed Ceph cluster supports snapshot based backups for Block and File based workloads.

This document is an index of upstream documentation available for snapshots along with some bridging commentary to help understand it better.

RBD Snapshots:

Ceph supports creating point in time read-only logical copies. This allows an operator to create a checkpoint for their workload backup. The snapshots can be exported for external backup or kept in Ceph for rollback to older version.

Pre-requisites

Refer to *How to mount MicroCeph Block Devices* for getting started with RBD.

Once you have a the block device mounted and in use, you can jump to Ceph RBD Snapshots

CephFs Snapshots:

Similar to RBD snapshots, CephFs snapshots are read-only logical copies of **any chosen sub-directory** of the corresponding filesystem.

Pre-requisites

Refer to *How to mount MicroCeph CephFs shares* for getting started with CephFs.

Once you have a the filesystem mounted and in use, you can jump to CephFs Snapshots

2.4.2 The Snap content interface

Access MicroCeph's configuration and credentials.

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.

2.4.3 The MicroCeph charm

The MicroCeph charm helps you deploy and manage your MicroCeph deployment with Juju.

The MicroCeph charm

The MicroCeph charm is used to incorporate MicroCeph into Juju-managed deployments. It offers an alternative method for deploying and managing MicroCeph. In effect, the charm installs the `microceph` snap. As expected, it provides MicroCeph management via standard Juju commands (e.g. `juju config` and `juju run`).

For more information, see the microceph entry on the Charmhub.

2.4.4 Security in MicroCeph

Learn about security approaches in MicroCeph, e.g. enabling support for MicroCeph's automatic full disk encryption on OSDs and cryptographic technology used in MicroCeph.

Security overview

Cryptographic approaches in MicroCeph

MicroCeph is a Ceph cluster in a single snap package. The snap is built on top of Ubuntu Ceph packages, and, therefore, shares some of their security features. This section makes references to Cryptography in Ubuntu Ceph packages.

Snap security features

MicroCeph is distributed as a snap package. Snaps offer some inherent security features, including:

- **Sandboxing and confinement**: snaps are containerized applications that run in a sandboxed environment. This isolation is achieved using Linux kernel security features such as `AppArmor`, `Seccomp`, and `cgroups`. These tools limit the system resources that snaps can access, preventing unauthorized access to the host system.
- **Confinement level**: MicroCeph uses strict confinement. This level provides the highest security by restricting access to system resources unless explicitly allowed through interfaces.
- **Interfaces and plugs**: snaps use interfaces to request specific permissions for accessing system resources. This allows for granular access to underlying hosts resources, i.e. only the resources a specific application requires need to be allowed. Specific resources that can be used by MicroCeph are discussed in the next section.
- **Updates**: snaps offer an easy network-based update mechanism. By default, snaps automatically upgrade; however a best practice for MicroCeph production deployments is to hold automatic upgrades so that operators can make use of zero-downtime upgrades for multi-node clusters.
- **Cryptographic signatures**: the snap store uses cryptographic signatures to verify the integrity and authenticity of snaps. This ensures that users download genuine software that has not been tampered with.
- **Secure distribution**: The snap store acts as a central repository where snaps are published and distributed securely. The store implements both automated checks and manual reviews to ensure the quality and security of the software it hosts.

Resources used by MicroCeph snaps

Interfaces

MicroCeph snaps can use the following snap interfaces:

- `kernel-module-load`: to enable loading of Ceph-specific kernel modules, such as the `rbd` module.
- `ceph-conf`: to expose Ceph configuration to cooperating snaps.
- `ceph-logs`: to allow access to log files.

Plugs

MicroCeph snaps can use the following snap plugs:

- `block-devices`: for storage
- `dm-crypt`: for disk encryption
- `hardware-observe`: for block device detection
- `home`: provides access to user-supplied configuration and certificates
- `microceph-support`: for additional storage
- `mount-observe`: for storage management
- `network`: networking client
- `network-bind`: network servers
- `process-control`:to support resource limits configuration

Microcluster security

MicroCeph is built on the Canonical Microcluster library. Microcluster manages cluster topology and cluster membership. Adding nodes is regulated by issuing tokens which new nodes can use to trigger a request to join the cluster. After successfully joining the cluster, Microcluster issues TLS certificates to new nodes. `TLS1.3`, by default, is the minimum supported version. This can be overridden by setting an environment variable to set the minimum supported version to `TLS1.2`.

Cluster members use the newly created certificates to authenticate for cluster API access.

Ceph authentication and authorization

Internally, MicroCeph deploys a Ceph cluster which uses the `cephx` protocol for authentication and authorization. See Cryptography in Ubuntu Ceph for more details.

Data at rest

MicroCeph offers full disk encryption (FDE) for Object Storage Daemons (OSDs), activated when adding disks. Note that this disk encryption only pertains to user data managed in Ceph, and not encryption of the cluster data (such as administrative data, logs, etc.). To use FDE, users must first connect the `dm-crypt` plug for MicroCeph (it is not auto-connected). When FDE is requested, MicroCeph generates a random key and stores it in the Ceph Monitor (MON). This key is then used to setup Linux Unified Key Setup (LUKS) via `cryptsetup`, using `cipher AES-XTS-plain64` and `SHA256` hashing, with a 256-bit keysize.

Note

While the FDE approach for OSD encryption shares some of the techniques employed by the data at rest encryption features in Ubuntu Ceph, it's a separate implementation due to the specific sandboxing needs of the MicroCeph snap.

Storage types

Like Ceph, MicroCeph provides three types of storage to clients, i.e. object, block and file storage, to clients. Each type of storage supports specific security features.

RGW object storage

Accessing Ceph object storage happens via the RADOS Gateway (RGW) service. This service supports transport security via SSL/TLS for encrypting client traffic. To do this, it will need to be configured with certificate material for SSL/TLS. In MicroCeph, SSL/TLS certificates can be provided when enabling the RGW service.

Server-side encryption

The RGW service supports server-side encryption (SSE) according to the Amazon SSE specifications. MicroCeph does not offer key management for RGW; therefore, only the customer key mechanism is supported. This is done via the Amazon `SSE-C` specification, which uses `AES256` symmetric encryption. RGW implements this as `AES256-CBC`. Moreover, as per the SSE-C specification, keys may be provided as `128-bit MD5 digest`.

RBD block storage

Like in Ubuntu Ceph, the RADOS Block Device (RBD) component can provide block devices backed by MicroCeph storage. Access to RBD is regulated using the native `cephx` protocol for authentication and authorization.

RBD encryption

Users can instruct Ceph to encrypt block device images utilizing the `rbd` encryption format commands. RBD supports the `AES128` and `AES256` algorithms, with `AES256 XTS-plain64` being the default.

CephFS file storage

CephFS provides filesystem storage to clients in MicroCeph, similarly to how Ubuntu Ceph provides filesystem storage. Client access is regulated using the Ceph-native `cephx` protocol, which performs authentication and authorization for clients. Ceph supports access control to specific filesystem and filesystem subtrees.

Dashboard

The MicroCeph dashboard provides basic administrative capabilities. Access to the dashboard can be secured via SSL/TLS. The dashboard module also exposes an API, the Ceph RESTful API. Like regular dashboard access, this can be secured through SSL/TLS. The RESTful API can make use of JSON Web Tokens (JWTs) using the `HMAC-SHA256` algorithm.

Summary of cryptographic components

In summary, the cryptographic libraries and tools used in MicroCeph are:

- `dm-crypt`
- LUKS
- OpenSSL

Full disk encryption

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.

Operating a MicroCeph instance involves managing Ceph storage components contained within a snap package, orchestrated by the microceph daemon (microcephd). Ensuring the security of this system is necessary to protect data integrity and confidentiality. This guide provides an overview of security aspects, potential attack vectors, and some best practices for deploying and operating MicroCeph in a secure manner.

Architectural overview

Understanding the MicroCeph architecture is the first step towards securing it. MicroCeph packages core Ceph daemons (MON, MGR, OSD, and optionally RGW, MDS) into a single snap. These daemons are managed by the microcephd service, which uses a distributed dqlite database for configuration and state. Management is primarily done via the microceph command-line tool interacting with microcephd, alongside standard snapd services.

Components

- Host System: The underlying Linux operating system where the MicroCeph snap is installed.
- MicroCeph Snap: The package containing Ceph daemons, microcephd, and management logic. It runs with confinement provided by snapd. Also see the Snap security documentation for details.
- microcephd: The core service (based on Microcluster) responsible for managing the MicroCeph cluster state, coordinating actions across nodes (if clustered), and managing the Ceph daemons within the snap.
- dqlite Database: A distributed SQLite database used by microcephd to store cluster configuration, node status, and other metadata.
- microceph CLI: The primary tool used by administrators to interact with microcephd for managing MicroCeph instances.
- Ceph Daemons (within the snap):
  - ceph-mon: Ceph Monitor (MON) daemon(s).
  - ceph-mgr: Ceph Manager (MGR) daemon(s), providing access to management APIs and modules like the Dashboard.
  - ceph-osd: Ceph Object Storage Daemons (OSDs), managing data on underlying storage devices.
  - ceph-radosgw (optional): RGW (object-storage S3/Swift gateway) service.
  - ceph-mds (optional): Metadata Server (MDS) daemons for CephFS.
- Client Workloads: Consume Ceph storage via RBD block devices, RGW object buckets, or CephFS shared filesystems.

Attack surface

The attack surface encompasses all points where an unauthorized user could attempt to enter or extract data from the system. For MicroCeph, these include:

Open ports and network interfaces

Ceph daemons and potentially microcephd listen on TCP ports. Use host-level firewalls (like ufw, firewalld, or nftables) to control access.

Port	Component	Purpose	Security Considerations
3300, 6789	Ceph MON	Monitor daemon client communication	Should ideally be restricted to internal networks and specific client subnets via firewall.
6800-7300	Ceph OSD/MGR/M	Intra-cluster communication	Must be strictly firewalled from external access. Essential for cluster operation.
80	RGW (HTTP)	RADOS Gateway (Object storage access)	Object storage access. Only enable if needed.
443	RGW (HTTPS)	RADOS Gateway secure traffic (HTTPS)	Object storage access. Requires TLS certificate management (see Encryption section). Only enable if needed.
9283	MGR (Dashboard)	Ceph Dashboard HTTPS access	Access should be restricted via firewall. Authentication is required.
9128	MGR (Prometheus)	Prometheus metrics endpoint	Restrict access to monitoring servers via firewall.
Internal/Local	microcephd	Local API socket for microceph CLI	Access controlled by filesystem permissions on the socket file within the snap's data directory.
7443	microcephd	Inter-node communication (if clustered)	Uses TLS. Must be firewalled from external access, allowing only cluster members.
22	SSH	Host OS access	Standard SSH hardening practices (key auth, restricted access, firewall).
Other	Other Services	Potentially other services on host	Audit all open ports on the host system.

Network protocols and endpoints

- Ceph Protocol (Messenger v1/v2): Used for all internal Ceph communication (MON, OSD, MGR, MDS). Messenger v2 (default in newer Ceph versions) provides encryption capabilities for data in transit.
- Microcluster Protocol: Used for communication between microcephd instances in a multi-node cluster. This communication is secured using TLS.
- Cephx Authentication: Primary mechanism for authenticating Ceph internal and client communication. It provides mutual authentication.
- HTTP/HTTPS (RGW): Used for S3/Swift access via the RADOS Gateway. HTTPS with strong TLS configuration is best practice.
- SSH: Used for accessing the host system to run microceph commands and perform system maintenance.
- Local Socket API (microcephd): Communication between microceph CLI and microcephd occurs over a Unix domain socket, protected by filesystem permissions.

Data interfaces

- Block Devices and Filesystems: OSDs interact directly with underlying storage (disks, partitions, or files configured via microceph disk add). The OSD processes require elevated privileges, managed within the snap's confinement.
- dqlite Database Files: microcephd reads/writes configuration and state to dqlite database files located within the snap's data directory (e.g., /var/snap/microceph/common/state/). Access is controlled by filesystem permissions.
- CephFS Mounts: Clients mounting CephFS interact via the Ceph kernel module or FUSE, requiring Cephx authentication.

Management infrastructure

The primary management attack surface is the host, snap environment, and the microcephd service:

- microceph CLI: Accessing this command usually requires sudo privileges on the host. Compromising a host would allow an attacker to impact the Ceph cluster.
- microcephd: Compromising the microcephd process could allow manipulation of the cluster state and Ceph daemon configuration. Vulnerabilities in microcephd or the underlying Microcluster library are potential vectors.
- Host OS: Compromise of the host OS grants control over MicroCeph, including access to microcephd and its database. Standard host hardening is advised.
- Snap Environment (snapd): Vulnerabilities in snapd or the MicroCeph snap package itself could be vectors. Note that MicroCeph is running with strict snap confinement; see here for details on confinement.
- Ceph Dashboard: If enabled, secure its access via network controls and strong authentication.

Access controls

Robust access controls limit users and services to only the permissions they require.

Cephx authentication and authorization

Cephx remains the native Ceph authentication system for Ceph client/daemon interactions.

- Key Types: Use distinct keys for different roles (admin, osd, mds, client). Manage these using standard Ceph commands prefixed with sudo microceph.ceph (e.g., sudo microceph.ceph auth add …).
- Capabilities (Caps): Assign the minimum necessary capabilities to each key e.g., `mon`, `allow r`, `osd`, `allow`). Avoid using the client.admin key for applications.

User management (Ceph Dashboard / RGW)

- Dashboard Users: Manage user accounts and roles within the Ceph Dashboard for accessing monitoring and limited management functions.
- RGW Users: If using RGW, manage its separate S3/Swift users, keys (access key, secret key), and potentially quotas using RGW admin commands (sudo microceph.radosgw-admin …).

Management infrastructure access

Control access at multiple levels:

- Host OS Access (POSIX permissions): Implement standard Linux user/group permissions, strong SSH key management, and tightly controlled sudo rules on the host machine where MicroCeph runs. This governs access to the microceph CLI and the microcephd's data files/socket.
- microcephd Access: Interaction with microcephd is primarily via the microceph CLI, which requires appropriate host OS permissions (sudo). Direct access to the microcephd API socket is protected by file permissions.
- Snap Confinement: Rely on the isolation provided by snapd as a baseline security feature for the snap's processes and data.
- Elevated Privileges: OSD processes require superuser privileges for device access, which is managed internally by MicroCeph and the snap environment.

Secrets

Protect sensitive information:

- Cephx Keys: Stored within the snap's data directory (e.g., /var/snap/microceph/current/conf). Protect host access.
- TLS Certificates & Keys: For microcephd inter-node communication (if clustered). These are typically managed internally by Microcluster/MicroCeph but stored within the snap's common directory.
- dqlite Database Files: The database files in /var/snap/microceph/common/state/ contain sensitive cluster configuration. Protect host access and ensure correct file permissions.
- RGW User Keys: S3/Swift access and secret keys. Treat these like passwords; manage and distribute them securely.

Encryption

Protecting data confidentiality both in transit and at rest:

In transit:

- Ceph Messenger v2: Configure Ceph internal communication (between MON, OSD, MGR, MDS) to use secure mode via Ceph configuration options (e.g., ms_cluster_mode = secure).
- microcephd Communication: Communication between microcephd nodes in a cluster is secured using TLS by default.
- TLS at RGW: Essential for encrypting S3/Swift traffic. Use strong TLS protocols (TLS 1.2+) and ciphers. Obtain certificates from a trusted CA or manage an internal PKI.
- Ceph Dashboard HTTPS: The dashboard uses HTTPS by default.
- microceph CLI to microcephd: Communication occurs over a local Unix domain socket and is not typically encrypted itself, relying on filesystem permissions for security.

At rest:

- OSD Encryption (via LUKS): MicroCeph supports encrypting data stored on OSDs using LUKS, configured during disk addition (via flags to microceph disk add). This protects data if physical drives are compromised. Key management is handled by MicroCeph.
- dqlite Database: The dqlite database files themselves are not typically encrypted at rest by default. Protection relies on standard filesystem permissions and potentially Full Disk Encryption (FDE) of the host system.
- Full Disk Encryption (FDE): Consider encrypting the entire host OS disk, to protect Ceph keys, the dqlite database, configs, and potentially cached data against physical access. Manage FDE at the OS level.

Secure deployment

Incorporate security from the initial setup of your MicroCeph instance.

Network architecture

- Segmentation: If the MicroCeph host has multiple network interfaces, configure Ceph's public_network and cluster_network settings appropriately (check MicroCeph docs for details), configure the microcephd listen/advertise addresses if needed for clustering, and use the firewall to enforce segregation between client access networks, cluster networks, and management networks.
- As a best practice, use firewalling or VLANs to segregate into these zones:

  - External (optional): If applicable, expose specific endpoints for external untrusted consumption, e.g. RGW.
  - Storage Access: Client access (including RGW if no external access provided), MON access.
  - Cluster Network: OSD replication and heartbeat traffic. Isolating this improves performance and security.
- Firewalls: Implement strict firewall rules (e.g. using iptables, nftables) on all nodes:
  - Deny all traffic by default.
  - Allow only necessary ports between specific hosts/networks (refer to the port table).
  - Restrict access to management interfaces (SSH, Juju, Dashboard) to trusted administrative networks.

Minimum privileges

- Cephx Keys: Create dedicated Cephx keys for each client/application with minimal capabilities. Don't use client.admin routinely.
- OS Users: Limit sudo access on the host machine. Restrict who can run microceph commands. Run other applications on the host as unprivileged users. Protect access to the snap's data directories.
- Explicit Assignment: Ensure all access relies on explicit permissions/capabilities rather than default permissive settings.

Auditing and centralized logging

- Enable Auditing:
  - Configure Ceph logging levels via Ceph configuration options (e.g., log_to_file = true, debug_mon, debug_osd). Check MicroCeph documentation for how to set these. Ceph logs are found in /var/snap/microceph/common/logs/ceph/.
  - microcephd logs to /var/log/syslog, see the MicroCeph documentation for details on setting log levels.
- Centralized Logging: Configure host-level standard log shipping mechanisms (e.g., rsyslog, journald forwarding) to send Ceph logs, microcephd logs, and host system logs (syslog, auth.log, kern.log, journald) to a central logging system (like Loki or ELK).
- Monitor and Audit: Regularly review logs for anomalies and security events (e.g., repeated auth failures, crashes, microcephd errors).

Alerting

- Configure Monitoring: Enable the Prometheus MGR module (sudo microceph.ceph mgr module enable Prometheus) and configure it if necessary via Ceph MGR configuration options (e.g., sudo microceph.ceph config set mgr mgr/prometheus/. . .).
- Security Alerts: Configure alerts for security anomalies and health issues such as:
  - Ceph health status changes (HEALTH_WARN, HEALTH_ERR).
  - Ceph daemon crashes or restarts (via systemd unit status or logs).
  - microcephd service failures or restarts.
  - Significant performance deviations.
  - Host system issues (CPU, RAM, Disk I/O).

Secure operation

Maintaining security is an ongoing process.

Vulnerability management

- Monitor Advisories: Actively track CVEs and security advisories for:
  - Ceph (via Ceph announce list, security trackers).
  - MicroCeph snap (check snap channels/updates).
  - Host OS (use relevant security advisories for the host OS, e.g., USNs for Ubuntu).
- Patch Management: Implement a process for testing and applying security patches promptly using sudo snap refresh microceph and the host OS's package manager (e.g., apt update && apt upgrade for Debian/Ubuntu). Use snap channels (e.g., the /candidate channel) for testing before refreshing stable.

Incident response

- Develop a Plan: Have a documented Incident Response (IR) plan for your MicroCeph environment, including steps related to microcephd and the dqlite database.
- Define Steps: Cover detection, containment (e.g., firewalling the host, stopping services like snap.microceph.daemon), eradication, recovery (potentially involving database restoration if needed), and post-mortem analysis.
- Practice: Test the plan periodically.

Perform audits

- Regular Checks: Conduct periodic security audits of the MicroCeph host, configuration, and data directories.
- Validate Controls: Verify firewall rules, Ceph configuration, microcephd status and configuration, Cephx permissions (sudo microceph.ceph auth ls), OS access controls (sudo rules, SSH keys, file permissions on /var/snap/microceph/), and encryption settings.

Perform upgrades

- Stay Current: Regularly upgrade MicroCeph (sudo snap refresh microceph), snapd (sudo snap refresh snapd), and the underlying OS (using the host's package manager) for security patches and features. Upgrading the MicroCeph snap updates Ceph, microcephd, dqlite, and Microcluster together.
- Schedule Proactively: Plan and test upgrades, especially for security vulnerabilities. Utilize snap channels for pre-production testing.

Release notes

- Always read the release notes for Ceph versions included in MicroCeph snap updates, the MicroCeph snap itself, and the host OS before upgrading or making significant changes, as they contain information about security fixes, new features, and potential issues.

If you have a specific goal, but are already familiar with MicroCeph, our *how-to guides* have more in-depth detail and instructions.

Take a look at our *reference* section when you need to know which MicroCeph commands to use.

2.5 Contribute to our documentation

Contributing to documentation is a great way to get started as a contributor to open-source projects, no matter your level of experience.

MicroCeph is growing rapidly, and we would love your help. We welcome, encourage and appreciate contributions from our user community in the form of suggestions, fixes and constructive feedback. Whether you are new to MicroCeph and want to highlight something you found confusing, or you're an expert and want to create a how-to guide to help others, we will be happy to work with you to make our documentation better for everybody.

Raise an issue in our GitHub repository or talk to us on our Matrix channel.

We hope to make it as easy as possible to contribute. If you feel something is unclear, wrong, or broken, please don't hesitate to leave a comment in the Matrix room.

2.5.1 MicroCeph documentation overview

The MicroCeph documentation is hosted in a GitHub repository and published on Read the Docs. You need to create a GitHub account to participate, but you do not need a Read the Docs account.

Contributing on GitHub

To create issues, comment, reply, or submit contributions, you need to set up a GitHub account and a Git environment. You don't need to know Git before you start, and you definitely don't need to work on the command line if you don't want to. Many documentation tasks can be done using GitHub's web interface. On the command line, we use the standard fork and pull model. Learn more about how to work with Git here.

For spelling and grammatical changes, which are quick and easy to submit, feel free to create a Pull Request (PR). For more substantial changes or suggestions, we recommend creating an issue first, so that we can discuss and agree on an approach before you spend time working on it.

Make sure to check the issues list before submitting a PR - if you start working on a task that is listed and already assigned to someone else, we won't be able to accept your PR.

The CLA check

If it's your first time contributing to a Canonical project, you will need to sign the Canonical Contributor Licence Agreement before your contribution can be considered for inclusion within our project. If you have already signed it, e.g. when contributing to another Canonical project, you do not need to sign it again. This licence protects your copyright over your contributions, including the right to use them elsewhere, but grants us (Canonical) permission to use them in our project. Our project repository will automatically check whether a contributor has signed the CLA when a contribution is made.

2.5.2 Diátaxis

Our documentation content, style and navigational structure follows the Diátaxis systematic framework for technical documentation authoring. This framework splits documentation pages into tutorials, how-to guides, reference material and explanatory text:

- **Tutorials** are lessons that accomplish specific tasks through *doing*. They help with familiarity and place users in the safe hands of an instructor.
- **How-to guides** are recipes, showing users how to achieve something, helping them get something done. A *How-to* has no obligation to teach.
- **Reference** material is descriptive, providing facts about functionality that is isolated from what needs to be done.
- **Explanation** is discussion, helping users gain a deeper or better understanding of MicroCeph, as well as how and why MicroCeph functions as it does.

To learn more about our Diátaxis strategy, see Diátaxis, a new foundation for Canonical documentation.

Improving our documentation and applying the principles of Diátaxis are on-going tasks. There's a lot to do, and we don't want to deter anyone from contributing to our docs. If you don't know whether something should be a tutorial, how-to, reference doc or explanatory text, either ask on the forum or publish what you're thinking. Changes are easy to make, and every contribution helps.

2.5.3 Open Documentation Academy

MicroCeph is a proud member of the Canonical Open Documentation Academy (CODA), an initiative led by the documentation team at Canonical to provide help, advice, mentorship, and dozens of different tasks to get started on, within a friendly and encouraging environment.

A key aim of this initiative is to help lower the barrier into successful open-source software contribution, by making documentation into the gateway, and it's a great way to make your first open source documentation contributions to MicroCeph.

But even if you're an expert, we want the academy to be a place to share knowledge, a place to get involved with new developments, and somewhere you can ask for help on your own projects.

The best way to get started is with our task list . Take a look, bookmark it, and see our Getting started guide for next steps.

Stay in touch either through the task list, or through one of the following locations:

- Our discussion forum on the Ubuntu Community Hub.
- In the Matrix room for interactive chat.
- Follow us on Fosstodon for the latest updates and events.

If you'd like to ask us questions outside of our public forums, feel free to email us at docsacademy@canonical.com.

In addition to the above, we have a weekly Community Hour starting at 16:00 UTC every Friday. Everyone is welcome, and links and comments can be found on the forum post.

Finally, subscribe to our Documentation event calendar. We'll expand our Community Hour schedule and add other events throughout the year.

Agreements

Everyone involved with CODA needs to follow the words and spirit of the Ubuntu Code of Conduct v2.0. You must sign and agree to the Canonical CLA.

Identifying suitable task

The academy uses issue labels to give the contributor a glimpse into the task and what it requires, including the type of task, skills or level of expertise required, and even the size estimation for the task. You can find tasks of all sizes in the academy issues list.

From small tasks, such as replacing outdated terminology, checking for broken links, testing a tutorial or ensuring adherence to the Canonical documentation style guide; to medium-sized tasks like, converting documentation from one format to another, or migrating the contents of a blog post into the official documentation; to more ambitious tasks, such as adding a new *How-to* guide, restructuring a group of documents, or developing new tests and automations.

Completing and closing tasks

When a task has been completed to your satisfaction, we'll ask the contributor whether they would prefer to merge their work into your project themselves, or leave this to the project.

Recognition

After successfully completing a task, we'll give credit to the contributor and share their success in our forums, on the pages themselves, and in our news updates and release notes.

2.5.4 Guidance for writing

Consistency of writing style in documentation is vital for a good user experience. To accommodate our audience with a huge variation in experience, we:

- write with our target audience in mind
- write inclusively and assume very little prior knowledge of the reader
- link or explain phrases, acronyms and concepts that may be unfamiliar, and if unsure, err on the side of caution
- adhere to the style guide

Language

Canonical previously used British (GB) English, so you may notice that older documentation is in this format. However, we have recently switched to US English. It's a good idea to set your spellchecker to `en-US`; which will pick up most of the inconsistencies. If it doesn't, they will be picked up in review by the documentation team.

There are many small differences between UK and US English, but for the most part, it comes down to spelling. Some common differences are:

- the *ize* suffix in preference to *ise* (e.g. capitalize rather than capitalise)
- *our* instead of *or* (as in color and colour)
- licence as both a verb and noun
- catalog rather than catalogue
- dates take the format 1 January 2013, 1-2 January 2025 and 1 January - 2 February 2025

We use an automated spelling checker that sometimes throws errors about terms we would like it to ignore:

- If it complains about a file name or a command, enclose the word in backticks (`) to render it as inline code.
- If the word is a valid acronym or a well-known technical term (that should not be rendered as code), add it to the spelling exception list, `.custom_wordlist.txt` (terms should be added in alphabetical order).

Both methods are valid, depending on whether you want the term to be rendered as normal font, or as inline code (monospaced).

Acronyms

Acronyms should always be capitalized.

They should always be expanded the first time they appear on a page, and then can be used as acronyms after that. E.g. OSD should be shown as Object Storage Daemon (OSD), and then can be referred to as OSD for the rest of the page.

Links

The first time you refer to a package or other product, you should make it a link to either that product's website, or its documentation, or its manpage.

Links should be from reputable sources (such as official upstream docs). Try not to include blog posts as references if possible. And, always verify that the links are correct and accurate.

Try to use inline links sparingly. If you have a lot of useful references you think the reader might be interested in, feel free to include a "Further reading" section at the end of the page.

Writing style

Try to be concise and to-the-point in your writing.

It’s alright to be a bit light-hearted and playful in your writing, but please keep it respectful, and don’t use emoji (they don’t render well in documentation, and may not be deemed professional).

It’s also good practice not to assume that your reader will have the same knowledge as you. If you’re covering a new topic (or something complicated) then try to briefly explain, or link to supporting explanations of, the things the typical reader may not know, but needs to (refer to the Diátaxis framework to help you decide what type of documentation you are writing and the level and type of information you need to include, e.g. a tutorial may require additional context but a how-to guide can skip some foundational knowledge - it is safer to assume some prior knowledge).

Thank you

We would like to thank you for spending your time to help make the MicroCeph documentation better. Every contribution, big or small, is important to us, and hopefully a step in the right direction.