Canonical

MicroCeph

Canonical

Deploy from Snap package

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.

1. In this documentation

MicroCeph can be deployed and managed as a standalone snap or as a charm as part of a Juju model.

MicroCeph snap › (page 4) The `microceph` snap is a self-contained, secure and dependency-free Linux app package used to deploy and manage a Ceph cluster. If you are new to MicroCeph, start here.

MicroCeph charm › (page 118) The `microceph` charm takes care of installing, configuring and managing MicroCeph on cloud instances managed by Juju.

2. 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.

2.1. Get involved

- Contribute to the project on the MicroCeph[1] or charm-microceph[2] GitHub repositories _ (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 Ceph General[3] or Ceph Devel[4]
- *Contribute to our documentation* (page 131)

2.2. Governance and policies

- We follow the Ubuntu community Code of conduct[5]

2.3. Commercial support

- Optionally enable Ubuntu Pro[6] on your Ceph nodes. This is a service that provides the Livepatch Service[7] and the Expanded Security Maintenance[8] (ESM) program.

2.3.1. Deploy a single-node MicroCeph cluster

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.

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.

[1] https://github.com/canonical/microceph
[2] https://github.com/canonical/charm-microceph
[3] https://matrix.to/#/#ubuntu-ceph:matrix.org
[4] https://matrix.to/#/#ceph-devel:ubuntu.com
[5] https://ubuntu.com/community/ethos/code-of-conduct
[6] https://ubuntu.com/pro
[7] https://ubuntu.com/security/livepatch
[8] https://ubuntu.com/security/esm

What you'll need

- The latest Ubuntu LTS version. See the Ubuntu release cycle[9] for release information.
- 2 CPU cores
- 4 GiB RAM
- 12GiB disk space
- An Internet connection

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.

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:

```
user@host:~$

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

[9] https://ubuntu.com/about/release-cycle

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.

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
```

```
user@host:~$

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

```
user@host:~$

MicroCeph deployment summary:
- ubuntu (10.246.114.49)
Services: mds, mgr, mon, osd
Disks: 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.

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
```

```
user@host:~$

MicroCeph deployment summary:
- ubuntu (10.246.114.49)
Services: mds, mgr, mon, rgw, osd
Disks: 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 --access-key=foo --secret-key=bar
```

The output should include user details as shown below.

```
user@host:~$

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

Note:

If `access-key` or `secret-key` is not provided by the user, it will be generated automatically.

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
```

```
user@host:~$

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

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

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

```
user@host:~$

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

Note:

Make note of the IP address used above, as it will is reused in subsequent steps.

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 `aws-cli` utility is installed and configured.

Install and configure `aws-cli`

To install `aws-cli`, run the following command:

```
sudo snap install aws-cli --classic
```

Run the below command to interactively populate required parameters as follows:

```
aws configure
```

```
user@host:~$

AWS Access Key ID [****************foo]: foo
AWS Secret Access Key [****************bar]: bar
Default region name [default]: default
Default output format [None]:
```

Create a bucket

You have verified that your cluster is accessible via RGW. Now, let's create a bucket using the `aws s3` command:

```
aws s3 mb s3://mybucket --endpoint=http://10.246.114.49
```

```
user@host:~$

make_bucket: mybucket
```

Our bucket is successfully created.

Upload a file into the bucket

```
aws s3 cp ./image.jpg s3://mybucket --endpoint=http://10.246.114.49
```

```
user@host:~$

upload: ./image.jpg to s3://mybucket/image.jpg
```

The output shows that your image is now stored in a S3 bucket.

Listing the contents of a bucket

Let's list the contents of `s3:mybucket` and see if our image is present there.

```
aws s3 ls s3://mybucket --endpoint=http://10.246.114.49
```

```
user@host:~$

2025-10-15 12:51:03          0 image.jpg
```

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.

```
user@host:~$

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
```

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* (page 10), packed with instructions to help you achieve specific goals with MicroCeph.

Or, explore our *Explanation* (page 83) and *Reference* (page 49) sections for additional information and quick references.

2.3.2. How-to guides

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

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 guides section* (page 10).

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

```
sudo ceph status
```

The cluster built during this tutorial gives the following output:

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

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

data:
  pools:   1 pools, 1 pgs
```

(continues on next page)

(continued from previous page)

```
  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

eyJuYW1lIjoibm9kZS0yIiwic2VjcmV0IjoiYmRjMzZlOWJmNmIzNzhiYzMwY2ZjOWVmMzRjNDM5YzNlZTMzMTlmZDIyZjkx
```

Token for node-3:

```
sudo microceph cluster add node-3

eyJuYW1lIjoibm9kZS0zIiwic2VjcmV0IjoiYTZjYWJjOTZiNDJkYjg0YTRkZTFiY2MzY2VkYTI1M2Y4MTU1ZTNhYjAwYWUy
```

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
eyJuYW1lIjoibm9kZS0yIiwic2VjcmV0IjoiYmRjMzZlOWJmNmIzNzhiYzMwY2ZjOWVmMzRjNDM5YzNlZTMzMTlmZDIyZjkx
```

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

```
sudo microceph cluster join
eyJuYW1lIjoibm9kZS0zIiwic2VjcmV0IjoiYTZjYWJjOTZiNDJkYjg0YTRkZTFiY2MzY2VkYTI1M2Y4MTU1ZTNhYjAwYWUy
```

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 *how-to guides section* (page 10).

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

services:
  mon: 3 daemons, quorum node-01,node-02,node-03 (age 14m)
```

(continues on next page)

(continued from previous page)

```
  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
```

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_u	The number of seconds dirty data is in the cache before write-back 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 client config list
+---+---------------+---------+----------+
| # |      KEY      |  VALUE  |   HOST   |
+---+---------------+---------+----------+
| 0 | rbd_cache     | true    | beta     |
+---+---------------+---------+----------+
| 1 | rbd_cache     | false   | alpha    |
+---+---------------+---------+----------+
| 2 | rbd_cache_size | 2048MiB | beta     |
+---+---------------+---------+----------+
```

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

```
$ sudo microceph client 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 client 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 client 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 client config reset rbd_cache_size
$ sudo microceph client 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 client config reset rbd_cache --target alpha
$ sudo microceph client 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[10] 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:

[10] https://docs.openstack.org/keystone/latest/getting-started/architecture.html#identity

Table 2: Supported Config Keys

Key	Description
rgw_s3_auth_use_keyston	Whether to use keystone auth for the S3 endpoints.
rgw_keystone_url	Keystone server address in {url:port} format
rgw_keystone_admin_toke	Keystone admin token (not recommended in production)
rgw_keystone_admin_toke	Path to Keystone admin token (recommended for production)
rgw_keystone_admin_user	Keystone service tenant user name
rgw_keystone_admin_pass	Keystone service tenant user password
rgw_keystone_admin_pass	Path to Keystone service tenant user password file
rgw_keystone_admin_proj	Keystone admin project name
rgw_keystone_admin_dom	Keystone admin domain name
rgw_keystone_service_tok	Whether to allow expired tokens with service token in requests
rgw_keystone_service_tok	Specify user roles accepted as service roles
rgw_keystone_expired_tol	Cache expiration period for an expired token allowed with a service token
rgw_keystone_api_version	Keystone API version
rgw_keystone_accepted_r	Accepted user roles for Keystone users
rgw_keystone_accepted_a	List of roles allowing user to gain admin privileges
rgw_keystone_token_cach	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_ter	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_ena	Enables object versioning
rgw_swift_enforce_conter	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 |
+---+--------------------------+-------+
| 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 the upstream Ceph documentation on RGW Keystone configuration[11].

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:

[11] https://docs.ceph.com/en/latest/radosgw/keystone/

```
$ 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   |
+---+-----------------+-------------+
| 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 *Cluster network configurations* (page 83).

Enabling metrics collection with Prometheus

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

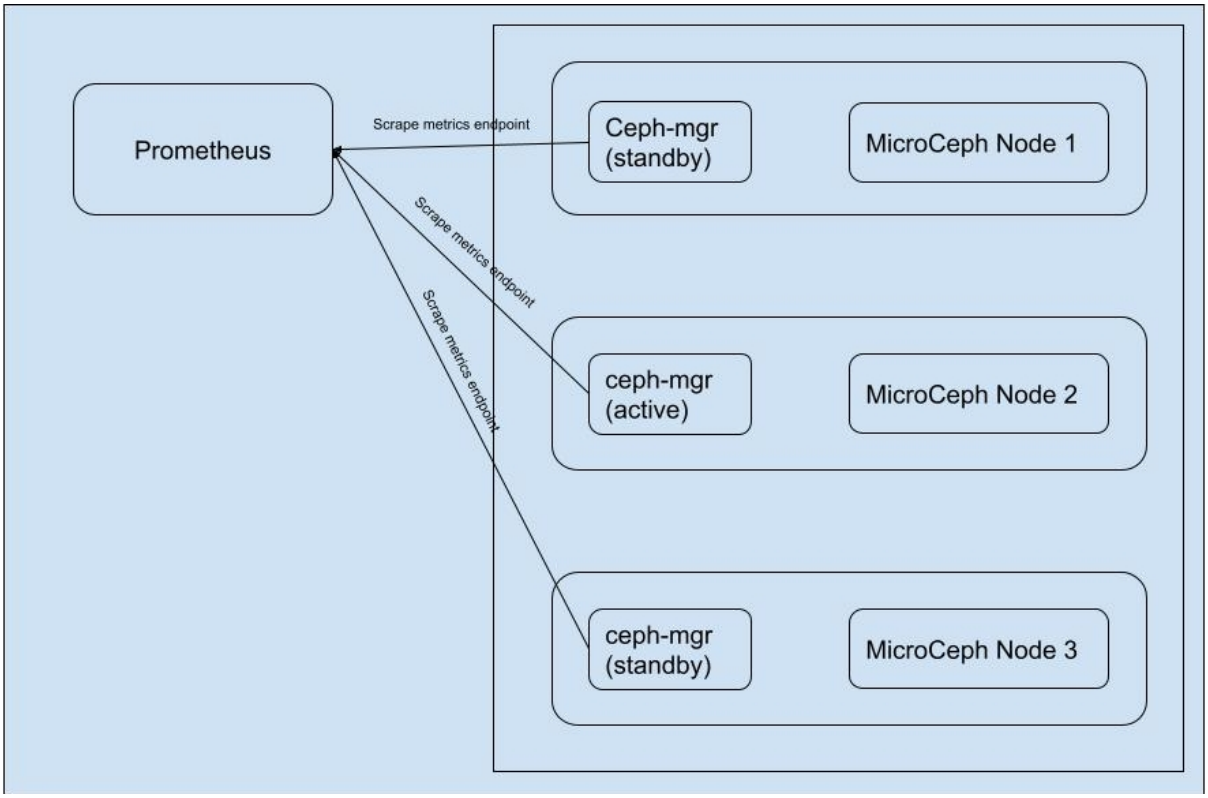


Fig. 1: Prometheus service scraping endpoints of a multi-node MicroCeph cluster.

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[12].

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[13]

A simple configuration file is provided below:

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

# Scrape Job
scrape_configs:
  - job_name: 'microceph'

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

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

Start Prometheus with provided configuration file.

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

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

Enable Prometheus Alertmanager alerts

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[14]. These rules allow 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 `prometheus_alerts.yaml`.

[12] https://docs.ceph.com/en/latest/mgr/prometheus/
[13] https://prometheus.io/docs/prometheus/latest/configuration/configuration/
[14] https://prometheus.io/docs/prometheus/latest/configuration/alerting_rules/

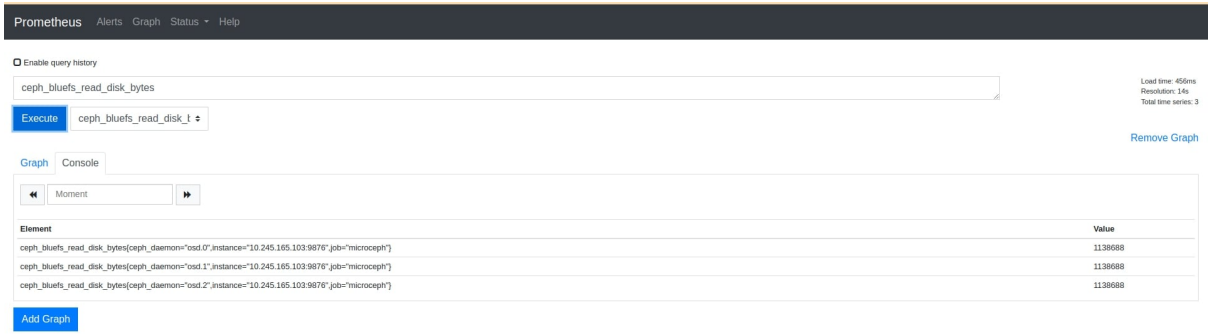


Fig. 2: A Prometheus console displaying scraped metric from MicroCeph cluster.

Prerequisites

In order to configure alerts, your MicroCeph deployment must enable metrics collections with Prometheus. Follow the *guide on enabling metrics collection with Prometheus* (page 21) if you haven't configured it. Also, Alertmanager is distributed as a separate binary which should be installed and running.

Configure Alert rules

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

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

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

# Scrape Job
scrape_configs:
  - job_name: 'microceph'

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

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

(continues on next page)

(continued from previous page)

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

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

Start Prometheus with provided configuration file.

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

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

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

Enable 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[15])
- MDS (Metadata service[16])
- MGR (Manager service[17])

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:

- NFS
- RGW (RADOS Gateway service[18])
- cephfs-mirror

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', 'nfs-<cluster-id>' and 'rgw'. The destination is a node name as discerned by the output of the **status** command:

[15] https://docs.ceph.com/en/latest/man/8/ceph-mon/
[16] https://docs.ceph.com/en/latest/man/8/ceph-mds/
[17] https://docs.ceph.com/en/latest/mgr/
[18] https://docs.ceph.com/en/latest/radosgw/

```
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
```

Let's take an example of enabling RGW and NFS services, viewing the possible extra parameters for both services.

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
```

Enable an NFS service

View any possible extra parameters for the NFS service:

```
sudo microceph enable nfs --help
```

To enable the NFS service on `node1`, and specify values for the extra parameters:

```
sudo microceph enable nfs --cluster-id foo-cluster --v4-min-version 2 --target
node1
```

View cluster status again and verify the expected changes:

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

Note:

Enabling NFS on multiple nodes with the same `--cluster-id` will effectively result in the running NFS services to be grouped in the same service cluster.

Caution:

Nodes in the same NFS service cluster **must** have matching configuration (`--v4-min-version`), otherwise MicroCeph will return an error when adding new nodes to the cluster.

Caution:

A node may join only one NFS service cluster. MicroCeph will return an error if there's already a NFS service registered on the node. If a node would have to join a different NFS service cluster, it would have to leave the original cluster first:

```
sudo microceph disable nfs --cluster-id foo-cluster --target node1
```

After the NFS cluster has been set up, you can create NFS shares. Next will be a basic example in which we're creating an NFS export and mounting it.

Create a volume:

```
sudo microceph.ceph fs volume create foo-vol
```

Create an NFS export by running the following command:

```
sudo microceph.ceph nfs export create cephfs foo-cluster /fs-foo-dir foo-vol

# Sample output:
{
  "bind": "/fs-foo-dir",
  "cluster": "foo-cluster",
  "fs": "foo-vol",
  "mode": "RW",
  "path": "/"
}
```

A client may now mount the NFS share. They will first need the `nfs-common` package:

```
sudo apt install nfs-common
```

Finally, the client will be able to mount the NFS share:

```
sudo mount -o rw -t nfs "nfs-bind-address:/fs-foo-dir /mnt
```

Enable cephfs-mirror service

View any possible extra parameters for the `cephfs-mirror` service:

```
sudo microceph enable cephfs-mirror --help
```

To enable the `cephfs-mirror` service on `node1`:

```
sudo microceph enable cephfs-mirror --target node1
```

View cluster status again and verify the expected changes:

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

Note:

At the moment, the `cephfs-mirror` service can only be enabled once per cluster.

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[19]
- MDS - Metadata service[20]
- MGR - Manager service[21]

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.

Note:

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.

[19] https://docs.ceph.com/en/latest/man/8/ceph-mon/
[20] https://docs.ceph.com/en/latest/man/8/ceph-mds/
[21] https://docs.ceph.com/en/latest/mgr/

Remove a disk

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 explanation* (page 85)
- the *disk command reference* (page 57)

Note:

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

Procedure

First get an overview of the cluster and its OSDs:

```
ceph status
```

Example output:

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

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

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

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

```
ceph osd tree
```

Sample output:

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

(continues on next page)

(continued from previous page)

```
 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

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

Before proceeding, please refer to the *cluster maintenance explanation* (page 85) to understand its functionality and impact.

Enable 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
```

Disable 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>
```

Rotate RGW TLS certificates

If you have RGW running with SSL enabled, you can rotate its TLS certificates without needing to disable and re-enable the service.

Prerequisites

- RGW must already be enabled with SSL. See *enable service instances* (page 25) for details on enabling RGW with `--ssl-certificate` and `--ssl-private-key`.
- The replacement certificate and private key must be base64 encoded.

Rotate with immediate effect

Use the `--restart` flag to write the new certificate and key to disk and restart the RGW service immediately. This will drop any existing client connections.

```
sudo microceph certificate set rgw \
  --ssl-certificate "$(base64 -w0 /path/to/new-server.crt)" \
  --ssl-private-key "$(base64 -w0 /path/to/new-server.key)" \
  --restart
```

Write certificate without restart

Without `--restart`, the certificate and key are written to disk but the RGW service continues serving the old certificate. You must restart the service manually for the change to take effect.

```
sudo microceph certificate set rgw \
  --ssl-certificate "$(base64 -w0 /path/to/new-server.crt)" \
  --ssl-private-key "$(base64 -w0 /path/to/new-server.key)"
```

Rotate on a specific node

In a multi-node cluster, each node has its own certificate files. Use `--target` to rotate the certificate on a specific node. Repeat for each node that runs RGW:

```
sudo microceph certificate set rgw \
  --ssl-certificate "$(base64 -w0 /path/to/new-server.crt)" \
  --ssl-private-key "$(base64 -w0 /path/to/new-server.key)" \
  --target node2 \
  --restart
```

Verify the certificate

After restarting the RGW service, verify that the new certificate is being served. The SSL port is the value passed to `microceph enable rgw --ssl-port` (default: 443). You can confirm the port by inspecting the RGW configuration:

```
sudo grep ssl_port /var/snap/microceph/current/conf/radosgw.conf
```

Then verify with:

```
echo | openssl s_client -connect localhost:443 2>/dev/null \
  | openssl x509 -noout -subject -dates
```

Replace `443` with your configured SSL port if different.

Enable full disk encryption in MicroCeph

Full disk encryption (FDE) in MicroCeph allows operating encrypted OSDs in a MicroCeph cluster. See the *FDE explanation* (page 101) to learn more about FDE protection and its limitations.

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[22] 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[23] has to be connected, and `microceph.daemon` subsequently restarted:

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

Enable FDE

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

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

Note that 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.

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
eyJmc2lkIjoiN2FiZmMwYmItNjIwNC00M2FmLTg4NDQtMjg3NDg2OGNiYTc0Iiwia2V5cmluZy5jbGllbnQubWFnaWNhbCI6
```

[22] https://www.kernel.org/doc/html/latest/admin-guide/device-mapper/dm-crypt.html
[23] https://snapcraft.io/docs/reference/interfaces/dm-crypt-interface/#interfaces-dm-crypt-interface

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

```
sudo microceph remote import simple
eyJmc2lkIjoiN2FiZmMwYmItNjIwNC00M2FmLTg4NDQtMjg3NDg2OGNiYTc0Iiwia2V5cmluZy5jbGllbnQubWFnaWNhbCI6
--local-name magical
```

This will create the required $simple.conf and $simple.keyring files. Note: Importing a remote cluster is a uni-directional 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
  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 our *guide on importing a remote cluster* (page 34).
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     |
+-------------+---------+-------------+--------+

+------------------+-----------+--------------------------------------+
|    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.

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

Disable 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 | im-
age_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 replica-
tion list rbd +————+————+————+————————+ | POOL NAME | IMAGE NAME
| IS PRIMARY | LAST LOCAL UPDATE | +————+————+————+————————+ |
pool_two | image_one | true | 2024-10-08 13:55:12 | +————+————+————+——
————+
```

Configure CephFS replication

CephFS is a POSIX-compliant file system over Ceph's distributed storage layer consumed by various cloud workloads. For improved resiliency and disaster recovery, it is often desirable to replicate CephFS data to a remote cluster.

MicroCeph replication framework supports enabling replication for CephFS workloads to a remote MicroCeph cluster.

This guide describes how an operator can enable, monitor, and disable CephFS replication for any subvolume or a directory path relative to the parent volume using MicroCeph CLI commands.

Note:

The CLI commands described in this guide are to be executed on the primary MicroCeph cluster.

Prerequisites

1. A primary and a secondary MicroCeph cluster, for example named `primary_cluster` and `secondary_cluster`.
2. Both `primary_cluster` and `secondary_cluster` have imported exchanged cluster tokens. Refer to our *guide on importing a remote cluster* (page 34).
3. Both clusters have an active CephFS volume called `vol`.
4. Both clusters have exactly one `cephfs-mirror` daemon enabled. Refer to our *guide on enabling the cephfs-mirror service* (page 28).

Enable CephFS replication

An operator can enable replication for a given cephfs directory path as follows:

```
user@host:~$

none

sudo microceph replication enable cephfs --volume vol --remote secondary_
cluster --dir-path </path/to/directory>
```

Here, `/path/to/directory` is the desired directory path relative to volume `vol` as root.

Similarly, an operator can enable replication for a given subvolume as follows:

```
user@host:~$

none

sudo microceph replication enable cephfs --volume vol --subvolume <subvolume>
--subvolumegroup <subvolumegroup>
```

Here, `<subvolume>` is the name of the subvolume and `<subvolumegroup>` is the name of the parent subvolume group. If the subvolume does not have a parent subvolume group, then `<subvolumegroup>` can be omitted.

List all CephFS replication images

An operator can list all the resources enabled for replication as follows:

```
user@host:~$

none

sudo microceph replication list cephfs
```

```
+--------+-----------------------------------+-----------+
| VOLUME | RESOURCE                          | TYPE      |
+--------+-----------------------------------+-----------+
| vol    | /dir1/dir2/dir3                   | directory |
| vol    | /volumes/subvol_group/subvol      | subvolume |
| vol    | /volumes/_nogroup/ungrouped_subvol | subvolume |
+--------+-----------------------------------+-----------+
```

Check CephFS replication status

An operator can check the current replication status of a volume as follows:

```
user@host:~$

none

sudo microceph replication status cephfs vol
```

```
+--------------------------+
|          SUMMARY         |
+----------------+---------+
| Volume         | vol     |
| Resource Count | 3       |
| Peer Count     | 1       |
+----------------+---------+

+---------------------+---------------------------+--------+-------------+------
--------+---------------+
| REMOTE NAME         | RESOURCE PATH                     | STATE  | SNAPS SYNCED
| SNAPS DELETED | SNAPS RENAMED |
+---------------------+---------------------------+--------+-------------+------
--------+---------------+
| primary_cluster     | /volumes/_nogroup/ungrouped_subvol | idle   |            1
|             0 |             0 |
| primary_cluster     | /volumes/subvol_group/subvol       | idle   |            1
|             0 |             0 |
| primary_cluster     | /path/to/directory                 | idle   |            1
|             0 |             0 |
+---------------------+---------------------------+--------+-------------+-------
--------+---------------+
```

The status shows that there are three resources in the volume (vol), all with one snapshot synced to the configured remotes.

Disable CephFS replication

In some use-cases (say migration), the operator may want to disable replication for a given resource.

For subvolumes

Disable replication for a subvolume resource, here `vol` is the parent volume, `<subvolumegroup>` is the parent subvolume group and `<subvolume>` is the subvolume.

```
user@host:~$

none

sudo microceph replication disable cephfs --volume vol --subvolumegroup
<subvolumegroup> --subvolume <subvolume>
```

Omit `subvolumegroup` if the subvolume does not belong to any group.

For directory paths

Disable replication for a directory resource, here `</path/to/directory>` is the directory path relative to the root of volume `vol`.

```
user@host:~$

none

sudo microceph replication disable cephfs --volume vol --dir-path </path/to/
directory>
```

For `all enabled resources` in a volume

Disabling all resources in a volume is supported but requires the operator to pass the `--force` flag to avoid accidental disablement.

```
user@host:~$

none

sudo microceph replication disable cephfs --volume vol --force
```

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 our *guide on importing a remote cluster* (page 34).
3. RBD replication is configured for at least one RBD image. Refer to *Configure RBD replication* (page 35).

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 |
```

(continues on next page)

(continued from previous page)

```
| 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.

Upgrading your cluster

Follow these steps carefully to perform a major upgrade.

Major Upgrades

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.

Then, review the *release notes* (page 70) to check for any version-specific information. Also consult the upstream Ceph release notes[24] for the target version, and check the Ubuntu release notes[25] for any relevant 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.

[24] https://docs.ceph.com/en/latest/releases/
[25] https://documentation.ubuntu.com/release-notes/

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.

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

$ 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 \
```

(continues on next page)

(continued from previous page)

```
    --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)
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)
```

(continues on next page)

(continued from previous page)

```
    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.

RBD features and older clients

MicroCeph sets the default RBD features to 63 which corresponds to `layering + exclusive-lock + object-map + fast-diff + deep-flatten + stripingv2`. This is done to ensure better performance and features for the RBD images. However, older kernels might not support all these features.

If you face issues mapping RBD images on older kernels, you can either disable the unsupported features for the specific image, change the default behaviour for the cluster, or for a specific pool.

To disable unsupported features for a specific image:

```
$ sudo rbd feature disable block_pool/bd_foo object-map fast-diff deep-flatten
```

To change the default behaviour for the cluster, update the `rbd_default_features` config option with:

```
$ sudo ceph config set global rbd_default_features <value>
```

For example, to set it to only `layering` (value 1), run:

```
$ sudo ceph config set global rbd_default_features 1
```

It's also possible to set the default features on a per-pool basis to tailor the level of compatibility needed:

```
$ sudo rbd config pool set <pool_name> rbd_default_features <value>
```

Refer to the upstream documentation on RBD image features[26] for more details on RBD features and their bitmask values.

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.

[26] https://docs.ceph.com/en/latest/rbd/rbd-config-ref/#image-features

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
$ 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
    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.

Contact us

Report security vulnerabilities

Important:

Please don't open a public GitHub issue for security problems.

The easiest way to report a security issue is through our security advisory form on GitHub[27]. See GitHub's guide on privately reporting a security vulnerability[28] for instructions.

The repository admins will be notified of the issue and will work with you to determine whether the issue qualifies as a security issue and, if so, in which component. We will then handle figuring out a fix, getting a CVE assigned and coordinating the release of the fix.

The Ubuntu Security disclosure and embargo policy[29] contains more information about what you can expect when you contact us, and what we expect from you.

2.3.3. Reference

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

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.

`certificate`

Manage SSL certificates for MicroCeph services.

Usage:

```
microceph certificate [command]
```

Available commands:

```
set         Set SSL certificates for services
```

Global flags:

[27] https://github.com/canonical/snap-openstack/security/advisories/new

[28] https://docs.github.com/en/code-security/security-advisories/guidance-on-reporting-and-writing/privately-reporting-a-security-vulnerability

[29] https://ubuntu.com/security/disclosure-policy

```
-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

Set SSL certificates for services.

Usage:

```
microceph certificate set [command]
```

Available commands:

```
rgw         Set the SSL certificate for the RGW service
```

set rgw

Set or rotate SSL certificates for the RGW service. The new certificate and key are written to disk. Use `--restart` to restart the RGW service and pick up the new certificate immediately. Without `--restart`, the certificate is stored but the service must be restarted manually for the change to take effect.

Usage:

```
microceph certificate set rgw --ssl-certificate <base64> --ssl-private-key
<base64> [--target <server>] [--restart] [flags]
```

Flags:

```
--ssl-certificate string   base64 encoded SSL certificate (required)
--ssl-private-key string   base64 encoded SSL private key (required)
--target string            Server hostname (default: this server)
--restart                  Restart the RGW service for immediate certificate
pickup
```

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:

```
--availability-zone string Availability zone for failure domain distribution.
--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:

```
--availability-zone string Availability zone for failure domain distribution.
--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:

```
nfs         Disable the NFS Ganesha service on the --target server (default: this
server)
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
```

nfs

Disables the NFS Ganesha service on the –target server (default: this server).

Usage:

```
microceph disable nfs --cluster-id <cluster-id> [--target <server>] [flags]
```

Flags:

```
--cluster-id string   NFS Cluster ID (must match regex: '^[\w][\w.-]{1,61}[\w]$')
--target string       Server hostname (default: this server)
```

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 absolute paths, e.g. "/dev/sda /dev/sdb …". You may also specify WAL and DB devices referred to by absolute paths. However when specifying WAL and DB devices you may only add a single 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-match string       DSL expression to match backing devices for DB partitions
--db-size string        Requested DB partition size for --db-match
--db-wipe               Wipe the DB device prior to use
--dry-run               Show matched devices without adding them (requires --osd-
match)
--encrypt               Encrypt the disk prior to use (only block devices)
--json                  Provide dry-run output as a JSON-encoded DiskAddResponse
--osd-match string      DSL expression to match devices for OSD creation
--wal-device string     The device used for WAL
--wal-encrypt           Encrypt the WAL device prior to use
--wal-match string      DSL expression to match backing devices for WAL partitions
--wal-size string       Requested WAL partition size for --wal-match
--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.

DSL-based device selection

The `--osd-match` flag allows selecting devices using a DSL expression based on device attributes. This is useful for automation scenarios where device names may vary between nodes but functionally similar devices are present.

Example expressions:

```
# Select all NVMe devices
microceph disk add --osd-match "eq(@type, 'nvme')"

# Select devices larger than 100GiB
microceph disk add --osd-match "gt(@size, 100GiB)"

# Complex selection: NVMe devices from Samsung
microceph disk add --osd-match "and(eq(@type, 'nvme'), re('samsung', @vendor))"

# Preview matches without adding
microceph disk add --osd-match "eq(@type, 'sata')" --dry-run

# Select WAL/DB carriers separately and control their wipe/encryption
independently
microceph disk add --osd-match "eq(@type, 'ssd')" --encrypt \
  --wal-match "eq(@type, 'nvme')" --wal-size 1GiB --wal-encrypt --wal-wipe \
  --db-match "eq(@type, 'sata')" --db-size 4GiB --db-encrypt --db-wipe
```

Dry-run planning output

When `--dry-run` is used with `--osd-match` only, MicroCeph prints the OSD devices that would be added.

When `--dry-run` is used together with WAL and/or DB matching, MicroCeph prints a provisioning plan table with the selected OSDs, the WAL/DB carrier paths, the planned partition numbers and sizes, and two additional columns: `WAL ACTION` and `DB ACTION`.

When `--dry-run --json` is used, MicroCeph prints the underlying `DiskAddResponse` document directly instead of a human-formatted table. This machine-readable output is intended for shell automation and behaviour tests. The JSON payload keeps the same fields used by the API under `metadata`: `validation_error`, `warnings`, `dry_run_devices`, and `dry_run_plan`. Each `dry_run_plan` entry contains the selected `osd_path` and optional nested `wal`/`db` objects with `kind`, `parent_path`, `partition`, `size`, and `reset_before_use`.

The action column values mean:

- `new`: create the first auxiliary partition on a clean carrier
- `append`: add another partition on a carrier already used by the current cluster
- `reset`: wipe/reset the carrier before creating the planned partition(s)

A `reset` action is shown when `--wal-wipe` or `--db-wipe` allows a matched carrier to be reclaimed before partitioning, for example when the disk already contains foreign data or a foreign partition table. In these cases, `--dry-run` also emits an explicit warning naming each carrier that would be wiped/reset before partitioning.

Available predicates:

- `and(a, b, ...)` - Logical AND (variadic)
- `or(a, b, ...)` - Logical OR (variadic)
- `not(a)` - Logical NOT
- `in(x, a, b, ...)` - True if x equals any of the other arguments
- `re(pattern, value)` - Regex match (Go RE2 syntax, case-insensitive)
- `eq(a, b)` - Equality
- `ne(a, b)` - Not equal
- `gt(a, b)`, `ge(a, b)`, `lt(a, b)`, `le(a, b)` - Comparisons

Available variables:

- `@type` - Device type (sata, nvme, virtio, etc.)
- `@vendor` - Vendor name extracted from model (lowercased)
- `@model` - Full model string (lowercased)
- `@size` - Device size in bytes (compare with units like 100GiB, 500MB)
- `@devnode` - Kernel device node path (e.g., /dev/sda, /dev/nvme0n1)
- `@host` - Short hostname

Size units: B, KiB, MiB, GiB, TiB, PiB (1024-based) or KB, MB, GB, TB, PB (1000-based). Numbers and units must be written without any space between them (e.g., `100GiB`, not `100 GiB`)

Limitations:

- `--osd-match` cannot be used together with `--wal-device` or `--db-device`.
- `--wal-encrypt` and `--wal-wipe` require `--wal-match` when using DSL-based selection.
- `--db-encrypt` and `--db-wipe` require `--db-match` when using DSL-based selection.
- `--wal-match` and `--db-match` must resolve to disjoint device sets.

`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)
nfs         Enable the NFS Ganesha 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)
```

nfs

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

Usage:

```
microceph enable nfs --cluster-id <cluster-id> [--bind-address <ip-address>] [--
bind-port <port-num>] [--v4-min-version 0/1/2] [--target <server>] [--wait <bool>]
[flags]
```

Flags:

```
--bind-address string   Bind address to use for the NFS Ganesha service (default
"0.0.0.0")
--bind-port uint        Bind port to use for the NFS Ganesha service (default
2049)
--cluster-id string     NFS Cluster ID (must match regex: '^[\w][\w.-]{1,61}[\w]$
')
--target string         Server hostname (default: this server)
--v4-min-version uint   Minimum supported version (default 1)
--wait                  Wait for nfs 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)
```

(continues on next page)

(continued from previous page)

```
--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

Manage replication to remote clusters

Usage:

```
microceph replication [command]
```

Available commands:

```
configure   configure replication parameters
demote      Demote a primary cluster to non-primary status
disable     Disable replication
enable      Enable replication
list        List all resources configured for replication.
promote     Promote a non-primary cluster to primary status
status      Show resource replication status
```

replication enable

Enable replication for a workload

Usage:

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

Available commands:

```
cephfs      Enable replication for CephFS resource (Directory or Subvolume)
rbd         Enable replication for RBD resource (Pool or Image)
```

replication enable rbd

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")
```

replication enable cephfs

Enable replication for CephFS resource (Directory or Subvolume)

Usage:

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

Flags:

```
--dir-path string         Directory path relative to file system
--remote string           remote MicroCeph cluster name
--subvolume string        CephFS Subvolume
--subvolumegroup string   CephFS Subvolume Group
--volume string           CephFS volume (aka file-system)
```

replication status

Show resource replication status

Usage:

```
microceph replication status [command]
```

Available Commands:

```
cephfs  Show CephFS resource replication status
rbd     Show RBD resource replication status
```

replication status rbd

Show RBD resource replication status

Usage:

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

Flags:

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

replication status cephfs

Show CephFS resource replication status

Usage:

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

Flags:

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

replication list

List all configured remotes replication pairs.

Usage:

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

Available Commands:

```
cephfs  List all CephFS resources configured for replication
rbd     List all RBD resources configured for replication
```

replication list rbd

List all RBD resources configured for replication

Usage:

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

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

replication list cephfs

List all CephFS resources configured for replication

Usage:

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

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

replication disable

Disable replication for a workload

Usage:

```
microceph replication disable [command]
```

Available Commands:

```
cephfs      Disable replication for CephFS resource (Directory or Subvolume)
rbd         Disable replication for RBD resource (Pool or Image)
```

replication disable rbd

Disable replication for RBD resource

Usage:

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

Flags:

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

replication disable cephfs

Disable replication for CephFS resource

Usage:

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

```
--dir-path string         Directory path relative to file system
--force                   forcefully disable replication for resource
--subvolume string        CephFS Subvolume
--subvolumegroup string   CephFS Subvolume Group
--volume string           CephFS volume (aka file-system)
```

replication configure

Configure replication parameters

Usage:

```
microceph replication configure [command]
```

Available Commands:

… code-block:: none

```
rbd Configure RBD replication parameters
```

replication configure rbd

Configure replication parameters for RBD resource

Usage:

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

Flags:

```
--schedule string   snapshot schedule in days, hours, or minutes using d, h, m
suffix respectively
```

replication promote

Promote a non-primary cluster to primary status

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

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

replication demote

Demote a primary cluster to non-primary status

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
```

`waitready`

Waits until the MicroCeph daemon is ready and Ceph is operational.

This command first waits for the MicroCeph daemon (microcluster) to become available, then polls the Ceph monitor until `ceph -s` succeeds. It is useful in scripting and CI environments where subsequent commands should not run until the cluster is fully ready to accept operations.

With the `--storage` flag, it additionally waits until enough OSDs are up to satisfy pool replication requirements. The required number of OSDs is determined by the maximum `size` across all existing pools. If no pools exist yet, the `osd_pool_default_size` configuration value is used as a fallback.

Usage:

```
microceph waitready [flags]
```

Flags:

```
--storage   Wait until enough OSDs are up to satisfy pool replication requirements
--timeout   Number of seconds to wait before giving up (0 = indefinitely)
```

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
```

Example

Wait for daemon, Ceph, and storage readiness with a 60s timeout:

```
sudo microceph waitready --storage --timeout 60
```

Release Notes

The release notes section provides details on major MicroCeph releases.

Release notes

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

MicroCeph Tentacle

The Ceph team is happy to announce the release of MicroCeph v20 (tentacle). This is the first stable release in the Tentacle series of releases.

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

Highlights

- Uses Ceph 20.2.0 (tentacle)
- Built on Ubuntu 24.04 (`core24` snap base)
- Upgraded to microcluster v3
- Log rotation for the `logs/` directory via `logrotate`
- Reference architecture documentation
- Consolidated charm-microceph and MicroCeph documentation
- Includes all features and fixes from the squid stable cycle

Important changes

The snap is now built on the `core24` base. Hosts must be running an Ubuntu release that supports `core24` snaps.

The microcluster dependency has been upgraded from v2 to v3. This is an internal change but affects the underlying cluster database schema.

Known issues

Upgrades from quincy directly to tentacle are not supported. Upgrade to squid first, then to tentacle.

Erasure-coded pool users are advised that the upstream Ceph developers identified a bug where OSDs crash when `allow_ec_optimizations` is set on a pool, regardless of the `allow_ec_overwrites` setting, making the cluster unusable. Read Tracker Issue 74813[30] before enabling `allow_ec_optimizations` on Ceph 20.2.0.

List of pull requests

- #718[31]: test: upgrade squid to tentacle
- #714[32]: feat: add log rotation for the logs directory
- #713[33]: docs: Add reference architecture
- #707[34]: docs: consolidate MicroCeph charm documentation with MicroCeph docs
- #706[35]: feat: move to MicroCeph Tentacle, upgrade cluster library to v3, and build on core 24
- #698[36]: ci: add weekly health report workflow

[30] https://tracker.ceph.com/issues/74813
[31] https://github.com/canonical/microceph/pull/718
[32] https://github.com/canonical/microceph/pull/714
[33] https://github.com/canonical/microceph/pull/713
[34] https://github.com/canonical/microceph/pull/707
[35] https://github.com/canonical/microceph/pull/706
[36] https://github.com/canonical/microceph/pull/698

MicroCeph Squid

The Ceph team is happy to announce the release of MicroCeph v19 (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
- CephFS remote replication (enable/disable, status, listing)
- NFS service support via Ceph NFS Ganesha
- Adopt existing (non-MicroCeph) Ceph clusters into MicroCeph management
- Availability Zone support for OSDs
- Cluster maintenance mode with monitor-quorum protection
- MicroCeph orchestrator module shipped in the snap
- DSL-based device matching for OSD, WAL, and DB selection
- Support for modifying RGW SSL certificates at runtime
- `microceph waitready` command to verify cluster readiness
- `stripingv2` enabled by default in the RBD feature set
- 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.

Monitors are now enforced to use the `v2` (msgr2) protocol. Clients that only support `v1` will not be able to connect.

The joiner address is now auto-detected from the join token peers when running `microceph cluster join`; manual address overrides remain supported.

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

Squid stable updates (post-v19.2.0):

- #711[37]: fix: update Go module dependencies
- #710[38]: fix: auto-detect joiner address from join token peers
- #708[39]: fix: resolve references to stale paths
- #703[40]: fix: increase disk operation timeout
- #702[41]: fix: resolve monitor refresh loop
- #700[42]: fix: resolve all Go static check failures and drop the previous linter
- #699[43]: feat: add support for declarative WAL and DB device usage with execution, cleanup, and validation
- #697[44]: feat: add support for modifying the RGW SSL certificate
- #696[45]: fix: wait for RBD mirror health before testing disable operations
- #695[46]: ci: cache the built snap between jobs
- #691[47]: fix: re-enable services after a snap disable and enable cycle
- #688[48]: docs: add a database schema update guide to the developer docs
- #687[49]: feat: add Availability Zone support
- #684[50]: refactor: maintenance mode quality improvements
- #683[51]: feat: add a wait-ready command to verify the cluster is ready
- #682[52]: docs: fix a documentation link
- #681[53]: fix: resolve “no disks present” error when adding all disks

[37] https://github.com/canonical/microceph/pull/711
[38] https://github.com/canonical/microceph/pull/710
[39] https://github.com/canonical/microceph/pull/708
[40] https://github.com/canonical/microceph/pull/703
[41] https://github.com/canonical/microceph/pull/702
[42] https://github.com/canonical/microceph/pull/700
[43] https://github.com/canonical/microceph/pull/699
[44] https://github.com/canonical/microceph/pull/697
[45] https://github.com/canonical/microceph/pull/696
[46] https://github.com/canonical/microceph/pull/695
[47] https://github.com/canonical/microceph/pull/691
[48] https://github.com/canonical/microceph/pull/688
[49] https://github.com/canonical/microceph/pull/687
[50] https://github.com/canonical/microceph/pull/684
[51] https://github.com/canonical/microceph/pull/683
[52] https://github.com/canonical/microceph/pull/682
[53] https://github.com/canonical/microceph/pull/681

- #680[54]: fix: resolve missing unlock of encrypted WAL and DB at OSD start
- #679[55]: feat: close inactive issues automatically
- #677[56]: ci: avoid building Sphinx from source
- #676[57]: fix: resolve unexpected loop device behaviour
- #672[58]: fix: make device-node matching conform to the device DSL spec
- #668[59]: feat: add declarative device matching for OSD selection
- #661[60]: tests: functional test helper housekeeping
- #659[61]: fix: amend the command parameter
- #657[62]: fix: multi-monitor adopt bootstrap
- #656[63]: fix: add content attributes for content plugs
- #650[64]: feat: add v2 striping to the default RBD feature set
- #646[65]: feat: add a format flag to cluster list
- #643[66]: docs: add HTML meta descriptions
- #642[67]: docs: add a how-to document for MicroCeph CephFS replication
- #641[68]: docs: use a ref target for the cluster network how-to
- #638[69]: docs: refine the get-started tutorial
- #637[70]: docs: add remote replication explanations
- #635[71]: fix: pin dqlite to the LTS release
- #633[72]: docs: create a redirect for a renamed file
- #632[73]: docs: split up the security overview
- #631[74]: docs: add a how-to for reporting security issues
- #630[75]: docs: split up the full-disk encryption documentation

[54] https://github.com/canonical/microceph/pull/680
[55] https://github.com/canonical/microceph/pull/679
[56] https://github.com/canonical/microceph/pull/677
[57] https://github.com/canonical/microceph/pull/676
[58] https://github.com/canonical/microceph/pull/672
[59] https://github.com/canonical/microceph/pull/668
[60] https://github.com/canonical/microceph/pull/661
[61] https://github.com/canonical/microceph/pull/659
[62] https://github.com/canonical/microceph/pull/657
[63] https://github.com/canonical/microceph/pull/656
[64] https://github.com/canonical/microceph/pull/650
[65] https://github.com/canonical/microceph/pull/646
[66] https://github.com/canonical/microceph/pull/643
[67] https://github.com/canonical/microceph/pull/642
[68] https://github.com/canonical/microceph/pull/641
[69] https://github.com/canonical/microceph/pull/638
[70] https://github.com/canonical/microceph/pull/637
[71] https://github.com/canonical/microceph/pull/635
[72] https://github.com/canonical/microceph/pull/633
[73] https://github.com/canonical/microceph/pull/632
[74] https://github.com/canonical/microceph/pull/631
[75] https://github.com/canonical/microceph/pull/630

- #628[76]: feat: add support for enabling and disabling CephFS replication
- #627[77]: docs: fix a documentation title
- #626[78]: docs: move the architecture documentation
- #625[79]: ci: increase wait time for OSDs
- #624[80]: ci: make the OSD check more robust
- #622[81]: feat: adopt existing Ceph clusters using MicroCeph
- #621[82]: fix: improve pristine disk check with `ceph-bluestore-tool` validation
- #619[83]: feat: expose useful Ceph tools
- #616[84]: fix: use `ceph-bluestore-tool` for wiping disks
- #607[85]: docs: correct command invocation in client config docs
- #606[86]: docs: fix section headings
- #604[87]: fix: implement structured logging with persistent configuration
- #601[88]: feat: add support for fetching CephFS mirroring status and lists to the replication framework
- #600[89]: fix: remove unnecessary references to the client from the command
- #599[90]: test: speed up tests
- #594[91]: fix: list virtio block disk devices
- #591[92]: refactor: move sub-process handling to a common package
- #590[93]: fix: check if disks are pristine before attempting to use them
- #588[94]: fix: add checks before adding OSD, WAL, or DB devices
- #585[95]: feat: create only one OSD pool for NFS Ganesha
- #584[96]: feat: add the MicroCeph orchestrator module to the snap build

[76] https://github.com/canonical/microceph/pull/628
[77] https://github.com/canonical/microceph/pull/627
[78] https://github.com/canonical/microceph/pull/626
[79] https://github.com/canonical/microceph/pull/625
[80] https://github.com/canonical/microceph/pull/624
[81] https://github.com/canonical/microceph/pull/622
[82] https://github.com/canonical/microceph/pull/621
[83] https://github.com/canonical/microceph/pull/619
[84] https://github.com/canonical/microceph/pull/616
[85] https://github.com/canonical/microceph/pull/607
[86] https://github.com/canonical/microceph/pull/606
[87] https://github.com/canonical/microceph/pull/604
[88] https://github.com/canonical/microceph/pull/601
[89] https://github.com/canonical/microceph/pull/600
[90] https://github.com/canonical/microceph/pull/599
[91] https://github.com/canonical/microceph/pull/594
[92] https://github.com/canonical/microceph/pull/591
[93] https://github.com/canonical/microceph/pull/590
[94] https://github.com/canonical/microceph/pull/588
[95] https://github.com/canonical/microceph/pull/585
[96] https://github.com/canonical/microceph/pull/584

- #583[97]: docs: update documentation to include information about enabling NFS
- #582[98]: refactor: add an OSD manager to improve testing
- #578[99]: feat: add CephFS mirror to the service placement interface
- #575[100]: fix: prevent enabling snapshot replication on RBD pools
- #574[101]: feat: add NFS support
- #573[102]: docs: update disk add documentation
- #572[103]: docs: migrate to the extension-based starter pack
- #567[104]: feat: enforce v2 for monitors
- #565[105]: feat: ensure a majority of monitor services remain available before entering maintenance mode
- #545[106]: docs: MicroCloud integration annotations

Initial v19.2.0 release:

- #467[107]: Fix: increase timings for osd release
- #466[108]: Adjust 'verify_health' iterations
- #464[109]: Test: upgrade update
- #463[110]: Fix: add python3-packaging
- #462[111]: Test: upgrade reef to local build
- #461[112]: Test: add reef to squid upgrade test
- #460[113]: Improve require-osd-release
- #459[114]: Set the 'require-osd-release' option on startup
- #458[115]: Updated readme.md
- #457[116]: Modify post-refresh hook to set OSD-release
- #456[117]: Make remote replication CLI conformant to CLI guidelines

[97] https://github.com/canonical/microceph/pull/583
[98] https://github.com/canonical/microceph/pull/582
[99] https://github.com/canonical/microceph/pull/578
[100] https://github.com/canonical/microceph/pull/575
[101] https://github.com/canonical/microceph/pull/574
[102] https://github.com/canonical/microceph/pull/573
[103] https://github.com/canonical/microceph/pull/572
[104] https://github.com/canonical/microceph/pull/567
[105] https://github.com/canonical/microceph/pull/565
[106] https://github.com/canonical/microceph/pull/545
[107] https://github.com/canonical/microceph/pull/467
[108] https://github.com/canonical/microceph/pull/466
[109] https://github.com/canonical/microceph/pull/464
[110] https://github.com/canonical/microceph/pull/463
[111] https://github.com/canonical/microceph/pull/462
[112] https://github.com/canonical/microceph/pull/461
[113] https://github.com/canonical/microceph/pull/460
[114] https://github.com/canonical/microceph/pull/459
[115] https://github.com/canonical/microceph/pull/458
[116] https://github.com/canonical/microceph/pull/457
[117] https://github.com/canonical/microceph/pull/456

- #454[118]: Pin LXD and use microcluster with dqlite LTS
- #447[119]: Update mods, build from noble
- #443[120]: Bootstrap: wait for daemon
- #441[121]: Build from noble-proposed
- #440[122]: Remove tutorial section
- #438[123]: MicroCeph Remote Replication (3/3): Site Failover/Failback
- #437[124]: MicroCeph Remote Replication (2/3): RBD Mirroring
- #433[125]: Docs: fix indexes
- #432[126]: Use square brackets around IPv6 in ceph.conf
- #430[127]: Adds support for RO cluster configs
- #429[128]: Move mounting CephFS shares tutorial to how-to section
- #428[129]: Move mounting RBD tutorial to how-to section
- #427[130]: Move multi-node tutorial to how-to section
- #426[131]: Move multi-node tutorial to how-to section
- #422[132]: Change tutorial landing page
- #419[133]: Change explanation landing page
- #418[134]: Add CephFS to wordlist
- #417[135]: Move MicroCeph charm to explanation section
- #416[136]: Fix reference landing page
- #415[137]: Move single-node tutorial to how-to section
- #409[138]: Fetch current OSD pool configuration over the API
- #407[139]: Add interfaces: rbd kernel module and support

[118] https://github.com/canonical/microceph/pull/454
[119] https://github.com/canonical/microceph/pull/447
[120] https://github.com/canonical/microceph/pull/443
[121] https://github.com/canonical/microceph/pull/441
[122] https://github.com/canonical/microceph/pull/440
[123] https://github.com/canonical/microceph/pull/438
[124] https://github.com/canonical/microceph/pull/437
[125] https://github.com/canonical/microceph/pull/433
[126] https://github.com/canonical/microceph/pull/432
[127] https://github.com/canonical/microceph/pull/430
[128] https://github.com/canonical/microceph/pull/429
[129] https://github.com/canonical/microceph/pull/428
[130] https://github.com/canonical/microceph/pull/427
[131] https://github.com/canonical/microceph/pull/426
[132] https://github.com/canonical/microceph/pull/422
[133] https://github.com/canonical/microceph/pull/419
[134] https://github.com/canonical/microceph/pull/418
[135] https://github.com/canonical/microceph/pull/417
[136] https://github.com/canonical/microceph/pull/416
[137] https://github.com/canonical/microceph/pull/415
[138] https://github.com/canonical/microceph/pull/409
[139] https://github.com/canonical/microceph/pull/407

- #405[140]: MicroCeph Remote Replication (1/3): Remote Awareness
- #401[141]: doc: remove `woke-install` as prereq for building the docs
- #400[142]: doc: remove `woke-install` as prereq for building the docs
- #398[143]: MicroCeph Remote Replication (2/3): RBD Mirroring
- #395[144]: Use LTS microcluster

Canonical Ceph hardware recommendations

Our hardware recommendations section contains hardware specification recommendations for Canonical Ceph clusters, basing these specifications on cluster service placement strategy. It also includes recommendations for infrastructure node requirements depending on the method of deployment, i.e. via charms or snap.

Canonical Ceph hardware recommendations

This section outlines Canonical Ceph hardware specification recommendations. Following these configurations is highly recommended as a proven way to achieve the best price-performance.

Caution:

Deviations from these specifications are possible, but they can affect both the overall storage performance and its total cost of ownership (TCO). We strongly recommend consulting Canonical before making any changes to the reference configurations and prior to purchasing any hardware.

Canonical provides different specifications based on cluster service placement architecture, i.e. hyperconverged or disaggregated. For hyperconverged nodes, we base our specifications on performance, capacity and general-purpose use cases. For disaggregated scenarios, we recommend specifications based on several combinations as per customer requirements, e.g. dedicated control plane nodes, dedicated RADOS Gateway (RGW) nodes, and dedicated Metadata server (MDS) nodes.

The Canonical Ceph hardware recommendation section provides infrastructure node recommendations for the different ways of deploying Canonical Ceph, i.e. via snap or charms. It also outlines memory and processor requirements, and provides software recommendations.

Deployment strategies

Hyperconverged architecture

In the context of hyperconverged nodes, **a minimum of six nodes is required** to be eligible for Ubuntu Pro Managed Solutions and Delivery services with Canonical.

We'll provide three reference specifications based on the purpose of the storage environment, i.e. performance, capacity and general-purpose storage.

[140] https://github.com/canonical/microceph/pull/405
[141] https://github.com/canonical/microceph/pull/401
[142] https://github.com/canonical/microceph/pull/400
[143] https://github.com/canonical/microceph/pull/398
[144] https://github.com/canonical/microceph/pull/395

Performance

In the trade-off between cost and performance, performance-oriented nodes prioritise low latency, high throughput, and high concurrency.

Processor	56 cores (x86) or greater
Memory	256 GB RAM
Network	2x dual-port NICs (25 Gb+), onboard NIC for OOB
Storage	2x 960 GB SSD/NVMe for OS; 8x enterprise NVMe for OSDs

Capacity

Available storage disk capacity (what users will see).

Processor	32 cores (x86)
Memory	256 GB RAM
Network	2x dual-port NICs (25 Gb+), onboard NIC for OOB
Storage	2x 960 GB SSD/NVMe for OS; 4x enterprise SSD/NVMe for RadosGW metadata OSDs (4% NL-SAS capacity) WAL/DB*; 20x NL-SAS or SSD for OSDs

Note:

Separate WAL/DB or bcache disks should only be purchased if there is a significant difference in performance between the OSD disks (slow disks, i.e. “Very Read Intensive SSDs”) and the dedicated WAL/DB disks (fast disks).

General-purpose storage

A balanced approach between available capacity and performance.

Processor	64 cores (x86) or greater
Memory	256 GB RAM
Network	2x dual-port NICs (25 Gb+), onboard NIC for OOB
Storage	2x 960 GB SSD/NVMe for OS; 20x enterprise SSD/NVMe for OSDs

Disaggregated architecture

In the context of disaggregated nodes, **a minimum of nine nodes** is required to be eligible for Ubuntu Pro Managed Solutions and Delivery services with Canonical.

The initial configuration suggestion would be three nodes for the control plane: Ceph Monitors (MONs), Ceph RADOS Gateway (RGW), and Ceph metadata servers (MDSs), and six nodes for dedicated object storage daemons (OSDs). However, the distribution needs to be adapted to your requirements, and allows for several combinations:

Dedicated control plane nodes

In some cases, customers prefer to have a set of nodes dedicated to control plane services, with the Ceph OSDs separated. The control plane nodes would host the Ceph MON service. It is preferred to host the RGW or MDS services on the Ceph OSD nodes, so that the RGW and MDS services scale out with the increase of the storage cluster itself.

Considering a generic use case, the specifications below are recommended:

Processor	16 cores (x86) or greater
Memory	32 GB RAM
Network	2x dual-port NICs (25 Gb+), onboard NIC for OOB
Storage	2x 960 GB SSD/NVMe for OS

If it is decided to host the RGW or MDS services on the control plane nodes, Canonical recommends increasing the specifications of the control plane nodes with the recommendations listed under the *Dedicated RGW nodes* (page 80) and *Dedicated Metadata nodes* (page 80) sections.

Dedicated RGW nodes

Having dedicated RGW nodes is relevant in a **high-performance object storage Ceph cluster**. These nodes can also have the MON services collocated. An object storage Ceph cluster needs to scale the RGW nodes with the load, and it is common to see over 12 threads in use on a loaded RGW node.

In this case, the following specifications are recommended:

Processor	16 cores (x86) or greater
Memory	32 GB RAM
Network	2x dual-port NICs (25 Gb+), onboard NIC for OOB
Storage	2x 960 GB SSD/NVMe for OS

Dedicated Metadata nodes

Dedicated MDS nodes are relevant in a **high-performance CephFS cluster**. These nodes can also have the MON services collocated. MDS nodes use a single worker thread, so CPU speed matters significantly for performance.

For this case, we recommend these specifications:

Processor	12 cores (x86), highest GHz
Memory	128 GB RAM
Network	2x dual-port NICs (25 Gb+), onboard NIC for OOB
Storage	2x 960 GB SSD/NVMe for OS

Infrastructure node requirements

For either hyperconverged or disaggregated scenarios, refer to the *cluster service placement strategies* (page 109) outlined in the explanation section of our reference architecture. The nodes should have the following specifications:

Processor	24 cores
Memory	128 GB RAM
Network	1x 4-port 1 Gb onboard NIC for OOB
Storage	2x 6 TB SAS 3.5” SSD in RAID-1 mode, hardware RAID controller

Memory requirements

The overall amount of random access memory (RAM) is dictated by the number of disks and overall size of the cluster. Each service type has its minimum memory amount. When building a server, the total memory must be calculated based on the number of services collocated on it (e.g. 12 disks equals 12 OSDs, each requiring a minimum RAM amount), and additionally the operating system itself also needs to be considered.

The table below specifies memory requirements for different service types, and the Ubuntu OS:

OS/Ceph service	Memory requirements (GB)	Performance notes
Ubuntu	16	
MON	8	RAM consumption increases with cluster instability.
OSD	8	HDD requires a minimum of 4 GB, but will perform better at 8 GB. SSD and NVMe require more memory, consuming up to 16–24 GB per disk depending on tuning.
MDS	64	Up to 96 GB can improve performance; after which horizontal scaling is recommended.
RadosGW	4	Up to 32 GB can improve performance; after that, horizontal scaling is recommended.
NVMe-oF GW	8	8 GB minimum per daemon. Requires more memory proportional to the number of RBD images mapped and IOPS. 16+ GB is required for 200,000+ IOPS scenarios.

Note:

Recommendations will change with the size and workload of the cluster.

Software requirements

For the best experience, Canonical always recommends using the latest long-term support (LTS) version of Ubuntu, and an LTS version of your Canonical Ceph deployment option. We also recommend updating the software on a regular basis to benefit from new capabilities brought by the latest kernels, features, and bug fixes. This prevents issues where, for example, using legacy versions of the operating system (OS) or Ceph may result in certain hardware components or features not being supported.

This table provides official release information for each component of a production Canonical Ceph cluster deployment:

Component	Release cycle information
Ubuntu (Host OS)	Ubuntu releases[145]
Linux Kernel	Ubuntu Kernel release cycle[146]
Canonical Ceph	Charmed Ceph release notes[147], MicroCeph release notes[148], charm-microceph[149]
MAAS	MAAS release notes and upgrade instructions[150]
Juju	Juju release notes[151]

CPU recommendations

Different Ceph services have different CPU resource requirements (page 112), and so does the Ubuntu OS. Below is a summary table of the processor requirements per service type running on a node.

OS/Ceph service	CPU requirements (cores)	Performance notes
Ubuntu Operating System	4	
Ceph MON	4	Fast disk (SSD or better)
Ceph OSD	1 per HDD 2–4 per SATA SSD 4–8 per NVMe	1 core per 1,000–3,000 IOPS 1 core per 200–500 MB/s
MDS	4	Single-threaded service. High clock rate (GHz) preferred.
RadosGW	4–16	Can consume up to 16 cores for higher performance (>25 Gbit/s).
NVMe-oF GW	4–30	For a PoC with <10,000 IOPS, 4 cores are the minimum requirement. Values between 200,000 and 300,000 IOPS require 25–30+ cores.

2.3.4. 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.

Working with MicroCeph

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

Cluster network configurations

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

[145] https://releases.ubuntu.com/
[146] https://ubuntu.com/about/release-cycle/
[147] https://ubuntu.com/ceph/docs/release-notes
[148] https://canonical-microceph.readthedocs-hosted.com/latest/snap/reference/release-notes/
[149] https://charmhub.io/microceph
[150] https://canonical.com/maas/docs/release-notes-and-upgrade-instructions
[151] https://documentation.ubuntu.com/juju/latest/releasenotes/

and backfilling) utilises a separate net. This helps to prevent malicious or malfunctioning clients from disrupting cluster backend operations.

For more details, refer to the upstream network configuration reference[152].

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.

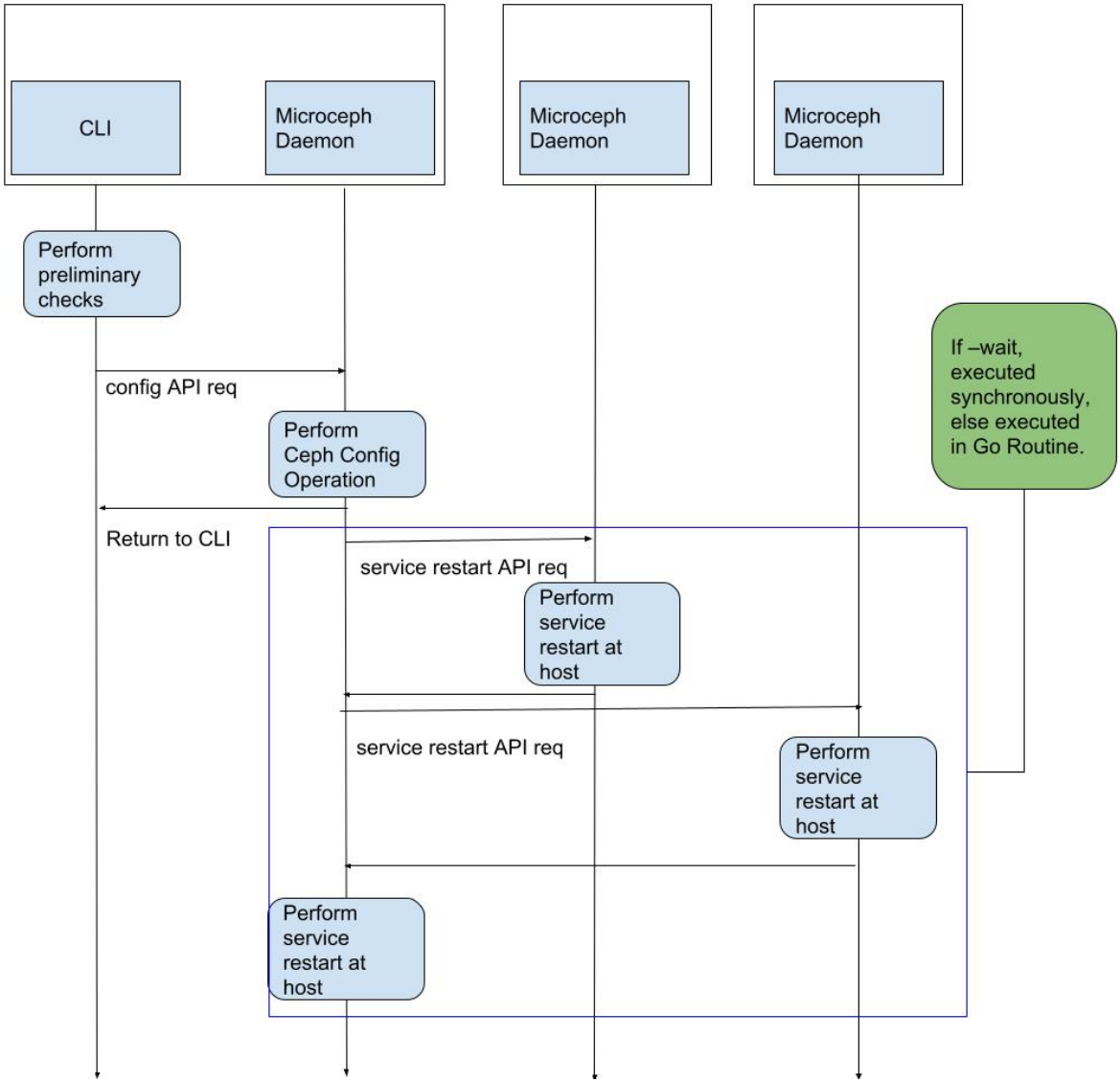


Fig. 3: Execution flow of config set/reset commands in multi-node MicroCeph deployment

[152] https://docs.ceph.com/en/latest/rados/configuration/network-config-ref/

Maintenance Mode

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 our *guide on performing cluster maintenance* (page 32).

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:

Enable 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 (majority of 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)

Disable 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

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[153] 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[154]) 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.

[153] https://en.wikipedia.org/wiki/Failure_domain
[154] https://docs.ceph.com/en/latest/rados/operations/crush-map/

Disk removal

The *disk command* (page 57) (**`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.

Prerequisites

Refer to our *guide on mounting MicroCeph-backed block devices* (page 43) for getting started with RBD.

Once you have a the block device mounted and in use, you can jump to Ceph RBD Snapshots[155].

CephFs snapshots

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

Prerequisites

Refer to our *guide on mounting MicroCeph-backed CephFs shares* (page 46).

Once you have a the filesystem mounted and in use, you can jump to CephFs Snapshots[156].

Remote replication

Cloud storage services (like Ceph) are responsible for persisting information irrespective of faults and failures in parts of the cluster. These faults could be temporary network failures, disk faults, power failures or even failure of multiple nodes. Remote replication is a mechanism used to replicate data to a remote storage cluster, typically located at a different geographical site to prevent complete outage in the event of a large enough fault like natural disaster.

This section covers the essential remote replication concepts for users.

Modes of data movement

Replication between clusters can be implemented in two common modes, each with specific operational characteristics:

[155] https://docs.ceph.com/en/latest/rbd/rbd-snapshot/
[156] https://docs.ceph.com/en/latest/dev/cephfs-snapshots/

Push replication

In this mode, the source cluster actively sends data updates (aka deltas or diffs) to the target cluster. The replication process is initiated and managed by the source cluster, ensuring changes are centrally propagated. This is easier to administer at smaller scale but can place higher resource demands on the primary cluster for larger clusters.

Pull replication

In this mode, the target cluster initiates and manages copying (or pulling) updates from the source. This model provide target sites the control over bandwidth and timing. It scales efficiently in large environments, although is slightly more complex to implement and administer.

Replication architectures

Based on cost, complexity, and recovery objectives, a choice can be made between these two architectures:

Active-Active replication

Both clusters handle read and write operations, synchronising changes in real time between them. This ensures high availability because if one site goes down, users can continue operation on the other with no data loss. However, in order to maintain simultaneous state consistency across active sites, each operation has to be acknowledged by each active site before being considered complete. This can introduce latency, especially for geographically distant sites. It is best for use cases requiring continuous operation and zero recovery point objective (*RPO* (page 89)).

Active-Passive replication

The active cluster is used to serve clients, while the passive cluster acts as a backup. Data is replicated asynchronously to the passive cluster, which becomes operational only during failover. This approach is suitable for Disaster Recovery (DR) in scenarios where the secondary site isn't needed for real-time access, accepting minor data delays after failover.

Disaster recovery objectives

It is important to set realistic objectives for recovery from service disruptions. Two key metrics are commonly used to plan for predictable restoration of services. These are as follows:

Recovery point objective (RPO)

RPO defines the maximum acceptable amount of data that can be lost in case of failure, typically expressed as a time interval.

- With synchronous replication, updates are mirrored instantly, resulting in zero data loss.
- With asynchronous replication, updates occur on a schedule, meaning any data written since the last successful replication may be lost during failover.

Recovery time objective (RTO)

RTO is the maximum acceptable amount of time that a system can be down after a disruption before significant damage or intolerable consequences occur.

For practical instructions on setting up replication, see:

- *Configure RBD replication* (page 35)
- *Configure CephFS mirroring* (page 37)
- *Perform site failover* (page 40)

The Snap content interface

Access MicroCeph's configuration and credentials.

Snap content interface for MicroCeph

Snap content interfaces[157] 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 Ceph Monitor (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.

[157] https://snapcraft.io/docs/reference/interfaces/content-interface/#interfaces-content-interface

The MicroCeph charm

The MicroCeph charm helps you deploy and manage your MicroCeph deployment with Juju[158].

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`).

To learn more, see the MicroCeph charm on Charmhub[159], the *MicroCeph charm tutorial* (page 118), and the *MicroCeph charm how-to guides* (page 121).

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.

MicroCeph architecture

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[160] 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. See the snap security confinement documentation[161] to learn more about snap security and isolation.
- microcephd: The core service (based on Microcluster[162]) 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[163]: Ceph Monitor (MON) daemon(s), maintaining cluster state and consensus.

[158] https://juju.is/
[159] https://charmhub.io/microceph
[160] https://canonical.com/dqlite/docs/
[161] https://snapcraft.io/docs/explanation/security/snap-confinement/#explanation-security-snap-confinement
[162] https://github.com/canonical/microcluster
[163] https://docs.ceph.com/en/latest/man/8/ceph-mon/

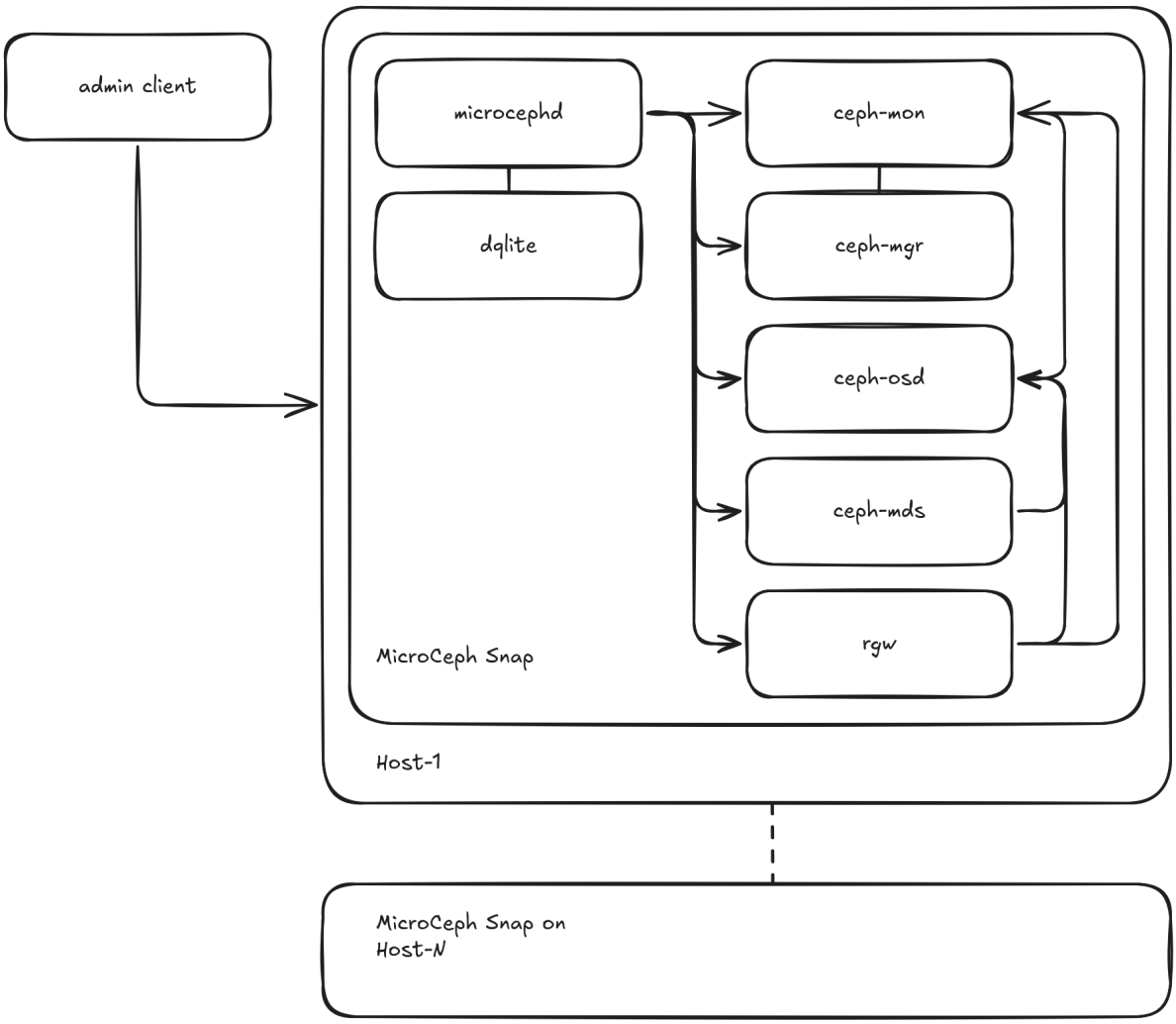


Fig. 4: MicroCeph Architecture Overview

  - ceph-mgr[164]: Ceph Manager (MGR) daemon(s), providing access to management APIs and modules like the Dashboard.
  - ceph-osd[165]: Ceph object storage daemons (OSDs), managing data on underlying storage devices.
  - ceph-radosgw[166] (optional): Ceph Object Gateway (RGW) service, providing S3/Swift-compatible object storage.
  - ceph-mds[167] (optional): Metadata Server (MDS) daemons for CephFS.
- Client Workloads: Consume Ceph storage via RBD block devices, RGW object buckets, or CephFS shared filesystems.

Security overview

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.

[164] https://docs.ceph.com/en/latest/mgr/
[165] https://docs.ceph.com/en/latest/man/8/ceph-osd/
[166] https://docs.ceph.com/en/latest/radosgw/
[167] https://docs.ceph.com/en/latest/man/8/ceph-mds/

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/MDS	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	microceph	Local API socket for microceph CLI	Access controlled by filesystem permissions on the socket file within the snap's data directory.
7443	microceph	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 Messenger protocol (v1/v2)[168]: 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.

[168] https://docs.ceph.com/en/latest/rados/configuration/msgr2/

- 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 the snap confinement documentation[169] for more details.
- Ceph Dashboard: If enabled, secure its access via network controls and strong authentication.

Best practices for 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 (see *cluster network configurations* (page 83) 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:

[169] https://snapcraft.io/docs/explanation/security/snap-confinement/#explanation-security-snap-confinement

  - 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* (page 93)).
  - 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`). See *changing the log level* (page 28) for how to set log levels. Ceph logs are found in `/var/snap/microceph/common/logs/ceph/`.
  - microcephd logs to `/var/log/syslog`, see *changing the log level* (page 28) 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/…``). See *enabling metrics collection with Prometheus* (page 21) for a full setup guide.
- 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).

Best practices for secure operation

Maintaining security is an ongoing process.

Vulnerability management

- Monitor advisories: Actively track CVEs and security advisories for:
  - Ceph (via the Ceph announce mailing list[170] and 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.

[170] https://lists.ceph.io/postorius/lists/ceph-announce.ceph.io/

- Schedule Proactively: Plan and test upgrades, especially for security vulnerabilities. Utilize snap channels for pre-production testing.

Release notes

- Always read the *MicroCeph release notes* (page 70), the upstream Ceph release notes[171], and the Ubuntu release notes[172] before upgrading or making significant changes, as they contain information about security fixes, new features, and potential issues.

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[173].

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.

[171] https://docs.ceph.com/en/latest/releases/
[172] https://documentation.ubuntu.com/release-notes/
[173] https://ubuntu.com/ceph/docs/cryptographic-technologies-in-charmed-ceph

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[174] 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).

[174] https://ubuntu.com/ceph/docs/cryptographic-technologies-in-charmed-ceph

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 in MicroCeph

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.

It's important to note that during operation of a host, the disk must be unlocked so that programs on the machine can access it. FDE does not protect data from exfiltration from a running system (for instance, in case of a malware infection).

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[175].

Limitations

- 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.

[175] https://gitlab.com/cryptsetup/cryptsetup/-/wikis/home

- As alluded to above, FDE protects data on disks. However while the host is running, this data will be made accessible to allow retrieval. This implies that if a malicious program were to run on the machine, it would also be able to access the data – FDE cannot protect against this scenario.

Usage

See our *guide on enabling FDE* (page 34) for MicroCeph OSDs. 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.

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.

Canonical Ceph reference architecture

Find detailed guidelines for optimising your Ceph architecture. Learn about the different deployment options; what to consider when designing a Canonical Ceph cluster; hardware and software specifications and recommendations, and information on Ubuntu certified hardware.

Canonical Ceph reference architecture

Canonical Ceph can be deployed as a stand-alone cluster, or integrated with OpenStack, VMware, or Kubernetes. Depending on your cluster use case, Canonical Ceph provides various deployment options[176]: single-node and multi-node deployments, through MicroCeph; large-scale deployments through Charmed Ceph, and containerized deployments with Ceph Rocks. A single cluster can provide block, file, and object storage, and can be customized to meet price-performance requirements.

Our reference architecture provides detailed guidelines for optimizing your Ceph architecture. It starts by detailing the different deployment options and their architectures, then highlights the factors to consider when designing a Canonical Ceph cluster, and then outlines hardware specifications, including compute, storage, memory and networking requirements. The *reference section of the reference architecture* (page 78) includes Canonical's recommendations regarding hardware and software configurations, to further inform your architectural choices.

Warning:

This reference architecture is a starting point, not a prescription. Before making any purchasing decisions, we recommend reviewing your plans with Canonical to ensure they align with your specific needs. Hardware requirements can vary significantly depending on your workload and goals.
Canonical is not liable or responsible for any equipment purchases made as a result of this reference architecture. Following these recommendations does not guarantee that the proposed hardware will meet the requirements of your project or use case.
To discuss your specific requirements, contact us[177].

[177] https://ubuntu.com/ceph/contact-us

Deployment options for Canonical Ceph

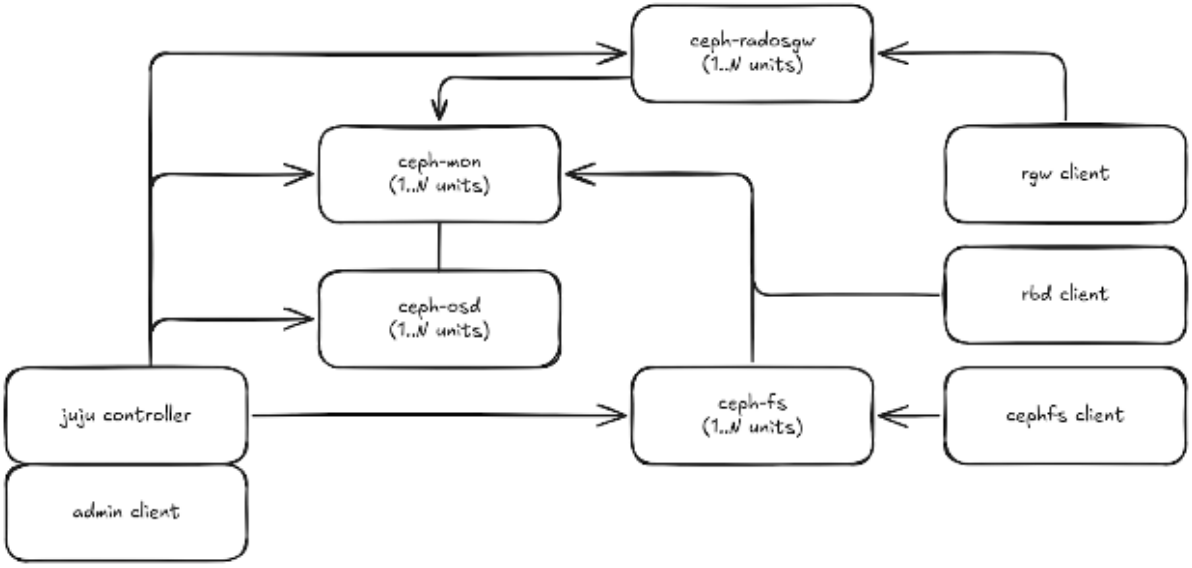
Canonical Ceph can be deployed using snaps or Juju charms, i.e. via MicroCeph (or charm-microceph), Charmed Ceph and Ceph Rocks. The choice of deployment method is informed by your storage cluster use case.

Charmed Ceph

Charmed Ceph[178] is the full-scale deployment option for private cloud infrastructure. It uses Juju and machine charms to deploy and manage Ceph clusters, providing comprehensive lifecycle management, configuration, and integration with other Juju-managed infrastructure. This is the appropriate choice for production private clouds where you want unified orchestration of storage alongside compute and networking.

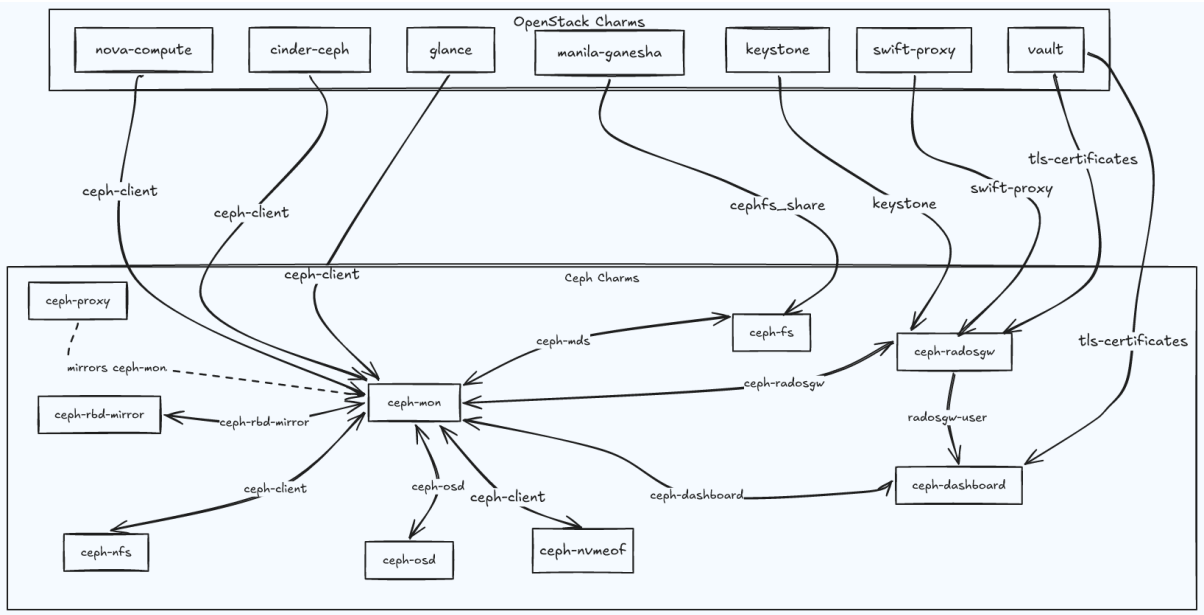
[176] https://ubuntu.com/ceph/install
[178] https://ubuntu.com/ceph/docs

The diagram below depicts a typical Charmed Ceph deployment with `ceph-radosgw` and `ceph-fs`. However, the Charmed Ceph ecosystem is flexible and can be tailored to a specific use case.



Learn more about the Charmed Ceph architecture[179] in the product documentation.

Charmed Ceph is often deployed with other products; the diagram below shows how Charmed Ceph is integrated with OpenStack OpenStack[180] charms.

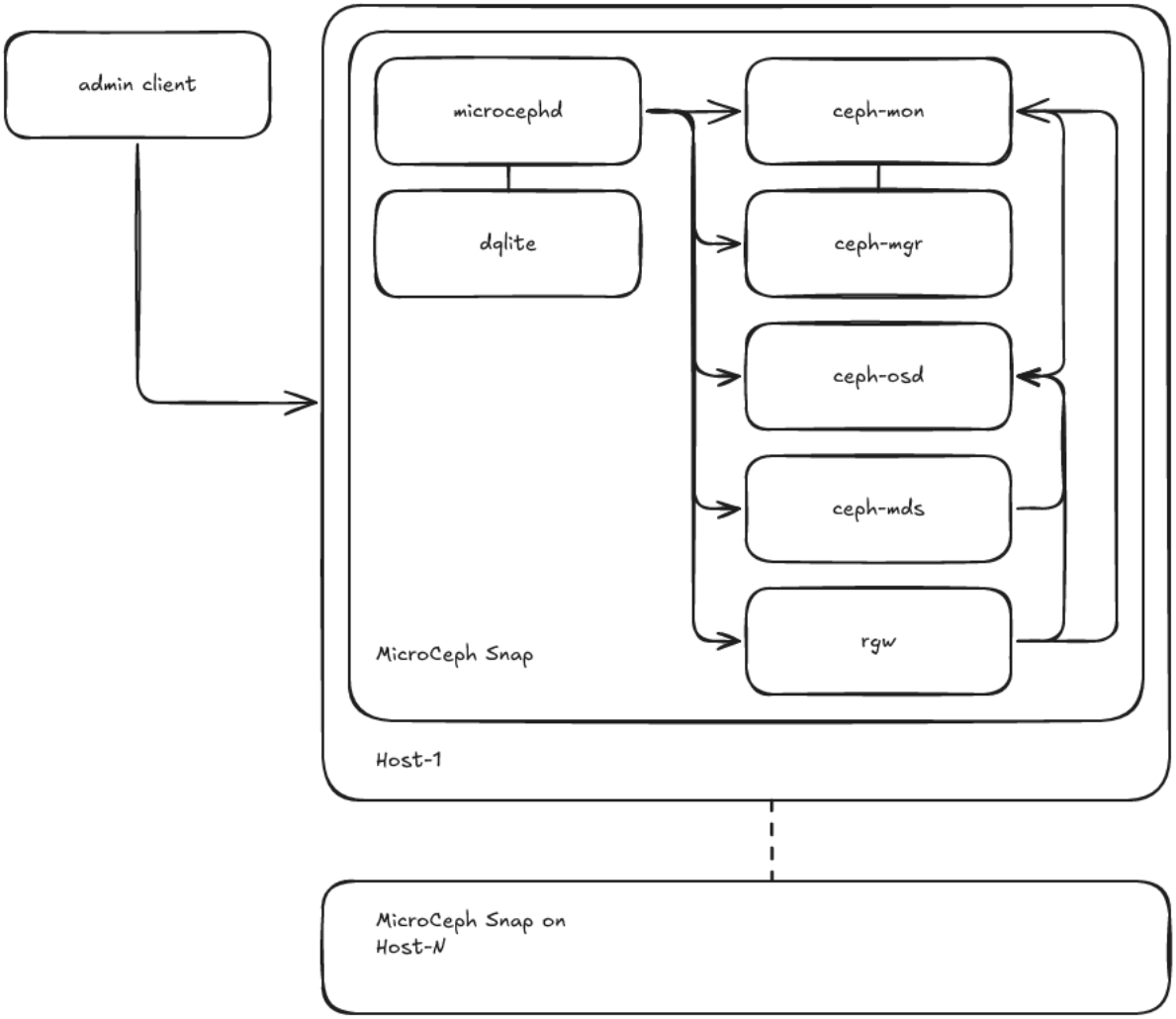


MicroCeph

MicroCeph[181] is a lightweight deployment option, packaged as a snap. It is designed for edge computing where minimal operational overhead matters, and for small-scale deployments like developer workstations, Continuous Integration (CI) environments, or training setups. The snap handles daemon lifecycle and upgrades with minimal configuration.

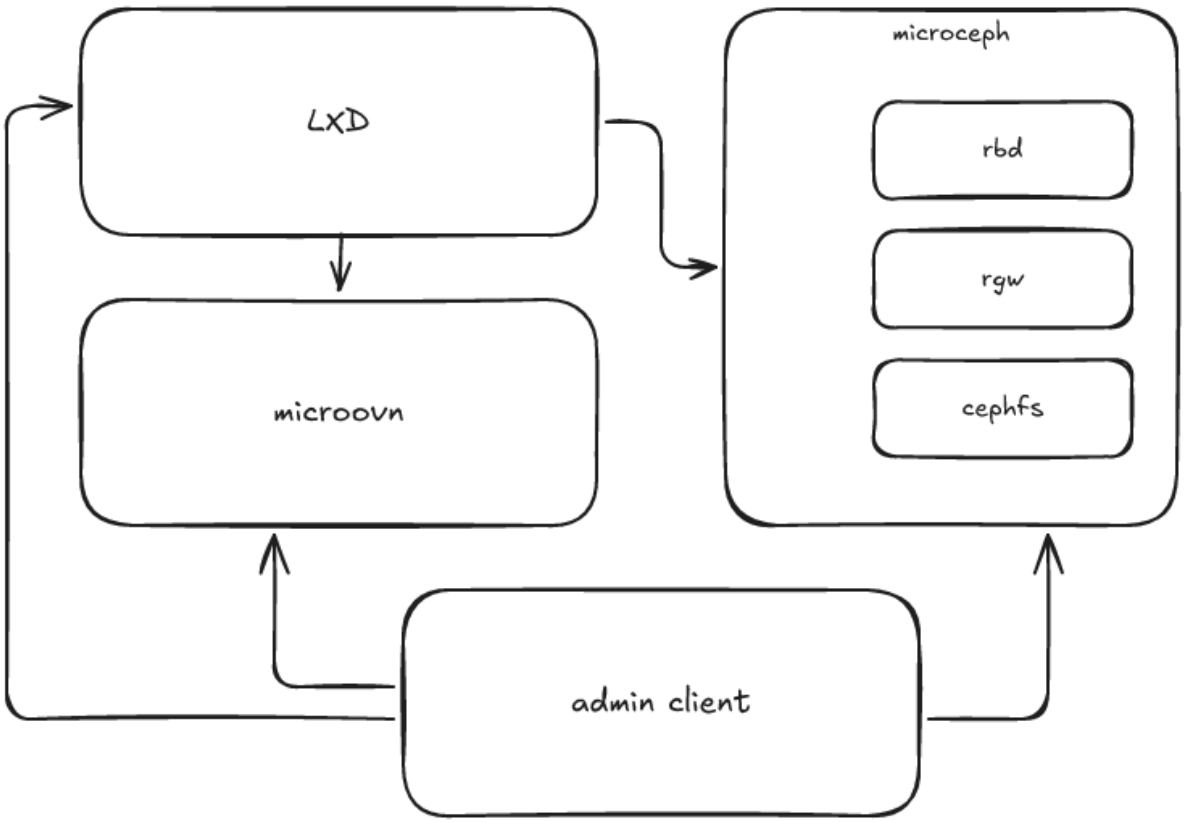
[179] https://ubuntu.com/ceph/docs/ceph-architecture
[180] https://canonical-openstack.readthedocs-hosted.com/en/latest/
[181] https://canonical-microceph.readthedocs-hosted.com/latest/

See an example of a MicroCeph cluster below.. The MicroCeph architecture section[182] provides more details about MicroCeph components.



MicroCeph is often deployed with other products, e.g. MicroCloud[183], as in this diagram:

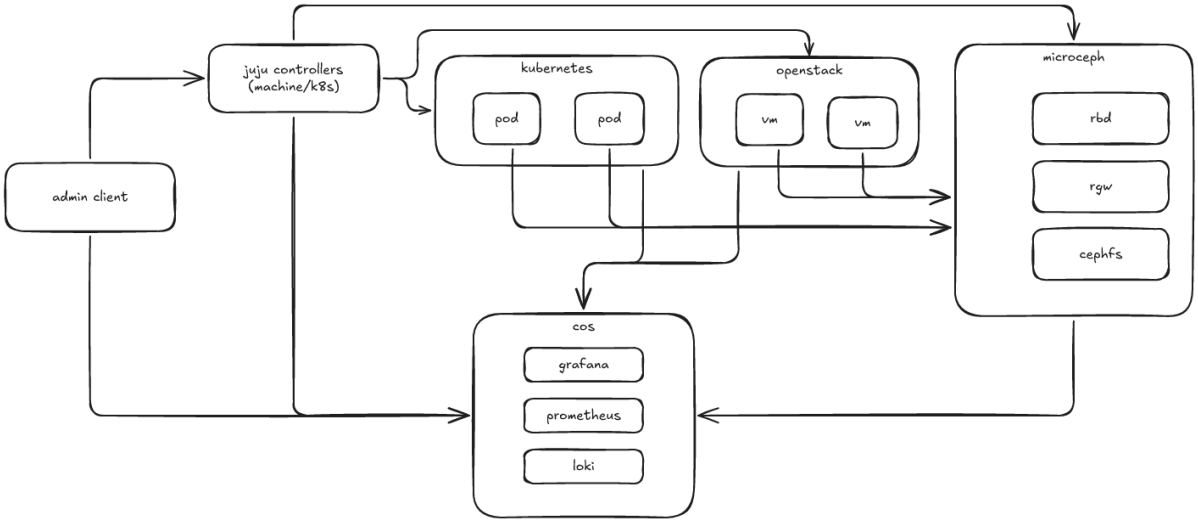
[182] https://canonical-microceph.readthedocs-hosted.com/latest/snap/explanation/microceph-architecture/
[183] https://documentation.ubuntu.com/microcloud/latest/



charm-microceph

charm-microceph[184] bridges MicroCeph and Juju. It deploys MicroCeph under the hood but exposes it as a Juju-managed application, giving you the lightweight footprint of MicroCeph while retaining compatibility with Juju's orchestration, relations, and model-driven operations. This fits edge deployments that are part of a larger Juju-managed estate.

Charm-microceph is often integrated with other product, e.g. the Canonical Observability Stack (COS), as in this diagram:

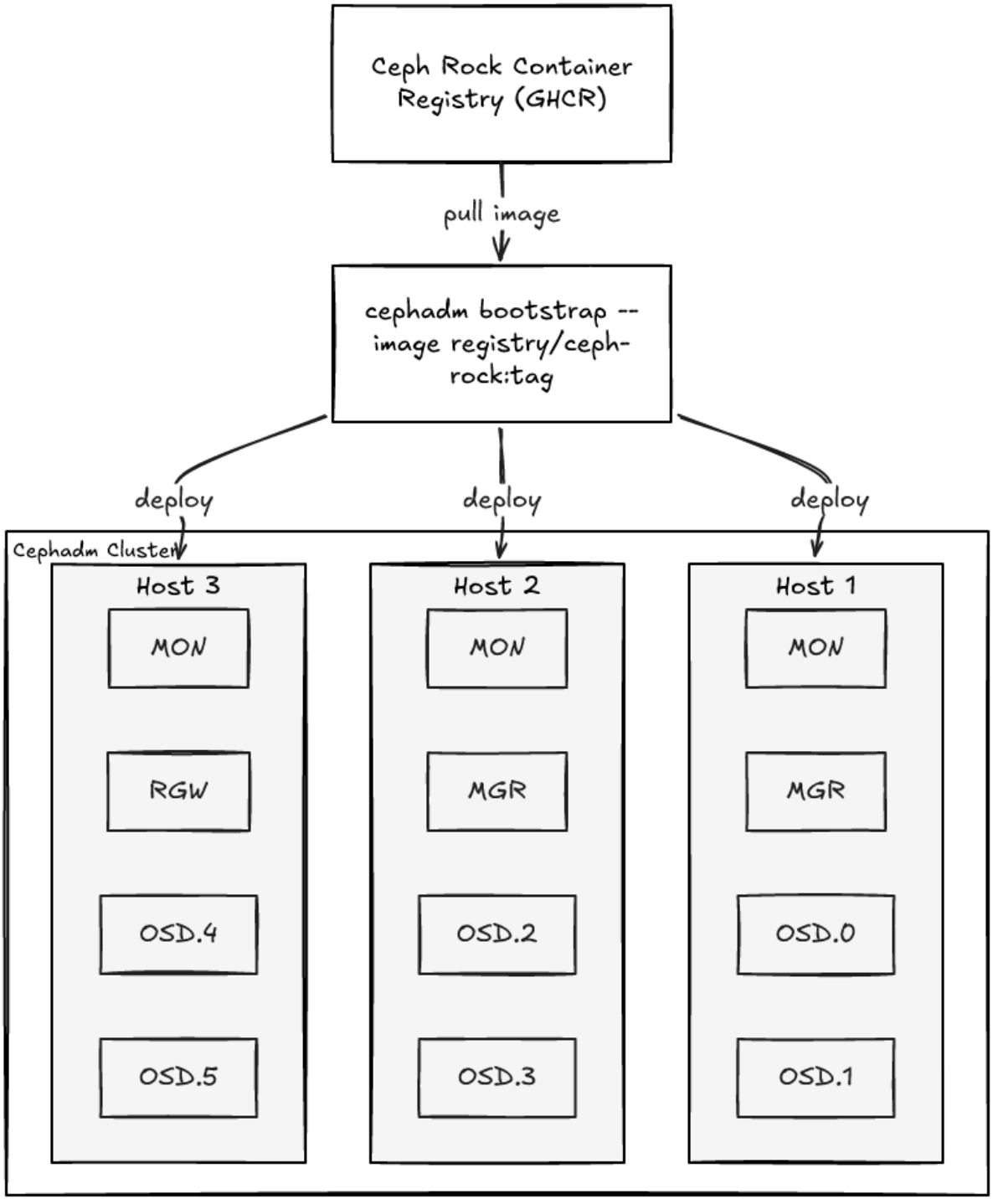


[184] https://charmhub.io/microceph

Ceph Rocks

Ceph Rocks[185] are OCI-compliant container images for users who want to deploy Ceph using their own tooling (cephadm, Rook, or other Kubernetes operators). This provides Canonical's security maintenance and packaging quality while leaving deployment orchestration to whatever container platform you're already using.

Here's a diagram of a cephadm cluster, using Ceph Rocks:



[185] https://github.com/canonical/ceph-containers

Architectural considerations

There are several factors to consider when selecting hardware for building a Ceph storage cluster. Canonical Ceph cluster design choices are influenced by infrastructure node requirements (which are influenced by method of deployment), cluster service placement approach, the purpose of the cluster, and various hardware and software specifications, e.g. Random Access Memory (RAM), operating system (OS), network, disks, server requirements, etc.

Infrastructure node requirements

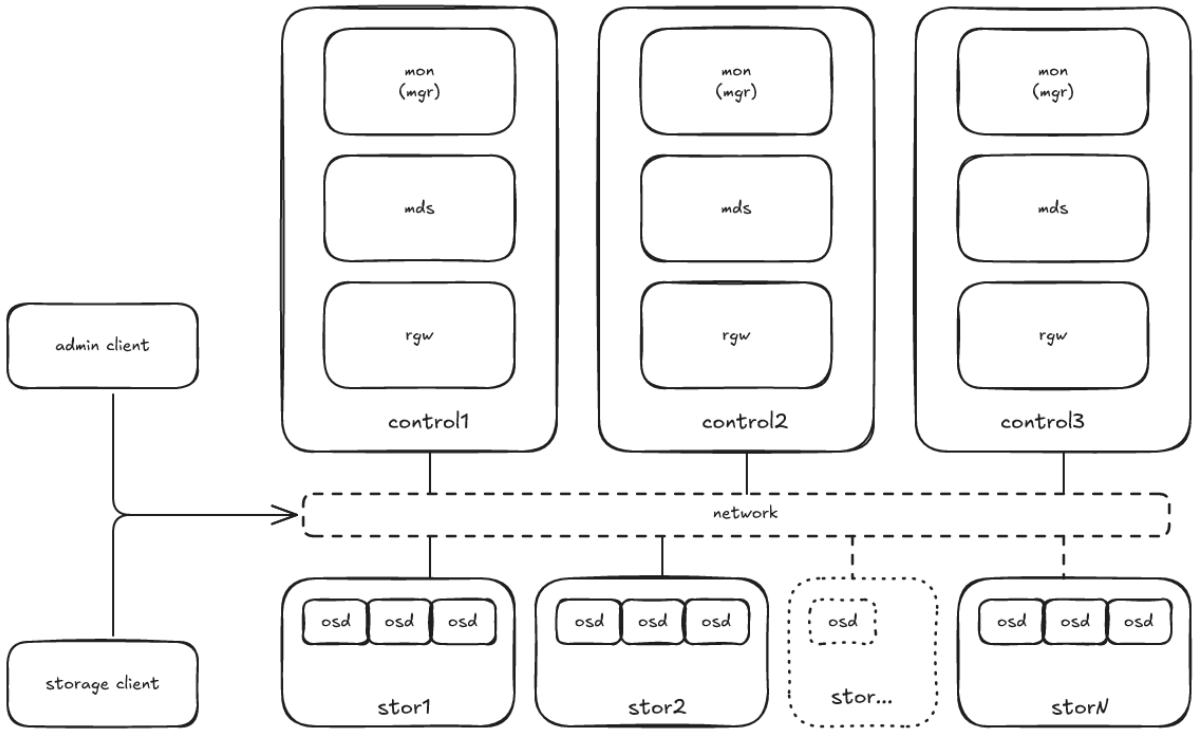
Management infrastructure requirements vary by *deployment option* (page 104). The Juju[186]-based options, i.e., Charmed Ceph and charm-microceph, require a Juju controller environment, typically deployed alongside Metal as a Service (MAAS)[187] for bare-metal provisioning. Canonical recommends dedicating three infrastructure nodes to host Juju and MAAS in a High Availability (HA) configuration. Ceph Rocks and stand-alone MicroCeph options have no such dependency; you bring your own orchestration or manage nodes directly.

Regardless of deployment option, production environments benefit from additional management infrastructure for observability, and fleet management/patching. For observability, we recommend the Canonical Observability Stack (COS)[188], and for management/patching, we recommend using Landscape[189]. These typically require another three-node cluster, which can be colocated with the Juju/MAAS infrastructure nodes where applicable, reducing the total infrastructure footprint.

Cluster service placement

Ceph is composed of several components, both for the control plane and the data plane. Control plane services include Ceph Monitors (MONs), Ceph Managers (MGRs), and Ceph metadata servers (MDSs), whereas the data-plane services include Ceph object storage daemons (OSDs), Ceph RADOS Gateway (RGW), and the Ceph Filesystem (CephFS).

[186] https://canonical.com/juju/docs
[187] https://canonical.com/maas
[188] https://documentation.ubuntu.com/observability/track-2/
[189] https://ubuntu.com/landscape



Canonical recommends the hyperconverged and disaggregated approaches to service placement in the Ceph cluster, depending on your use case. In general, we recommend minimising the number of hardware configurations to simplify capacity planning and replacement strategy.

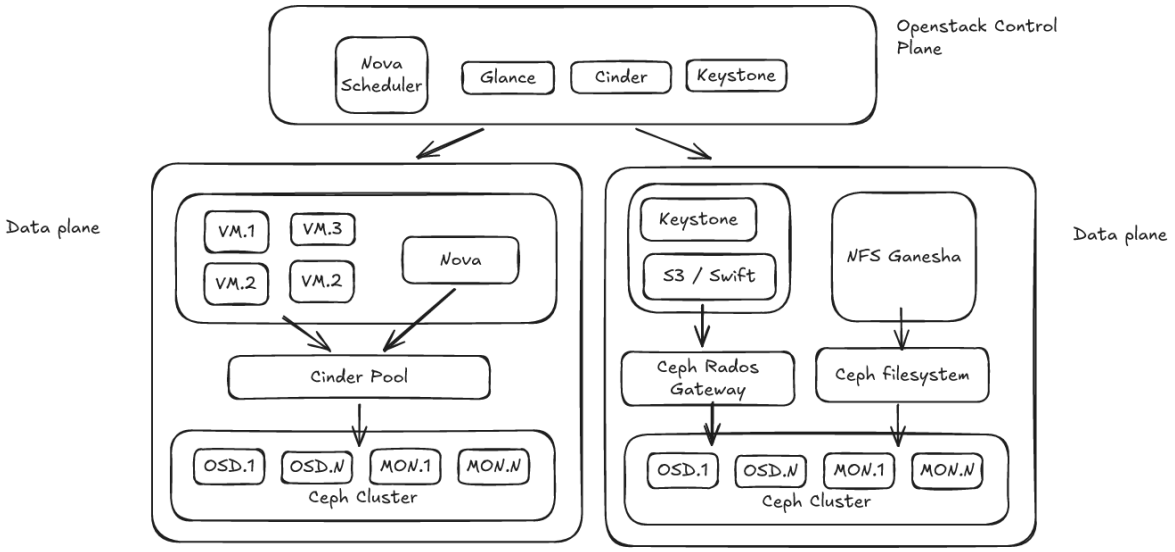
Hyperconverged architecture

The *hyperconverged approach* (page 78) to cluster placement is characterised by a setup where every node in the cloud is hosting the control plane services.

The hyperconverged architecture enables standardisation on a single hardware configuration, ensures maximum resource utilisation and minimises the overall number of nodes. Therefore, it has proven to provide the lowest private infrastructure total cost of ownership (TCO).

The hyperconverged architecture is suitable for general-purpose storage, but it may not be a suitable option for specific workloads and certain storage use cases.

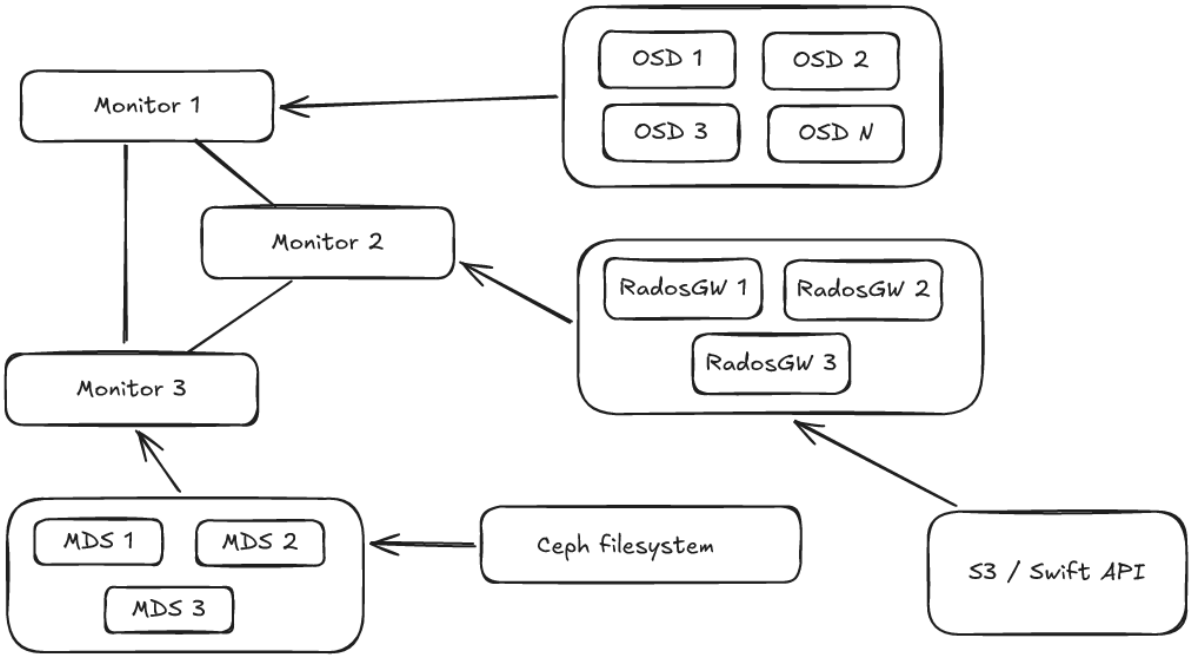
The diagram below is a representation of a hyperconverged architecture setup in the context of an OpenStack deployment.



Disaggregated architecture

The *disaggregated approach* (page 79), on the other hand, features dedicated nodes for each type of service.

This architecture is suitable in specific scenarios, for example, where applications require high performance object or file system storage, because dedicated RGWs or MDSs can provide lower latency and higher bandwidth by allocating dedicated resources to those services.



Purpose of the storage cluster

The next aspect to consider when selecting hardware for a Ceph cluster is the purpose of the storage environment, which may be one of the following:

- Performance
- Capacity

- General-purpose storage

A high performance cluster will look differently from a general-purpose or large capacity cluster.

The type of storage needed will also impact the overall design. For example, a high-performance object storage cluster will perform better if it is not mixed with RBD (block storage) and CephFS (file system storage) storage pools. Empty storage pools can unbalance the distribution of the storage on the cluster and lead to performance issues. For this reason, we will consider three scenarios throughout the reference architecture:

1. Balanced use case; with block, object and file system storage available
2. Object storage only (RGW)
3. File system storage only (CephFS)

Processor requirements

Choosing a processor with sufficient capacity is very important. In most cases, the processor is the only component that cannot be replaced once purchased. Its capacity (measured by the number of cores or threads) cannot be scaled up as with the memory or storage. Moreover, the number of central processing unit (CPU) sockets inside a server is always limited. So, in order to get more CPU resources, it is required to either buy higher-end processors, or buy more servers. More servers generate additional costs on their own, require additional space in the data centre and network fabric, and consume more power, so it is usually more economical to choose higher-end processors unless the cost per thread becomes unreasonably high.

Some CPU resources are required for operating system (OS) and control plane services. Different Ceph services have different CPU resource requirements, and so does the Ubuntu OS. For both OS and control plane services, the performance governor should be used, as Ceph benefits from consistently fast CPU performance.

This section outlines CPU requirements for Ceph Object Storage Daemons (OSDs), Ceph Monitors (MONs), the CephFS Metadata Server (MDS), RADOS Gateway (RGW), and the NVMe-oF Gateway service. Read the upstream Ceph documentation[190] to learn more about CPU recommendations.

OSDs

In general, reserving 2 cores per OSD is sufficient. The type of storage device also impacts the number of cores utilised by the OSD. A non-volatile memory express (NVMe) OSD drive can leverage up to 6 cores, which will provide higher input/output operations per second (IOPS) than if fewer cores are available. It is therefore more reliable to consider the desired IOPS and calculate the number of cores required from there. 1 core can provide 1000–3000 IOPS and 200–500 MB/s throughput.

MONs

Ceph MONs are control plane services that are not very CPU-intensive. Resource requirements spike when the cluster is undergoing changes; the Ceph MON service requires 2 to 4 cores to function adequately. For quorum, Ceph MONs should be an odd number[191].

[190] https://docs.ceph.com/en/latest/start/hardware-recommendations/
[191] https://docs.ceph.com/en/latest/rados/operations/add-or-rm-mons/

MDS

Metadata servers are CPU-intensive. They benefit from high clock rate (GHz), and typically only need 2 to 4 cores per MDS.

RGW

RGW requires a minimum of 2 cores. In high-performance scenarios, providing a dedicated server to RADOSGW with 16+ cores will provide better performance.

NVMe-oF Gateway

The Ceph NVMe-oF Gateway service is a CPU-intensive service, and CPU usage grows together with IOPS. In general, separate hosts are strongly preferred for NVMe-oF gateway daemons (or at least isolating them). Provision at least two NVMe-oF gateways in a gateway group, on separate Ceph cluster nodes, for a highly-available Ceph NVMe/TCP solution.

At least 4 CPU cores dedicated to each NVMe-oF gateway instance are highly recommended.

See CPU requirements for a summary of the processor requirements per type of Ceph service.

Data persistence mechanisms

Ceph uses intelligent data replication to ensure resiliency. Ceph models its underlying infrastructure with the concept of failure domains; replicated data is saved across different failure domains. Each failure domain should be separated in such a way that losing one domain won't impact the remaining failure domains. Examples include, but are not limited to, separated power supplies and cooling systems, networking equipment, etc.

Ceph also offers erasure coding for improved storage efficiency, where data is split into chunks that are distributed across failure domains.

In the unlikely event of a complete domain failure or broken networking between failure domains, Ceph will try to avoid a split-brain scenario. A single failure domain will never form a quorum, so no write operations will happen there. Only if a majority of the failure domains are able to communicate will Ceph permit quorum formation and allow full service.

Info

For replicated pools, the number of replicas for objects in the pool is configured to 3 (`size`). The minimum number of written replicas for objects in the pool in order to acknowledge an I/O operation to the client is by Ceph's default configured to 2 (`min_size`). See the `pool, placement group (PG), and CRUSH config reference`[192] in the upstream documentation.

[192] https://docs.ceph.com/en/latest/rados/configuration/pool-pg-config-ref/#confval-osd_pool_default_min_size

Replication

By default, data in Ceph is stored in three copies over different hosts. Canonical's recommendation is to configure the failure domain to reflect the deployment's physical architecture. This can mean configuring the failure domain to the availability zone (AZ) level, assuming the

physical deployment has at least three separate AZs, preferably four. The three copies are then spread across AZs. This ensures that each AZ can be lost without short-term impact to the Ceph service. In the event of a host failure, Ceph will replicate the objects stored in that host to another host in the same AZ. If a complete AZ is down, Ceph will replicate to another AZ, only if an unused AZ is available.

Erasure coding

Erasure coding allows dividing objects into K data chunks and M calculated coding chunks that allow data to be recovered in the event of a missing chunk. The K+M chunks are spread across failure domains, one chunk per failure domain. This requires the Ceph cluster to have K+M failure domains, with K+M+1 or greater failure domains recommended.

Replication vs. erasure coding

The decision between erasure coding and replicated pools is defined by the trade-off between usable storage and performance needed for the environment. For replicated pools, each data object will typically be replicated twice, which gives 33% of usable storage compared to the raw total of storage. Erasure coding can increase storage efficiency, at the expense of increased CPU usage, increased OSD memory usage, higher latency and increased network load.

Learn how Ceph uses these two data persistence mechanisms in the *Data persistence mechanisms* (page 113) page.

Performance considerations and recommendation

Canonical's testing has shown that the performance of erasure-coded pools is much lower than that of replicated pools, for the same amount of CPUs and RAM. Erasure coding is generally only justified for Write Once Read Many (WORM) use-cases, or cold storage. If erasure coding is being considered to lower the cost of storage, it is necessary to compare the cost of cores and cost of higher clock speed, compared to the cost of storage disks. If it is less expensive to buy more expensive CPUs than it is to purchase additional storage devices, then using erasure coding might be justified for the use case.

Note:

As general guidance, replicated pools should be used in most scenarios, and especially for performance- and latency-sensitive workloads.

Server recommendations

Canonical strongly recommends using rack servers for private Ceph implementations. Software-defined storage (SDS) platforms, like Ceph, are designed to run on commodity hardware. Choosing rack servers for private cloud deployment is therefore a natural move. While blade servers are very popular for traditional virtualisation environments supported by storage area network (SAN) storage, the overall economics for those configurations tend to be very poor due to the inability to utilise commodity parts with multiple suppliers. They also dramatically limit options for storage and network configuration.

Disk recommendations

When it comes to storage configuration, the more important factor to consider is the disk and interface type, rather than the disk capacity. There are various types of storage devices: hard disk drives (HDDs); solid-state drives (SSDs), and various types of storage interfaces: serial-attached SCSI (SAS); Serial Advanced Technology Attachment (SATA), and Non-Volatile Memory Express (NVMe), available on the market. Their cost per GB varies from around $0.03 to $3. Moreover, there are various types of storage available in the cloud: ephemeral, block, and object storage. Therefore, designing storage price-performance requires some more attention and further optimisations.

Optimisation is required for persistent storage (block, file, and object). Canonical Ceph uses replication as default, where data is stored in three copies distributed over three failure domains. This ensures that each failure domain can be lost without short-term impact to services. This also means that the overall amount of raw persistent storage required is usually significant.

Enterprise NVMe or SAS/SATA SSD devices with power loss protection should be purchased. Consumer-grade devices without this feature have very unpredictable performance characteristics, and in the worst-case scenario, can cause data-loss.

NVMe and SSD devices have a durability characteristic, Drive Writes per Day (DWPD); for Ceph, we recommend a value of 1 or above.

Since Ceph is designed to work with more nodes, each with less storage, it is recommended to limit the Ceph raw storage to the allowance per object storage daemon (OSD) node as defined in the Ubuntu Pro service description[193]. OSDs should always be distributed equally across all available OSD nodes in the cluster for predictable performance, higher resilience and sustainable maintenance.

Improving HDD performance

A recommended approach to increase the performance of an OSD is to use a faster storage device for the write-ahead-log (WAL) and database (DB). For example, an OSD on a SAS-HDD could have its WAL and DB located on a faster NVMe device. One faster device can be shared between multiple OSDs by locating the WAL and DB on partitions of the faster device.

Care needs to be taken to ensure that the NVMe device is not used for too many OSDs, otherwise it can become a performance bottleneck and create operational risk when shared amongst a large number of OSDs.

This approach accelerates effective write performance with the WAL and metadata performance with the DB. As metadata performance is an important part of OSD operations, Canonical recommends providing fast storage for both the WAL and DB.

Network recommendations

Network usually tends to be a bottleneck in the traffic between cloud instances and nodes. Therefore, network topology should be designed carefully based on the individual requirements and the characteristics of the workloads. We will recommend network speed based on the different categories of Ceph traffic and the choice of network topology.

[193] https://ubuntu.com/legal/ubuntu-advantage-service-description#uasd-storage-support

Network type

Canonical recommends using at least 10 Gbps Network Interface Cards (NICs) for general-purpose traffic. Since 100 Gbps NICs currently provide the best value for money, measured as the cost per 1 Gbps, they are the preferred option. However, not all types of workloads and environments require such a fast networking configuration.

More precisely, Ceph traffic is divided into three categories:

- **Management traffic** between management tools such as Metal as a Service (MAAS), Juju Controller, and Canonical Observability Stack (COS) on the one hand, and the Ceph cluster on the other. This traffic can live on a 1 Gbps bond at a minimum, but 10 Gbps is recommended.
- The **Ceph Access network**, defined for client access to the Ceph cluster. All Ceph nodes should be on a shared Layer 2 (L2) domain, and ideally the clients should be on the L2 domain of the access network.
- The **Ceph Replication network**, which should be on a separate bond, and ideally at least 25 Gbps to allow for sufficient bandwidth in a recovery situation. When one node fails, replication traffic to recover replicas/parity consumes significant network resources.

To ensure resilience against failures on the network level and eliminate all single points of failure, Canonical recommends using dual-port Network Interface Cards (NICs) for all traffic. Those NICs plug into the network fabric which should also be designed in a highly available fashion.

Network topology

The preferred network fabric topology depends on the size of the deployment. Small-scale deployments that are not immediately expected to grow should use a simple topology with two network switches for high availability. In turn, small-scale deployments that are expected to grow rapidly and large-scale deployments should use a Clos network topology with Layer 3 (L3) leaf and spine switches.

The following types of switches are required for the spine-leaf architecture:

- **Leaf switches** — connect servers inside of a single availability zone (AZ) and act as a gateway to other AZs for general-purpose traffic. Each AZ, consisting of two racks, contains a pair of leaf switches and every server in the AZ is connected to both switches using link aggregation control protocol (LACP) technology. Leaf switches terminate L2 inside of the AZ and can only route connections to other AZs via spine switches.
- **Spine switches** — connect leaf switches inside different AZs using L3 routing protocols. They are connected to every leaf switch, but not to each other. At least two spine switches are required for the recommended cloud deployment.
- **Management switches** — connect all servers to enable operations, administration and management (OAM) traffic and automated server provisioning. Segregating these types of traffic to management switches guarantees reachability even in the face of heavy tenant and storage load and flooding.

In order to prevent network oversubscription, the number of ports in leaf switches should match the number of ports in spine switches.

Minimum node count and rack layout

To qualify for Canonical Ceph design and delivery services, Ubuntu Pro and Managed Ceph, the cloud has to consist of the following components at minimum:

- **Three infrastructure nodes** hosting the automation infrastructure, including Metal as a Service (MAAS), Juju and the observability stack, for the Juju-based options, i.e., Charmed Ceph and charm-microceph.
- **Six Ceph nodes** for a hyperconverged architecture, or **nine nodes** for a disaggregated architecture.
- Required rack, power and network infrastructure.

In order to ensure sufficient power supply, non-oversubscribed network and resilience against failures, we highly recommend designing the hardware layer in the following way:

- Using at least three availability zones (AZs) so that one unit of each automation service and cloud control plane service would be running in a separate AZ.
- Using at least three racks so that services from different AZs would be running physically separated in different racks.
- Distributing hyperconverged nodes equally across racks to ensure sufficient power supply and cooling inside a single rack.
- Using at least two leaf switches per AZ to ensure high availability on the network layer inside the AZ.
- Using a sufficient number of spine switches (usually one per two racks) to ensure network non-oversubscription.
- Using at least two managed switches for management and provisioning.

Canonical provides capacity estimates to ensure that the cloud being built has enough resources to run customer workloads. The actual number of hyperconverged node racks and switches may vary depending on the capacity requirements.

Data encryption

Data encryption in Canonical Ceph is provided via integration with Vault[194]. This encryption takes two forms:

- Encryption in transit
- Encryption at rest (bytes on disk)

Encryption in transit and at rest is enabled by default once Canonical Ceph is related to a Vault `secrets-storage` relation. Data encryption in transit is enabled via the Messenger V2[195] feature.

Encryption can lower the Input/Output (I/O) bandwidth of the storage layer. The impact of this varies per workload and should be considered when deciding whether or not encryption should be enabled.

[194] https://canonical-vault-charms.readthedocs-hosted.com/en/latest/
[195] https://docs.ceph.com/en/latest/rados/configuration/msgr2/

Certified hardware

It is highly recommended to use Ubuntu certified hardware[196]; doing so helps avoid additional charges or project delays. Canonical runs a certification programme with most hardware vendors, through partnership agreements. All servers must be equipped with an intelligent platform management interface (IPMI) and support preboot execution environment (PXE) over all network interface cards (NICs).

Canonical can run a bidding process for its customers to ensure that the customer always gets hardware matching the official recommendations at the lowest possible price. If you have a specific goal, and are already familiar with MicroCeph, our *how-to guides* (page 10) have more in-depth detail and instructions.

Take a look at our *reference section* (page 49) to find our release notes and recommended hardware specifications, or when you need to know which MicroCeph commands to use.

2.3.5. Deploy MicroCeph via Charm

In this tutorial we will deploy a three node charm-microceph cluster in an LXD environment using Juju. We will set up LXD on a single physical machine, and use simulated storage to keep hardware requirements at a minimum. This works great for learning and testing purposes; in a production environment you would typically utilize Juju to deploy to several physical machines and storage media.

First, we will install and configure LXD which will provide virtual machines for our cluster. Then we will install and bootstrap Juju, using LXD as a provider. Subsequently we will install and bootstrap the MicroCeph cluster, and in the final step configure (simulated) storage for MicroCeph.

Prerequisites

To successfully follow this tutorial, you will need:

- a Linux machine with at least 16G of memory and 50G of free disk space
- with snapd installed
- and virtualization-enabled

Note that this tutorial might run into issues on a container (such as docker) as containers typically are lacking virtualization capabilities.

Install and configure LXD

Install the LXD snap and auto-configure it – this will give the host machine the ability to spawn VMs, including networking and storage:

```
$ sudo snap install lxd
$ sudo lxd init --auto
```

Note that depending on your system, LXD might come pre-installed.

[196] https://ubuntu.com/certified

Install and bootstrap Juju

Install and then configure Juju to make use of the LXD provider that was setup in the previous step:

```
$ sudo snap install juju
juju (3/stable) 3.6.8 from Canonical✓ installed

$ juju bootstrap localhost lxd-controller
Since Juju 3 is being run for the first time, it has downloaded the latest public
cloud information.
Creating Juju controller "lxd-controller" on localhost/localhost
...
Now you can run
    juju add-model <model-name>
to create a new model to deploy workloads.
```

You have successfully created a Juju controller named `lxd-controller`. Notice that the juju bootstrap command prompts you to create a Juju model. Models are the logical grouping of connected applications in Juju. We will create a model `mymodel`:

```
$ juju add-model mymodel
Added 'mymodel' model on localhost/localhost with credential 'localhost' for user
'admin'
```

Configure the model to spawn VMs with 4G of memory and a disk of 16G:

```
$ juju set-model-constraints virt-type=virtual-machine mem=4G root-disk=16G
```

For further details consult the Juju documentation on LXD[197]

Deploy MicroCeph

With the Juju environment configured, the next step is to deploy MicroCeph. Deploy three MicroCeph units with this command:

```
$ juju deploy microceph --num-units 3
Deployed "microceph" from charm-hub charm "microceph", revision 155 in channel
squid/stable on ubuntu@24.04/stable
```

Juju deploys the MicroCeph units in the background. This process might take a few minutes depending on network speed and available resources. Check progress by running the `juju status` command. Once the deployment is done, `juju status` will report 3 active units:

```
$ juju status
Model    Controller      Cloud/Region         Version  SLA          Timestamp
mymodel  lxd-controller  localhost/localhost  3.6.8    unsupported  11:15:26Z

App        Version  Status  Scale  Charm      Channel       Rev  Exposed  Message
microceph           active      3  microceph  squid/stable  155  no
```

(continues on next page)

[197] https://documentation.ubuntu.com/juju/3.6/reference/cloud/list-of-supported-clouds/the-lxd-cloud-and-juju/#the-lxd-cloud-and-juju

(continued from previous page)

```
Unit          Workload  Agent  Machine  Public address  Ports  Message
microceph/0   active    idle   0        10.106.25.67
microceph/1   active    idle   1        10.106.25.66
microceph/2*  active    idle   2        10.106.25.144

Machine  State    Address        Inst id        Base          AZ  Message
0        started  10.106.25.67   juju-9fe08a-0  ubuntu@24.04      Running
1        started  10.106.25.66   juju-9fe08a-1  ubuntu@24.04      Running
2        started  10.106.25.144  juju-9fe08a-2  ubuntu@24.04      Running
```

Installing the MicroCeph units also bootstrapped a Ceph cluster. Check the status by SSHing into one unit and running `ceph -s`. This should result in something like the below:

```
$ juju ssh microceph/0 "sudo ceph -s"
  cluster:
    id:     e131a957-bb56-489c-bb10-1782cd29e5f2
    health: HEALTH_WARN
            OSD count 0 < osd_pool_default_size 3

  services:
    mon: 3 daemons, quorum juju-9fe08a-2,juju-9fe08a-0,juju-9fe08a-1 (age 77s)
    mgr: juju-9fe08a-2(active, since 118s), standbys: juju-9fe08a-0, juju-9fe08a-1
    osd: 0 osds: 0 up, 0 in
...
```

The above output shows a running Ceph cluster, however it displays a health warning. Ceph warns us because it by default expects three disks (OSDs in Ceph parlance) for storage, and we have not yet configured any.

Adding disks

For the purposes of this tutorial we will be setting up small simulated disks for ease of configuration. Note these small loop disks are only suitable for a demo setup like this; in a production environment physical disks would be utilized instead.

Run this command to add loop based storage of 2G size to a unit:

```
$ juju add-storage microceph/0 osd-standalone="loop,2G,1"
added storage osd-standalone/0 to microceph/0
```

This will configure the first unit with a 2G OSD. When running `juju status` you should see the unit status change to `executing` and a status message appear that an OSD is being enrolled.

Continue by running the same command for the other two units:

```
juju add-storage microceph/1 osd-standalone="loop,2G,1"
juju add-storage microceph/2 osd-standalone="loop,2G,1"
```

It will take a few minutes to configure the storage for Ceph, but once done (the units will display a status of `active` / `idle`) the Ceph status should look something like this:

```
$ juju ssh microceph/0 "sudo ceph -s"
  cluster:
    id:     e131a957-bb56-489c-bb10-1782cd29e5f2
    health: HEALTH_OK

  services:
    mon: 3 daemons, quorum juju-9fe08a-2,juju-9fe08a-0,juju-9fe08a-1 (age 7m)
    mgr: juju-9fe08a-2(active, since 8m), standbys: juju-9fe08a-0, juju-9fe08a-1
    osd: 3 osds: 3 up (since 34s), 3 in (since 42s)
```

Next steps

We have successfully set up a Juju-managed MicroCeph cluster, ready to serve as a test and learning environment. To see how to configure and integrate MicroCeph with other Juju applications, see the MicroCeph page on Charmhub[198].

2.3.6. How-to guides

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

Install and deploy MicroCeph with Juju

Deploy a MicroCeph cluster from a charm.

Install MicroCeph with Juju

This guide shows how to perform a general install of MicroCeph using Juju.

Prerequisites

- A snapd-compatible host to run the Juju client[199]
- A machine cloud (substrate), e.g. MAAS[200], LXD[201] or AWS[202] at your disposal. See the Juju documentation[203] for the full list of supported clouds and more information on how to set them up.

Important: MicroCeph requires a machine cloud and will not work with a Kubernetes cluster.

- Disks on each node to add as object storage daemons (OSDs) to the Ceph cluster

[198] https://charmhub.io/microceph
[199] https://documentation.ubuntu.com/juju/3.6/howto/manage-juju/?utm_source%3Atakeover=#install-juju
[200] https://documentation.ubuntu.com/juju/3.6/reference/cloud/list-of-supported-clouds/the-maas-cloud-and-juju/
[201] https://documentation.ubuntu.com/juju/3.6/reference/cloud/list-of-supported-clouds/the-lxd-cloud-and-juju/
[202] https://documentation.ubuntu.com/juju/3.6/reference/cloud/list-of-supported-clouds/the-amazon-ec2-cloud-and-juju/
[203] https://documentation.ubuntu.com/juju/3.6/reference/cloud/list-of-supported-clouds/

Deploy MicroCeph

Deploy three units of the MicroCeph charm to three machines.

```
juju deploy -n 3 microceph --channel latest/edge --to 0,1,2
```

The output to the `juju status` command should look similar to this:

```
Model      Controller       Cloud/Region     Version    SLA          Timestamp
microceph  sunbeam-default  sunbeam/default  3.2-beta3  unsupported  03:40:02Z
App        Version  Status  Scale  Charm      Channel      Rev  Exposed  Message
microceph           active      3  microceph  latest/edge    3  no
Unit          Workload  Agent  Machine  Public address  Ports  Message
microceph/0*  active    idle   0        10.5.0.106
microceph/1   active    idle   1        10.5.1.191
microceph/2   active    idle   2        10.5.1.81
Machine  State    Address     Inst id            Base          AZ  Message
0        started  10.5.0.106  manual:10.5.0.106  ubuntu@22.04      Manually
provisioned machine
1        started  10.5.1.191  manual:10.5.1.191  ubuntu@22.04      Manually
provisioned machine
2        started  10.5.1.81   manual:10.5.1.81   ubuntu@22.04      Manually
provisioned machine
```

Verify your deployment

Verify the state of the Ceph cluster by running the `ceph status` command on one of the nodes:

```
juju ssh microceph/leader sudo microceph.ceph status
```

Sample output:

```
cluster:
  id:     edd914f5-fdf8-4b56-bdd7-95d6c5e10d81
  health: HEALTH_WARN
          OSD count 0 < osd_pool_default_size 3

services:
  mon: 3 daemons, quorum microceph2,microceph3,microceph4 (age 57s)
  mgr: microceph2(active, since 74s), standbys: microceph3, microceph4
  osd: 0 osds: 0 up, 0 in

data:
  pools:   0 pools, 0 pgs
  objects: 0 objects, 0 B
  usage:   0 B used, 0 B / 0 B avail
  pgs:
```

The next step is to add disks to the microceph nodes.

Manage cluster storage

Add physical or virtual OSDs to your cluster.

Adding osds

OSDs are added by triggering `add-osd action` on each microceph unit. The unit's underlying machine is understood to house one or more storage devices.

1. List the available disks on the microceph node:

```
juju run microceph/0 list-disks
```

The output of the above command should specify the OSDs that are already attached to Microceph and any unpartitioned disks that can be used to attach to microceph as OSD.

```
Running operation 1 with 1 task
  - task 2 on unit-microceph-0
Waiting for task 2...
osds: '[]'
unpartitioned-disks: '[{''model'': '''', ''size'': ''10.00GiB'', ''type'': '
'virtio'',
  ''path'': ''/dev/disk/by-id/virtio-71aa0fef-aec9-4129-9''}, {''model'': '''', '
'size'':
  ''40.00GiB'', ''type'': ''virtio'', ''path'': ''/dev/disk/by-id/''}]'
```

2. Add the unpartitioned disks as OSD to the microceph cluster:

```
juju run microceph/0 add-osd device-id=<DISK PATH>
```

Multiple disks can be added in the `add-osd` action.

```
juju run microceph/0 add-osd device-id=<DISK PATH>,<DISK PATH>
```

The output of `add-osd action` should look similar to this:

```
$ juju run microceph/0 add-osd device-id=/dev/disk/by-id/virtio-71aa0fef-aec9-
4129-9
Running operation 3 with 1 task
  - task 4 on unit-microceph-0
Waiting for task 4...
status: success
```

3. Verify if the disks are added as OSDs to the ceph cluster:

   juju run microceph/0 list-disks

Now the added disks should be visible in osds. Sample output of the above command:

```
Running operation 5 with 1 task
  - task 6 on unit-microceph-0
Waiting for task 6...
osds: '[{''osd'': ''0'', ''location'': ''microceph2'', ''path'': ''/dev/disk/by-
id/virtio-71aa0fef-aec9-4129-9''}]'
```

(continues on next page)

(continued from previous page)

```
unpartitioned-disks: '[{''model'': '''', ''size'': ''40.00GiB'', ''type'': '
'virtio'',
  ''path'': ''/dev/disk/by-id/''}]'
```

4. Run steps 1,2,3 on all the storage nodes.
5. Run ceph cluster status to check if OSDs are up:

```
juju ssh microceph/leader sudo microceph.ceph status
```

Sample output is:

```
cluster:
  id:     edd914f5-fdf8-4b56-bdd7-95d6c5e10d81
  health: HEALTH_OK

services:
  mon: 3 daemons, quorum microceph2,microceph3,microceph4 (age 12m)
  mgr: microceph2(active, since 13m), standbys: microceph3, microceph4
  osd: 3 osds: 3 up (since 34s), 3 in (since 56s)

data:
  pools:   1 pools, 1 pgs
  objects: 2 objects, 577 KiB
  usage:   66 MiB used, 30 GiB / 30 GiB avail
  pgs:     1 active+clean

io:
  client:   938 B/s rd, 43 KiB/s wr, 0 op/s rd, 1 op/s wr
```

Now the ceph cluster is healthy and ready to use.

Add virtual OSDs for test clusters

MicroCeph also supports creating and using files as backing devices for OSDs. This is a handy feature for testing out MicroCeph without committing physical block devices.

OSDs are added by triggering `add-osd action` on a microceph unit.

Procedure

1. Add file based OSDs to the microceph cluster

Add a single 4G file based OSD:

```
juju run microceph/0 add-osd loop-spec="4G,1"
```

Multiple OSDs (assuming count N) can be added in the `add-osd` action.

```
juju run microceph/0 add-osd loop-spec="4G,<N>"
```

The output of `add-osd action` should look similar to this:

```
$ juju run microceph/0 add-osd loop-spec="4G,3"
Running operation 3 with 1 task
  - task 4 on unit-microceph-0
Waiting for task 4...
status: success
```

2. Run ceph cluster status to check if OSDs are up

```
juju ssh microceph/leader sudo ceph status
```

Sample output is:

```
cluster:
  id:     edd914f5-fdf8-4b56-bdd7-95d6c5e10d81
  health: HEALTH_OK

services:
  mon: 1 daemons, quorum juju-ae397f-mcu-3 (age 20m)
  mgr: juju-ae397f-mcu-3(active, since 20m)
  osd: 3 osds: 3 up (since 7s), 3 in (since 10s)

data:
  pools:   1 pools, 1 pgs
  objects: 2 objects, 449 KiB
  usage:   79 MiB used, 12 GiB / 12 GiB avail
  pgs:     1 active+clean

io:
  client:   938 B/s rd, 43 KiB/s wr, 0 op/s rd, 1 op/s wr
```

Now the ceph cluster is healthy and ready to use.

Configure your cluster

Enable and manage optional cluster services.

Enable rgw

Enable/Disable Ceph RADOS Gateway by setting the config option `enable-rgw`.

1. Enable RGW service

   juju config microceph enable-rgw=”*”

2. Check microceph status

   juju ssh microceph/leader sudo microceph status

The output of the above command should list rgw as part of services running on each node. Sample output is:

```
MicroCeph deployment summary:
- microceph2 (10.121.193.184)
```

(continues on next page)

(continued from previous page)

```
  Services: mds, mgr, mon, rgw, osd
  Disks: 1
- microceph3 (10.121.193.185)
  Services: mds, mgr, mon, rgw, osd
  Disks: 1
- microceph4 (10.121.193.186)
  Services: mds, mgr, mon, rgw, osd
  Disks: 1
```

3. Run ceph cluster status to check if rgw daemon is running

   juju ssh microceph/leader sudo microceph.ceph status

The output of the above command should list rgw under services. Sample output is:

```
cluster:
  id:     edd914f5-fdf8-4b56-bdd7-95d6c5e10d81
  health: HEALTH_OK

services:
  mon: 3 daemons, quorum microceph2,microceph3,microceph4 (age 12m)
  mgr: microceph2(active, since 13m), standbys: microceph3, microceph4
  osd: 3 osds: 3 up (since 34s), 3 in (since 56s)
  rgw: 3 daemons, quorum microceph2,microceph3,microceph4 (age 30s)

data:
  pools:   5 pools, 5 pgs
  objects: 2 objects, 577 KiB
  usage:   66 MiB used, 30 GiB / 30 GiB avail
  pgs:     5 active+clean

io:
  client:   938 B/s rd, 43 KiB/s wr, 0 op/s rd, 1 op/s wr
```

Now the ceph cluster is healthy and ready to use.

4. Disable RGW service

   juju config microceph enable-rgw=""

Perform cluster maintenance

Put individual nodes into and out of maintenance mode safely.

Perform cluster maintenance

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

Prerequisites

Cluster maintenance requires extra redundancy in ceph services, make sure you have

- at least 3 units of MicroCeph

For example, a three-node MicroCeph cluster would look something similar to this:

```
Model  Controller  Cloud/Region         Version  SLA          Timestamp
ceph   overlord    localhost/localhost  3.6.8    unsupported  10:13:41+08:00

App        Version  Status  Scale  Charm      Channel       Rev  Exposed  Message
microceph           active      3  microceph  squid/stable  155  no

Unit          Workload  Agent  Machine  Public address  Ports  Message
microceph/0   active    idle   0        10.42.75.116
microceph/1*  active    idle   1        10.42.75.230
microceph/2   active    idle   2        10.42.75.218

Machine  State    Address       Inst id        Base          AZ  Message
0        started  10.42.75.116  juju-6ae532-0  ubuntu@24.04      Running
1        started  10.42.75.230  juju-6ae532-1  ubuntu@24.04      Running
2        started  10.42.75.218  juju-6ae532-2  ubuntu@24.04      Running

MicroCeph deployment summary:
- juju-6ae532-0 (10.42.75.116)
  Services: mds, mgr, mon, osd
  Disks: 1
- juju-6ae532-1 (10.42.75.230)
  Services: mds, mgr, mon, osd
  Disks: 1
- juju-6ae532-2 (10.42.75.218)
  Services: mds, mgr, mon, osd
  Disks: 1
```

Review the action plan of maintenance mode

The action plan for entering or exiting the maintenance mode can reviewed by using the run dry-run option.

```
juju run microceph/leader exit-maintenance dry-run=True
juju run microceph/leader enter-maintenance dry-run=True
```

Some steps in the action plan can be optionally added or removed. To see what steps are optional, run:

```
juju show-action microceph exit-maintenance
juju show-action microceph enter-maintenance
```

Enter maintenance mode

To put unit `microceph/2` into maintenance mode, and optionally disable the OSD service on that node, run

```
juju run microceph/2 enter-maintenance stop-osds=True
```

Our sample output looks like this:

```
Running operation 17 with 1 task
  - task 18 on unit-microceph-2

Waiting for task 18...
actions:
  step-1:
    description: Check if osds.[3] in node 'juju-6ae532-2' are ok-to-stop.
    error: ""
    id: check-osd-ok-to-stop-ops
  step-2:
    description: Check if there are at least a majority of mon services, 1 mds
service,
      and 1 mgr service in the cluster besides those in node 'juju-6ae532-2'
    error: ""
    id: check-non-osd-svc-enough-ops
  step-3:
    description: Run `ceph osd set noout`.
    error: ""
    id: set-noout-ops
  step-4:
    description: Assert osd has 'noout' flag set.
    error: ""
    id: assert-noout-flag-set-ops
  step-5:
    description: Stop osd service in node 'juju-6ae532-2'.
    error: ""
    id: stop-osd-ops
errors: ""
status: success
```

After entering maintenance mode, this is the status of the cluster:

```
$ juju ssh microceph/2 -- sudo snap services microceph
Service                  Startup   Current   Notes
microceph.cephfs-mirror  disabled  inactive  -
microceph.daemon         enabled   active    -
microceph.mds            enabled   active    -
microceph.mgr            enabled   active    -
microceph.mon            enabled   active    -
microceph.nfs            disabled  inactive  -
microceph.osd            disabled  inactive  -
microceph.rbd-mirror     disabled  inactive  -
microceph.rgw            disabled  inactive  -
```

(continues on next page)

(continued from previous page)

```
$ juju ssh microceph/2 -- sudo microceph.ceph -s
  cluster:
    id:     91da3928-adbb-4675-8dc0-52bb2a07e027
    health: HEALTH_WARN
            mons juju-6ae532-0,juju-6ae532-1,juju-6ae532-2 are low on available
space
            noout flag(s) set
            1 osds down
            1 host (1 osds) down
            Degraded data redundancy: 2/6 objects degraded (33.333%), 1 pg
degraded, 1 pg undersized

  services:
    mon: 3 daemons, quorum juju-6ae532-1,juju-6ae532-0,juju-6ae532-2 (age 9m)
    mgr: juju-6ae532-1(active, since 10m), standbys: juju-6ae532-0, juju-6ae532-2
    osd: 3 osds: 2 up (since 64s), 3 in (since 7m)
         flags noout

  data:
    pools:   1 pools, 1 pgs
    objects: 2 objects, 449 KiB
    usage:   81 MiB used, 12 GiB / 12 GiB avail
    pgs:     2/6 objects degraded (33.333%)
             1 active+undersized+degraded
```

Compare the status of the cluster with the cluster status **before** entering maintenance mode:

```
$ juju ssh microceph/2 -- sudo snap services microceph
Service                  Startup   Current   Notes
microceph.cephfs-mirror  disabled  inactive  -
microceph.daemon         enabled   active    -
microceph.mds            enabled   active    -
microceph.mgr            enabled   active    -
microceph.mon            enabled   active    -
microceph.nfs            disabled  inactive  -
microceph.osd            enabled   active    -
microceph.rbd-mirror     disabled  inactive  -
microceph.rgw            disabled  inactive  -

$ juju ssh microceph/2 -- sudo microceph.ceph -s
  cluster:
    id:     91da3928-adbb-4675-8dc0-52bb2a07e027
    health: HEALTH_WARN
            mons juju-6ae532-0,juju-6ae532-1,juju-6ae532-2 are low on available
space

  services:
    mon: 3 daemons, quorum juju-6ae532-1,juju-6ae532-0,juju-6ae532-2 (age 12m)
```

(continues on next page)

(continued from previous page)

```
    mgr: juju-6ae532-1(active, since 12m), standbys: juju-6ae532-0, juju-6ae532-2
    osd: 3 osds: 3 up (since 50s), 3 in (since 9m)

  data:
    pools:   1 pools, 1 pgs
    objects: 2 objects, 449 KiB
    usage:   481 MiB used, 12 GiB / 12 GiB avail
    pgs:     1 active+clean
```

[!Note] The `microceph.osd` service is disabled and inactive after entering maintenance mode; the cluster also has noout flag set.

Exit maintenance mode for microceph node

To recover unit `microceph/2` from maintenance mode, run

```
juju run microceph/2 exit-maintenance
```

Our sample output looks like this:

```
$ juju run microceph/2 exit-maintenance

Running operation 19 with 1 task
  - task 20 on unit-microceph-2

Waiting for task 20...
actions:
  step-1:
    description: Run `ceph osd unset noout`.
    error: ""
    id: unset-noout-ops
  step-2:
    description: Assert osd has 'noout' flag unset.
    error: ""
    id: assert-noout-flag-unset-ops
  step-3:
    description: Start osd service in node 'juju-6ae532-2'.
    error: ""
    id: start-osd-ops
errors: ""
status: success
```

This is the cluster status after exiting maintenance node for unit `microceph/2`

```
$ juju ssh microceph/2 -- sudo snap services microceph
Service                  Startup   Current   Notes
microceph.cephfs-mirror  disabled  inactive  -
microceph.daemon         enabled   active    -
microceph.mds            enabled   active    -
```

(continues on next page)

(continued from previous page)

```
microceph.mgr            enabled   active    -
microceph.mon            enabled   active    -
microceph.nfs            disabled  inactive  -
microceph.osd            enabled   active    -
microceph.rbd-mirror     disabled  inactive  -
microceph.rgw            disabled  inactive  -

$ juju ssh microceph/2 -- sudo microceph.ceph -s
  cluster:
    id:     91da3928-adbb-4675-8dc0-52bb2a07e027
    health: HEALTH_WARN
            mons juju-6ae532-0,juju-6ae532-1,juju-6ae532-2 are low on available
space

  services:
    mon: 3 daemons, quorum juju-6ae532-1,juju-6ae532-0,juju-6ae532-2 (age 16m)
    mgr: juju-6ae532-1(active, since 16m), standbys: juju-6ae532-0, juju-6ae532-2
    osd: 3 osds: 3 up (since 4m), 3 in (since 13m)

  data:
    pools:   1 pools, 1 pgs
    objects: 2 objects, 449 KiB
    usage:   481 MiB used, 12 GiB / 12 GiB avail
    pgs:     1 active+clean
```

[!Note] The `microceph.osd` service is enabled and active again after exiting maintenance mode; the cluster also does not have noout flag set.

2.3.7. 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[204] in our GitHub repository or talk to us on our Matrix[205] 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.

MicroCeph documentation overview

The MicroCeph documentation is hosted in a GitHub repository[206] and published on Read the Docs[207]. You need to create a GitHub account to participate, but you do not need a Read the

[204] https://github.com/canonical/microceph/issues/new
[205] https://matrix.to/#/#ceph-general:ubuntu.com
[206] https://github.com/canonical/microceph
[207] https://about.readthedocs.com/

Docs account.

Contributing on GitHub

To create issues, comment, reply, or submit contributions, you need to set up a GitHub[208] account and a Git[209] 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 in the Open Documentation Academy Git guide[210].

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[211] 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.

Diátaxis

Our documentation content, style and navigational structure follows the Diátaxis[212] 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[213].

[208] https://github.com/
[209] https://git-scm.com/
[210] https://canonical-open-documentation-academy.readthedocs.io/en/latest/docs/howto/get-started/using_git/
[211] https://ubuntu.com/legal/contributors
[212] https://diataxis.fr/
[213] https://ubuntu.com/blog/diataxis-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.

Open Documentation Academy

MicroCeph is a proud member of the Canonical Open Documentation Academy[214] (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[215] . Take a look, bookmark it, and see our guide on getting started with the Open Documentation Academy[216] for next steps.

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

- Our discussion forum[217] on the Ubuntu Community Hub
- In the Matrix room[218] for interactive chat
- Follow us on Fosstodon[219] 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 Community Hour forum post[220].

Finally, subscribe to our Documentation event calendar[221]. 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[222]. You must sign and agree to the Canonical CLA.

[214] https://github.com/canonical/open-documentation-academy
[215] https://github.com/canonical/open-documentation-academy/issues
[216] https://discourse.ubuntu.com/t/getting-started/42769
[217] https://canonical.com/documentation/open-documentation-academy
[218] https://matrix.to/#/#documentation:ubuntu.com
[219] https://fosstodon.org/@CanonicalDocumentation
[220] https://discourse.ubuntu.com/t/documentation-office-hours/42771
[221] https://calendar.google.com/calendar/u/0?cid=Y19mYTY4YzE5YWEwY2Y4YWE1ZWNkNzMyNjZmNmM0ZDllOTRhNTIwNTNj
[222] https://ubuntu.com/community/ethos/code-of-conduct

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[223]; 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.

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

[223] https://docs.ubuntu.com/styleguide/en

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.