Red Hat Ceph Storage 1.3 Object Gateway Guide for Ubuntu

Installing, configuring, and administering the Ceph Storage Object Gateway on Ubuntu

Red Hat Ceph Storage Documentation Team

Red Hat Ceph Storage 1.3 Object Gateway Guide for Ubuntu

Installing, configuring, and administering the Ceph Storage Object Gateway on Ubuntu

Abstract

This document provides instructions for installing, configuring, and administering the Ceph Storage Object Gateway on Ubuntu 14.04 running on AMD64 and Intel 64 architectures.

Table of Contents

CHAPTER 1. OVERVIEW

Designed for cloud infrastructures and web-scale object storage, Red Hat® Ceph Storage is a massively scalable, open, software-defined storage platform that combines the most stable version of Ceph with a Ceph management platform, deployment tools, and support services. Providing the tools to flexibly and cost-effectively manage petabyte-scale data deployments in the enterprise, Red Hat Ceph Storage manages cloud data so enterprises can focus on managing their businesses.

Ceph Object Gateway is an object storage interface built on top of **`librados`** to provide applications with a RESTful gateway to Ceph Storage Clusters. Ceph Object Storage supports two interfaces:

1. **S3-compatible:** Provides object storage functionality with an interface that is compatible with a large subset of the Amazon S3 RESTful API.

2. **Swift-compatible:** Provides object storage functionality with an interface that is compatible with a large subset of the OpenStack Swift API.

Ceph Object Storage uses the Ceph Object Gateway daemon (**`radosgw`**), which is a server for interacting with a Ceph Storage Cluster. Since it provides interfaces compatible with OpenStack Swift and Amazon S3, the Ceph Object Gateway has its own user management. Ceph Object Gateway can store data in the same Ceph Storage Cluster used to store data from Ceph Block Device clients; however, you will use separate pools. The S3 and Swift APIs share a common namespace, so you may write data with one API and retrieve it with the other.

CHAPTER 2. INSTALLATION

For Red Hat Ceph Storage v1.3, Red Hat supports the Ceph Object Gateway running on Civetweb (embedded into the **radosgw** daemon) instead of Apache and FastCGI. Using Civetweb simplifies the installation and configuration.

Note

To run the Ceph Object Gateway service On Ubuntu **Trusty** 14.04, you should have a running Ceph storage cluster, and the Ceph Object Gateway node should have access to the public network.

Note

In version 1.3, the Ceph Object Gateway does not support SSL. You may setup a reverse proxy server with SSL to dispatch HTTPS requests as HTTP requests to CivetWeb.

2.1. PREREQUISITES

Refer to the **Red Hat Ceph Storage 1.3 Installation Guide for Ubuntu (x86_64)** and execute the pre-installation procedures on your Ceph Object Gateway node. Specifically, you should set up a Ceph Deploy user with password-less **sudo**. For Ceph Object Gateways, you will need to open the port that Civetweb will use in production.

Note

Civetweb runs on port **7480** by default.

2.2. INSTALLATION TYPES

Starting from version 1.3.1, Red Hat Ceph Storage supports the use of online repositories from https://rhcs.download.redhat.com/ubuntu. So, for RHCS v1.3.2, you can install repository in the Ceph Object Gateway node in two ways:

- ISO based installation
- Online repository based installation

2.2.1. ISO Based Installation

In ISO based installation, to install and setup Ceph Object Gateway on the Ceph Object Gateway node, you will need either **ceph-mon** or **ceph-osd** repository. Both of these repositories contain the **radosgw** package. So, you can use anyone of them for the Ceph Object Gateway node.

To install **ceph-mon** or **ceph-osd** repository on Ceph Object Gateway node, execute the following from **admin** node:

```
ceph-deploy repo [ceph-mon|ceph-osd] <gateway-node>
```

For example:

```
ceph-deploy repo ceph-mon <gateway-node>
```

The above command will write a **`.list`** file on the Ceph Object Gateway node pointing to the local repositories created on the admin node by **`ice_setup`**. This will enable the Ceph Object Gateway node to receive packages from local repositories.

2.2.2. Online Repository Based Installation

In online repository based installation, to install and setup Ceph Object Gateway on the Ceph Object Gateway node, you will need **`Tools`** repository.

To install **`Tools`** repository in Ceph Object Gateway node, execute the following from **`admin`** node:

```
ceph-deploy repo --repo-url
'https://customername:customerpasswd@rhcs.download.redhat.com/ubuntu/1.
3-updates/Tools' --gpg-url https://www.redhat.com/security/fd431d51.txt
Tools <gateway-node>
```

Important

To execute the above command, your **`admin`** node should be properly setup with the required online repositories and the latest version of **`ceph-deploy`** for RHCS v1.3.1 should be installed. See the **`Set Online Ceph Repositories`** section in **`Red Hat Ceph Storage 1.3 Installation Guide for Ubuntu (x86_64)`** for details.

2.3. INSTALLING THE OBJECT GATEWAY

From the working directory of your administration server, install the Ceph Object Gateway package on the Ceph Object Gateway node. For example:

```
ceph-deploy install --rgw <gateway-node1> [<gateway-node2> ...]
```

The **`ceph-common`** package is a dependency, so **`ceph-deploy`** will install this too. The **`ceph`** CLI tools are intended for administrators. To make your Ceph Object Gateway node an administrator node, execute the following from the working directory of your administration server.

```
ceph-deploy admin <node-name>
```

2.4. CREATE A GATEWAY INSTANCE

From the working directory of your administration server, execute the following to create an instance of the Ceph Object Gateway on the Ceph Object Gateway:

```
ceph-deploy rgw create <gateway-node1>
```

Once the gateway is running, you should be able to access it on port **`7480`** with an unauthenticated request in a web browser or a command line HTTP client (e.g., **`curl`**, **`wget`** etc.) like this:

```
http://gateway-node1:7480
```

If the gateway instance is working properly, your browser or CLI HTTP client should receive a response like this:

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

If at any point you run into trouble and you want to start over, execute the following from the working directory of your administration server to purge the configuration:

```
ceph-deploy purge <gateway-node1> [<gateway-node2>]
ceph-deploy purgedata <gateway-node1> [<gateway-node2>]
```

If you execute **`purge`**, you must re-install Ceph.

2.5. CHANGING YOUR DEFAULT PORT

Civetweb runs on port **`7480`** by default. To change the default port (e.g,. to port **`80`**), modify your Ceph configuration file in **`/etc/ceph`** directory of your administration server. Add a section entitled **`[client.rgw.<gateway-node>]`**, replacing **`<gateway-node>`** with the short host name of your Ceph Object Gateway node (i.e., **`hostname -s`**).

Note

In version 1.3, the Ceph Object Gateway does not support SSL. You may setup a reverse proxy web server with SSL to dispatch HTTPS requests as HTTP requests to CivetWeb.

For example, if your node name is **`gateway-node`**, add a section like this after the **`[global]`** section:

```
[client.rgw.gateway-node]
rgw_frontends = "civetweb port=80"
```

Note

Ensure that you leave no whitespace between **`port=<port-number>`** in the **`rgw_frontends`** key/value pair. The **`[client.rgw.gateway-node]`** heading identifies this portion of the Ceph configuration file as configuring a Ceph Storage Cluster client where the client type is a Ceph Object Gateway (i.e., **`rgw`**), and the name of the instance is **`gateway-node`**.

Pull the updated configuration file from **`/etc/ceph`** directory to the working directory of your administration server (e.g. ceph-config directory) and push it to your Ceph Object Gateway node and other Ceph nodes.

```
ceph-deploy --overwrite-conf config pull <admin-node>
ceph-deploy --overwrite-conf config push <gateway-node> [<other-nodes>]
```

Stop Apache so that Ceph Object Gateway can run on port **`80`**.

```
sudo service apache2 stop
```

To make the new port setting take effect, restart the Ceph Object Gateway.

```
sudo service radosgw restart id=rgw.<short-hostname>
```

Finally, check to ensure that the port you selected is open on the node's firewall (e.g., port **`80`**). If it is not open, add the port and reload the firewall configuration. For example:

```
sudo iptables --list
sudo iptables -I INPUT 1 -i <iface> -p tcp -s <ip-address>/<netmask> --
dport 80 -j ACCEPT
```

Replace **`<iface>`** and **`<ip-address>/<netmask>`** with the relevant values for your Ceph Object Gateway node.

Once you have finished configuring **`iptables`**, ensure that you make the change persistent so that it will be in effect when your Ceph Object Gateway node reboots.

Execute:

```
sudo apt-get install iptables-persistent
```

A terminal UI will open up. Select **`yes`** for the prompts to save current **`IPv4`** iptables rules to **`/etc/iptables/rules.v4`** and current **`IPv6`** iptables rules to **`/etc/iptables/rules.v6`**.

The **`IPv4`** iptables rule that you set in the earlier step will be loaded in **`/etc/iptables/rules.v4`** and will be persistent across reboots.

If you add a new **`IPv4`** iptables rule after installing **`iptables-persistent`** you will have to add it to the rule file. In such case, execute the following as a **`root`** user:

```
iptables-save > /etc/iptables/rules.v4
```

2.6. MIGRATING FROM APACHE TO CIVETWEB

If you're running the Ceph Object Gateway on Apache and FastCGI with Red Hat Ceph Storage v1.2.x or above, you're already running Civetweb. It starts with the **`radosgw`** daemon and it's running on port 7480 by default so that it doesn't conflict with your Apache and FastCGI installation and other commonly used web service ports. Migrating to use Civetweb basically involves removing your Apache installation. Then, you must remove Apache and FastCGI settings from your Ceph configuration file and reset **`rgw_frontends`** to Civetweb.

Referring back to the description for installing a Ceph Object Gateway with **`ceph-deploy`**, notice that the configuration file only has one setting **`rgw_frontends`** (and that's assuming you elected to change the default port). The **`ceph-deploy`** utility generates the data directory and the keyring for you, placing the keyring in **`/var/lib/ceph/radosgw/{rgw-intance}`**. The daemon looks in default locations, whereas you may have specified different settings in your Ceph configuration file. Since you already have keys and a data directory, you will want to maintain those paths in your Ceph configuration file if you used something other than default paths.

A typical Ceph Object Gateway configuration file for an Apache-based deployment looks something like this:

```
[client.radosgw.gateway-node]
host = {hostname}
keyring = /etc/ceph/ceph.client.radosgw.keyring
rgw socket path = /var/run/ceph/ceph.radosgw.gateway.fastcgi.sock
log file = /var/log/radosgw/client.radosgw.gateway-node1.log
```

To modify it for use with Civetweb, simply remove the Apache-specific setting **`rgw_socket_path`**. Then, change the **`rgw_frontends`** setting to reflect Civetweb rather than the Apache FastCGI front end and specify the port number you intend to use. For example:

```
[client.radosgw.gateway-node]
host = {hostname}
keyring = /etc/ceph/ceph.client.radosgw.keyring
log file = /var/log/radosgw/client.radosgw.gateway-node1.log
rgw_frontends = civetweb port=80
```

Finally, restart the Ceph Object Gateway.

```
sudo service radosgw restart id=rgw.<short-hostname>
```

If you used a port number that is not open, you will also need to open that port on your firewall.

2.7. ADDING A WILDCARD TO DNS

To use Ceph with S3-style subdomains (e.g., bucket-name.domain-name.com), you need to add a wildcard to the DNS record of the DNS server you use with the **`ceph-radosgw`** daemon.

The address of the DNS must also be specified in the Ceph configuration file with the **`rgw dns name = {hostname}`** setting.

For **`dnsmasq`**, add the following address setting with a dot (.) prepended to the host name:

```
address=/.{hostname-or-fqdn}/{host-ip-address}
```

For example:

```
address=/.gateway-node1/192.168.122.75
```

For **`bind`**, add a wildcard to the DNS record. For example:

```
$TTL    604800
@       IN      SOA     gateway-node1. root.gateway-node1. (
                              2         ; Serial
                         604800         ; Refresh
                         86400         ; Retry
                       2419200         ; Expire
                        604800 )       ; Negative Cache TTL
;
@       IN      NS      gateway-node1.
@       IN      A       192.168.122.113
*       IN      CNAME   @
```

Restart your DNS server and ping your server with a subdomain to ensure that your **`ceph-radosgw`** daemon can process the subdomain requests:

```
ping mybucket.{hostname}
```

For example:

```
ping mybucket.gateway-node1
```

2.8. ADJUSTING LOGGING AND DEBUGGING OUTPUT

Once you finish the setup procedure, check your logging output to ensure it meets your needs. Log files are located in **`/var/log/radosgw`** by default. If you encounter issues with your configuration, you can increase logging and debugging messages in the **`[global]`** section of your Ceph configuration file and restart the gateway(s) to help troubleshoot any configuration issues. For example:

```
[global]
#append the following in the global section.
debug ms = 1
debug rgw = 20
debug civetweb = 20
```

You may also modify these settings at runtime. For example:

```
ceph tell osd.0 injectargs --debug_civetweb 10/20
```

For general details on logging and debugging, see Logging and Debugging. For Ceph Object Gateway-specific details on logging settings, see **Logging Settings** in this guide.

2.9. USING THE OBJECT GATEWAY

To use the REST interfaces, first create an initial Ceph Object Gateway user for the S3 interface. Then, create a subuser for the Swift interface. You then need to verify if the created users are able to access the gateway.

2.9.1. Create a `radosgw` User for S3 Access

A **`radosgw`** user needs to be created and granted access. The command **`man radosgw-admin`** will provide information on additional command options.

To create the user, execute the following on the **`gateway host`**:

```
sudo radosgw-admin user create --uid="testuser" --display-name="First
User"
```

The output of the command will be something like the following:

```
{
 "user_id": "testuser",
 "display_name": "First User",
 "email": "",
 "suspended": 0,
 "max_buckets": 1000,
 "auid": 0,
 "subusers": [],
 "keys": [{
  "user": "testuser",
  "access_key": "I0PJDPCIYZ665MW88W9R",
  "secret_key": "dxaXZ8U90SXydYzyS5ivamEP20hkLSUViiaR+ZDA"
 }],
 "swift_keys": [],
 "caps": [],
 "op_mask": "read, write, delete",
 "default_placement": "",
 "placement_tags": [],
 "bucket_quota": {
  "enabled": false,
  "max_size_kb": -1,
  "max_objects": -1
 },
 "user_quota": {
  "enabled": false,
  "max_size_kb": -1,
  "max_objects": -1
 },
 "temp_url_keys": []
}
```

Note

The values of **keys→access_key** and **keys→secret_key** are needed for access validation.

Important

Check the key output. Sometimes **radosgw-admin** generates a JSON escape character \ in **access_key** or **secret_key** and some clients do not know how to handle JSON escape characters. Remedies include removing the JSON escape character \, encapsulating the string in quotes, regenerating the key and ensuring that it does not have a JSON escape character or specify the key and secret manually. Also, if **radosgw-admin** generates a JSON escape character \ and a forward slash **/** together in a key, like **\\/**, only remove the JSON escape character \. Do not remove the forward slash **/** as it is a valid character in the key.

2.9.2. Create a Swift User

A Swift subuser needs to be created if this kind of access is needed. Creating a Swift user is a two step process. The first step is to create the user. The second is to create the secret key.

Execute the following steps on the **`gateway host`**:

Create the Swift user:

```
sudo radosgw-admin subuser create --uid=testuser --
subuser=testuser:swift --access=full
```

The output will be something like the following:

```
{
 "user_id": "testuser",
 "display_name": "First User",
 "email": "",
 "suspended": 0,
 "max_buckets": 1000,
 "auid": 0,
 "subusers": [{
  "id": "testuser:swift",
  "permissions": "full-control"
 }],
 "keys": [{
  "user": "testuser:swift",
  "access_key": "3Y1LNW4Q6X0Y53A52DET",
  "secret_key": ""
 }, {
  "user": "testuser",
  "access_key": "I0PJDPCIYZ665MW88W9R",
  "secret_key": "dxaXZ8U90SXydYzyS5ivamEP20hkLSUViiaR+ZDA"
 }],
 "swift_keys": [],
 "caps": [],
 "op_mask": "read, write, delete",
 "default_placement": "",
 "placement_tags": [],
 "bucket_quota": {
  "enabled": false,
  "max_size_kb": -1,
  "max_objects": -1
 },
 "user_quota": {
  "enabled": false,
  "max_size_kb": -1,
  "max_objects": -1
 },
 "temp_url_keys": []
}
```

Create the secret key:

```
sudo radosgw-admin key create --subuser=testuser:swift --key-type=swift
--gen-secret
```

–

The output will be something like the following:

```
{
 "user_id": "testuser",
 "display_name": "First User",
 "email": "",
 "suspended": 0,
 "max_buckets": 1000,
 "auid": 0,
 "subusers": [{
  "id": "testuser:swift",
  "permissions": "full-control"
 }],
 "keys": [{
  "user": "testuser:swift",
  "access_key": "3Y1LNW4Q6X0Y53A52DET",
  "secret_key": ""
 }, {
  "user": "testuser",
  "access_key": "I0PJDPCIYZ665MW88W9R",
  "secret_key": "dxaXZ8U90SXydYzyS5ivamEP20hkLSUViiaR+ZDA"
 }],
 "swift_keys": [{
  "user": "testuser:swift",
  "secret_key": "244+fz2gSqoHwR3lYtSbIyomyPHf3i7rgSJrF\/IA"
 }],
 "caps": [],
 "op_mask": "read, write, delete",
 "default_placement": "",
 "placement_tags": [],
 "bucket_quota": {
  "enabled": false,
  "max_size_kb": -1,
  "max_objects": -1
 },
 "user_quota": {
  "enabled": false,
  "max_size_kb": -1,
  "max_objects": -1
 },
 "temp_url_keys": []
}
```

2.9.3. Test S3 Access

You need to write and run a Python test script for verifying S3 access. The S3 access test script will connect to the **`radosgw`**, create a new bucket and list all buckets. The values for **`aws_access_key_id`** and **`aws_secret_access_key`** are taken from the values of **`access_key`** and **`secret_key`** returned by the **`radosgw_admin`** command.

Execute the following steps:

1. You will need to install the **`python-boto`** package.

   ▪

```
sudo apt-get install python-boto
```

2. Create the Python script:

   ```
   vi s3test.py
   ```

3. Add the following contents to the file:

   ```
   import boto
   import boto.s3.connection
   access_key = 'I0PJDPCIYZ665MW88W9R'
   secret_key = 'dxaXZ8U90SXydYzyS5ivamEP20hkLSUViiaR+ZDA'
   conn = boto.connect_s3(
   aws_access_key_id = access_key,
   aws_secret_access_key = secret_key,
   host = '{FQDN}',
   port = {port},
   is_secure=False,
   calling_format = boto.s3.connection.OrdinaryCallingFormat(),
   )
   bucket = conn.create_bucket('my-new-bucket')
   for bucket in conn.get_all_buckets():
    print "{name}\t{created}".format(
     name = bucket.name,
     created = bucket.creation_date,
   )
   ```

   Replace **`{FQDN}`** with the full hostname i.e, the fully qualified domain name of Ceph Object Gateway node. Replace {port} with the port number you are using with Civetweb.

4. Run the script:

   ```
   python s3test.py
   ```

   The output will be something like the following:

   ```
   my-new-bucket 2015-02-16T17:09:10.000Z
   ```

2.9.4. Test Swift Access

Swift access can be verified via the **`swift`** command line client. The command **`man swift`** will provide more information on available command line options.

To install **`swift`** client, execute the following:

```
sudo apt-get install python-setuptools
sudo easy_install pip
sudo pip install --upgrade setuptools
sudo pip install --upgrade python-swiftclient
```

To test swift access, execute the following:

```
swift -A http://{IP ADDRESS}:{port}/auth/1.0 -U testuser:swift -K
'{swift_secret_key}' list
```

–

Replace **`{IP ADDRESS}`** with the public IP address of the gateway server and **`{swift_secret_key}`** with its value from the output of **`radosgw-admin key create`** command executed for the **`swift`** user. Replace {port} with the port number you are using with Civetweb (e.g., **`7480`** is the default). If you don’t replace the port, it will default to port **`80`**.

For example:

```
swift -A http://10.19.143.116:7480/auth/1.0 -U testuser:swift -K
'244+fz2gSqoHwR3lYtSbIyomyPHf3i7rgSJrF/IA' list
```

The output should be:

```
my-new-bucket
```

CHAPTER 3. ADMINISTRATION (CLI)

Administrators can manage the Ceph Object Gateway using the **`radosgw-admin`** command-line interface.

- Administrative Data Storage
- Storage Policies
- Bucket Sharding
- User Management
- Quota Management
- Usage Tracking

3.1. ADMINISTRATIVE DATA STORAGE

A Ceph Object Gateway stores administrative data in a series of pools defined in an instance's zone configuration. For example, the buckets, users, user quotas and usage statistics discussed in the subsequent sections are stored in pools in the Ceph Storage Cluster. By default, Ceph Object Gateway will create the following pools and map them to the default zone.

- **`.rgw`**
- **`.rgw.control`**
- **`.rgw.gc`**
- **`.log`**
- **`.intent-log`**
- **`.usage`**
- **`.users`**
- **`.users.email`**
- **`.users.swift`**
- **`.users.uid`**

You should consider creating these pools manually so that you can set the CRUSH ruleset and the number of placement groups. In a typical configuration, the pools that store the Ceph Object Gateway's administrative data will often use the same CRUSH ruleset and use fewer placement groups, because there are 10 pools for the administrative data. See Pools and Storage Strategies for additional details.

Also see Ceph Placement Groups (PGs) per Pool Calculator for placement group calculation details. The **`mon_pg_warn_max_per_osd`** setting warns you if assign too many placement groups to a pool (i.e., 300 by default). You may adjust the value to suit your needs and the capabilities of your hardware where **`n`** is the maximum number of PGs per OSD.

```
mon_pg_warn_max_per_osd = n
```

3.2. STORAGE POLICIES

Ceph Object Gateway stores the client bucket and object data by identifying placement targets, and storing buckets and objects in the pools associated with a placement target. If you don't configure placement targets and map them to pools in the instance's zone configuration, the Ceph Object Gateway will use default targets and pools (e.g., **`default_placement`**).

Storage policies give Ceph Object Gateway clients a way of accessing a storage strategy--i.e., the ability to target a particular type of storage (e.g., SSDs, SAS drives, SATA drives), a particular way of ensuring durability (replication, erasure coding), etc. To create a storage policy, use the following procedure:

1. Create a new pool **`.rgw.buckets.special`** with the desired storage strategy. For example, a pool customized with erasure-coding, a particular CRUSH ruleset, the number of replicas and the **`pg_num`** and **`pgp_num`** count.

2. Get the region configuration and store it in a file (e.g., **`region.json`**).

```
radosgw-admin region get > region.json
```

3. Add a **`special-placement`** entry under **`placement_target`** in the **`reqion.json`** file.

```
{
 "name": "default",
 "api_name": "",
 "is_master": "true",
 "endpoints": [],
 "hostnames": [],
 "master_zone": "",
 "zones": [{
  "name": "default",
  "endpoints": [],
  "log_meta": "false",
  "log_data": "false",
  "bucket_index_max_shards": 5
 }],
 "placement_targets": [{
  "name": "default-placement",
  "tags": []
 }, {
  "name": "special-placement",
  "tags": []
 }],
 "default_placement": "default-placement"
}
```

4. Set the region with the modified **`region.json`** file.

```
radosgw-admin region set < region.json
```

5. Get the zone configuration and store it in a file (e.g., **`zone.json`**).

```
radosgw-admin zone get > zone.json
```

6. Edit the zone file and add the new placement policy key under **`placement_pool`**.

```
{
 "domain_root": ".rgw",
 "control_pool": ".rgw.control",
 "gc_pool": ".rgw.gc",
 "log_pool": ".log",
 "intent_log_pool": ".intent-log",
 "usage_log_pool": ".usage",
 "user_keys_pool": ".users",
 "user_email_pool": ".users.email",
 "user_swift_pool": ".users.swift",
 "user_uid_pool": ".users.uid",
 "system_key": {
  "access_key": "",
  "secret_key": ""
 },
 "placement_pools": [{
  "key": "default-placement",
  "val": {
   "index_pool": ".rgw.buckets.index",
   "data_pool": ".rgw.buckets",
   "data_extra_pool": ".rgw.buckets.extra"
  }
 }, {
  "key": "special-placement",
  "val": {
   "index_pool": ".rgw.buckets.index",
   "data_pool": ".rgw.buckets.special",
   "data_extra_pool": ".rgw.buckets.extra"
  }
 }]
}
```

7. Set the new zone configuration.

```
radosgw-admin zone set < zone.json
```

8. Update the region map.

```
radosgw-admin regionmap update
```

The **`special-placement`** entry should be listed as a **`placement_target`**.

9. Now restart the Ceph Object Gateway service.

```
sudo systemctl restart ceph-radosgw.service
```

Usage example:

```
curl -i http://10.0.0.1/swift/v1/TestContainer/file.txt -X PUT -H "X-
Storage-Policy: special-placement" -H "X-Auth-Token: AUTH_rgwtxxxxxx"
```

3.3. BUCKET SHARDING

The Ceph Object Gateway stores bucket index data in the index pool (**`index_pool`**), which defaults to **`.rgw.buckets.index`**. If you put many objects (hundreds of thousands to millions of objects) in a single bucket without having set quotas for the maximum number of objects per bucket, the index pool can suffer significant performance degradation.

Bucket index sharding helps prevent performance bottlenecks when allowing a high number of objects per bucket.

See Configuring Bucket Index Sharding for details on configuring bucket index sharding for new buckets.

See Bucket Index Resharding for details on changing the bucket index sharding on already existing buckets.

Configuring Bucket Index Sharding

To enable and configure bucket index sharding on all new buckets, use:

- the **`rgw_override_bucket_index_max_shards`** setting for simple configurations,
- the **`bucket_index_max_shards`** setting for federated configurations.

Set the settings to:

- **`0`** to disable bucket index sharding. This is the default value.
- A value greater than **`0`** to enable bucket sharding and to set the maximum number of shards.

Use the following formula to calculate the recommended number of shards:

```
number of objects expected in a bucket / 100,000
```

Note that maximum number of shards is 7877.

Simple configurations

1. Add **`rgw_override_bucket_index_max_shards`** to the Ceph configuration file:

   ```
   rgw_override_bucket_index_max_shards = 10
   ```

   - To configure bucket index sharding for all instances of the Ceph Object Gateway, add **`rgw_override_bucket_index_max_shards`** under the **`[global]`** section.
   - To configure bucket index sharding only for a particular instance of the Ceph Object Gateway, add **`rgw_override_bucket_index_max_shards`** under the instance.

2. Restart the Ceph Object Gateway:

   ```
   $ sudo service radosgw restart id=rgw.<hostname>
   ```

   Replace **`<hostname>`** with the short host name of the node where the Ceph Object Gateway is running.

Federated configurations

In federated configurations, each zone can have a different **`index_pool`** setting to manage failover. To configure a consistent shard count for zones in one region, set the **`bucket_index_max_shards`** setting in the configuration for that region. To do so:

1. Extract the region configuration to the **`region.json`** file:

   ```
   $ radosgw-admin region get > region.json
   ```

2. In the **`region.json`** file, set the **`bucket_index_max_shards`** setting for each named zone.

3. Reset the region:

   ```
   $ radosgw-admin region set < region.json
   ```

4. Update the region map:

   ```
   radosgw-admin regionmap update --name <name>
   ```

   Replace **`<name>`** with the name of the Ceph Object Gateway user, for example:

   ```
   $ radosgw-admin regionmap update --name client.rgw.ceph-client
   ```

Note

Mapping the index pool (for each zone, if applicable) to a CRUSH ruleset of SSD-based OSDs might also help with bucket index performance.

Bucket Index Resharding

If a bucket has grown larger than the initial configuration was optimized for, reshard the bucket index pool by using the **`radosgw-admin bucket reshard`** command. This command:

- Creates a new set of bucket index objects for the specified object.
- Spreads all objects entries of these index objects.
- Creates a new bucket instance.
- Links the new bucket instance with the bucket so that all new index operations go through the new bucket indexes.
- Prints the old and the new bucket ID to the command output.

To reshard the bucket index pool:

1. Make sure that all operations to the bucket are stopped.

2. Back the original bucket index up:

   ```
   radosgw-admin bi list --bucket=<bucket_name> >
   <bucket_name>.list.backup
   ```

For example, for a bucket named **data**, enter:

```
$ radosgw-admin bi list --bucket=data > data.list.backup
```

3. Reshard the bucket index:

```
radosgw-admin bucket reshard --bucket=<bucket_name>
--num-shards=<new_shards_number>
```

For example, for a bucket named **data** and the required number of shards being 100, enter:

```
$ radosgw-admin bucket reshard --bucket=data
--num-shards=100
```

As part of its output, this command also prints the new and the old bucket ID. Note the old bucket ID down; you will need it to purge the old bucket index objects.

4. Verify that the objects are listed correctly by comparing the old bucket index listing with the new one.

5. Purge the old bucket index objects:

```
radosgw-admin bi purge --bucket=<bucket_name> --bucket-id=
<old_bucket_id>
```

For example, for a bucket named **data** and the old bucket ID being **123456**, enter:

```
$ radosgw-admin bi purge --bucket=data --bucket-id=123456
```

3.4. RADOS GATEWAY USER MANAGEMENT

Ceph Object Storage user management refers to users that are client applications of the Ceph Object Storage service (i.e., not the Ceph Object Gateway as a client application of the Ceph Storage Cluster). You must create a user, access key and secret to enable client applications to interact with the Ceph Object Gateway service.

There are two user types:

- **User:** The term 'user' reflects a user of the S3 interface.
- **Subuser:** The term 'subuser' reflects a user of the Swift interface. A subuser is associated to a user .

You can create, modify, view, suspend and remove users and subusers. In addition to user and subuser IDs, you may add a display name and an email address for a user. You can specify a key and secret, or generate a key and secret automatically. When generating or specifying keys, note that user IDs correspond to an S3 key type and subuser IDs correspond to a swift key type. Swift keys also have access levels of **read**, **write**, **readwrite** and **full**.

User management command-line syntax generally follows the pattern **user <command> <user-id>** where **<user-id>** is either the **--uid=** option followed by the user's ID (S3) or the **--subuser=** option followed by the user name (Swift). For example:

```
radosgw-admin user <create|modify|info|rm|suspend|enable|check|stats>
<--uid={id}|--subuser={name}> [other-options]
```

■

Additional options may be required depending on the command you execute.

3.4.1. Create a User

Use the **`user create`** command to create an S3-interface user. You MUST specify a user ID and a display name. You may also specify an email address. If you DO NOT specify a key or secret, **`radosgw-admin`** will generate them for you automatically. However, you may specify a key and/or a secret if you prefer not to use generated key/secret pairs.

```
radosgw-admin user create --uid=<id> \
[--key-type=<type>] [--gen-access-key|--access-key=<key>]\
[--gen-secret | --secret=<key>] \
[--email=<email>] --display-name=<name>
```

For example:

```
radosgw-admin user create --uid=janedoe --display-name="Jane Doe" --
email=jane@example.com
```

```
{ "user_id": "janedoe",
  "display_name": "Jane Doe",
  "email": "jane@example.com",
  "suspended": 0,
  "max_buckets": 1000,
  "auid": 0,
  "subusers": [],
  "keys": [
        { "user": "janedoe",
          "access_key": "11BS02LGFB6AL6H1ADMW",
          "secret_key": "vzCEkuryfn060dfee4fgQPqFrncKEIkh3ZcdOANY"}],
  "swift_keys": [],
  "caps": [],
  "op_mask": "read, write, delete",
  "default_placement": "",
  "placement_tags": [],
  "bucket_quota": { "enabled": false,
      "max_size_kb": -1,
      "max_objects": -1},
  "user_quota": { "enabled": false,
      "max_size_kb": -1,
      "max_objects": -1},
  "temp_url_keys": []}
```

Important

Check the key output. Sometimes **`radosgw-admin`** generates a JSON escape (**`\`**) character, and some clients do not know how to handle JSON escape characters. Remedies include removing the JSON escape character (**`\`**), encapsulating the string in quotes, regenerating the key and ensuring that it does not have a JSON escape character or specify the key and secret manually.

3.4.2. Create a Subuser

To create a subuser (Swift interface), you must specify the user ID (**`--uid={username}`**), a subuser ID and the access level for the subuser. If you DO NOT specify a key or secret, **`radosgw-admin`** will generate them for you automatically. However, you may specify a key and/or a secret if you prefer not to use generated key/secret pairs.

Note

`full` is not **`readwrite`**, as it also includes the access control policy.

```
radosgw-admin subuser create --uid={uid} --subuser={uid} --access=[
read | write | readwrite | full ]
```

For example:

```
radosgw-admin subuser create --uid=janedoe --subuser=janedoe:swift --
access=full
```

```
{ "user_id": "janedoe",
  "display_name": "Jane Doe",
  "email": "jane@example.com",
  "suspended": 0,
  "max_buckets": 1000,
  "auid": 0,
  "subusers": [
        { "id": "janedoe:swift",
          "permissions": "full-control"}],
  "keys": [
        { "user": "janedoe",
          "access_key": "11BS02LGFB6AL6H1ADMW",
          "secret_key": "vzCEkuryfn060dfee4fgQPqFrncKEIkh3ZcdOANY"}],
  "swift_keys": [],
  "caps": [],
  "op_mask": "read, write, delete",
  "default_placement": "",
  "placement_tags": [],
  "bucket_quota": { "enabled": false,
      "max_size_kb": -1,
      "max_objects": -1},
  "user_quota": { "enabled": false,
      "max_size_kb": -1,
      "max_objects": -1},
  "temp_url_keys": []}
```

3.4.3. Get User Information

To get information about a user, you must specify **`user info`** and the user ID (**`--uid={username}`**) . :

```
radosgw-admin user info --uid=janedoe
```

3.4.4. Modify User Information

To modify information about a user, you must specify the user ID (**`--uid={username}`**) and the attributes you want to modify. Typical modifications are to keys and secrets, email addresses, display names and access levels. For example:

```
radosgw-admin user modify --uid=janedoe --display-name="Jane E. Doe"
```

To modify subuser values, specify **`subuser modify`** and the subuser ID. For example:

```
radosgw-admin subuser modify --uid=janedoe:swift --access=full
```

3.4.5. Enable and Suspend Users

When you create a user, the user is enabled by default. However, you may suspend user privileges and re-enable them at a later time. To suspend a user, specify **`user suspend`** and the user ID. :

```
radosgw-admin user suspend --uid=johndoe
```

To re-enable a suspended user, specify **`user enable`** and the user ID. :

```
radosgw-admin user enable --uid=johndoe
```

Note

Disabling the user disables the subuser.

3.4.6. Remove a User

When you remove a user, the user and subuser are removed from the system. However, you may remove just the subuser if you wish. To remove a user (and subuser), specify **`user rm`** and the user ID.

```
radosgw-admin user rm --uid=<uid> [--purge-keys] [--purge-data]
```

For example:

```
radosgw-admin user rm --uid=johndoe --purge-data
```

To remove the subuser only, specify **`subuser rm`** and the subuser name.

```
radosgw-admin subuser rm --subuser=johndoe:swift --purge-keys
```

Options include:

- **Purge Data:** The **`--purge-data`** option purges all data associated to the UID.
- **Purge Keys:** The **`--purge-keys`** option purges all keys associated to the UID.

3.4.7. Remove a Subuser

When you remove a sub user, you are removing access to the Swift interface. The user will remain in the system. The Ceph Object Gateway To remove the subuser, specify **`subuser rm`** and the subuser ID. :

```
radosgw-admin subuser rm --uid=johndoe:swift
```

Options include:

- **Purge Keys:** The **`--purge-keys`** option purges all keys associated to the UID.

3.4.8. Create a Key

To create a key for a user, you must specify **`key create`**. For a user, specify the user ID and the **`s3`** key type. To create a key for subuser, you must specify the subuser ID and the **`swift`** keytype. For example:

```
radosgw-admin key create --subuser=johndoe:swift --key-type=swift --
gen-secret
```

```
{ "user_id": "johndoe",
  "rados_uid": 0,
  "display_name": "John Doe",
  "email": "john@example.com",
  "suspended": 0,
  "subusers": [
     { "id": "johndoe:swift",
       "permissions": "full-control"}],
  "keys": [
    { "user": "johndoe",
      "access_key": "QFAMEDSJP5DEKJO0DDXY",
      "secret_key": "iaSFLDVvDdQt6lkNzHyW4fPLZugBAI1g17LO0+87"}],
  "swift_keys": [
    { "user": "johndoe:swift",
      "secret_key": "E9T2rUZNu2gxUjcwUBO8n\/Ev4KX6\/GprEuH4qhu1"}]}
```

3.4.9. Add and Remove Access Keys

Users and subusers must have access keys to use the S3 and Swift interfaces. When you create a user or subuser and you do not specify an access key and secret, the key and secret get generated automatically. You may create a key and either specify or generate the access key and/or secret. You may also remove an access key and secret. Options include:

- **`--secret=<key>`** specifies a secret key (e.g,. manually generated).
- **`--gen-access-key`** generates random access key (for S3 user by default).
- **`--gen-secret`** generates a random secret key.
- **`--key-type=<type>`** specifies a key type. The options are: swift, s3

To add a key, specify the user. :

```
radosgw-admin key create --uid=johndoe --key-type=s3 --gen-access-key -
-gen-secret
```

You may also specify a key and a secret.

To remove an access key, specify the user. :

```
radosgw-admin key rm --uid=johndoe
```

3.4.10. Add and Remove Admin Capabilities

The Ceph Storage Cluster provides an administrative API that enables users to execute administrative functions via the REST API. By default, users DO NOT have access to this API. To enable a user to exercise administrative functionality, provide the user with administrative capabilities.

To add administrative capabilities to a user, execute the following:

```
radosgw-admin caps add --uid={uid} --caps={caps}
```

You can add read, write or all capabilities to users, buckets, metadata and usage (utilization). For example:

```
--caps="[users|buckets|metadata|usage|zone]=[*|read|write|read, write]"
```

For example:

```
radosgw-admin caps add --uid=johndoe --caps="users=*"
```

To remove administrative capabilities from a user, execute the following:

```
radosgw-admin caps remove --uid=johndoe --caps={caps}
```

3.5. QUOTA MANAGEMENT

The Ceph Object Gateway enables you to set quotas on users and buckets owned by users. Quotas include the maximum number of objects in a bucket and the maximum storage size in megabytes.

- **Bucket:** The **`--bucket`** option allows you to specify a quota for buckets the user owns.
- **Maximum Objects:** The **`--max-objects`** setting allows you to specify the maximum number of objects. A negative value disables this setting.
- **Maximum Size:** The **`--max-size`** option allows you to specify a quota for the maximum number of bytes. A negative value disables this setting.
- **Quota Scope:** The **`--quota-scope`** option sets the scope for the quota. The options are **`bucket`** and **`user`**. Bucket quotas apply to buckets a user owns. User quotas apply to a user.

Important

Buckets with a large number of objects can cause serious performance issues. The recommended maximum number of objects in a one bucket is 100,000. To increase this number, configure bucket index sharding. See Section 3.3, “Bucket Sharding” for details.

3.5.1. Set User Quotas

Before you enable a quota, you must first set the quota parameters. For example:

```
radosgw-admin quota set --quota-scope=user --uid=<uid> [--max-objects=
<num objects>] [--max-size=<max size>]
```

For example:

```
radosgw-admin quota set --quota-scope=user --uid=johndoe --max-
objects=1024 --max-size=1024
```

A negative value for num objects and / or max size means that the specific quota attribute check is disabled.

3.5.2. Enable and Disable User Quotas

Once you set a user quota, you may enable it. For example:

```
radosgw-admin quota enable --quota-scope=user --uid=<uid>
```

You may disable an enabled user quota. For example:

```
radosgw-admin quota-disable --quota-scope=user --uid=<uid>
```

3.5.3. Set Bucket Quotas

Bucket quotas apply to the buckets owned by the specified **`uid`**. They are independent of the user. :

```
radosgw-admin quota set --uid=<uid> --quota-scope=bucket [--max-
objects=<num objects>] [--max-size=<max size]
```

A negative value for num objects and / or max size means that the specific quota attribute check is disabled.

3.5.4. Enable and Disable Bucket Quotas

Once you set a bucket quota, you may enable it. For example:

```
radosgw-admin quota enable --quota-scope=bucket --uid=<uid>
```

You may disable an enabled bucket quota. For example:

```
radosgw-admin quota-disable --quota-scope=bucket --uid=<uid>
```

3.5.5. Get Quota Settings

You may access each user’s quota settings via the user information API. To read user quota setting information with the CLI interface, execute the following:

```
radosgw-admin user info --uid=<uid>
```

3.5.6. Update Quota Stats

Quota stats get updated asynchronously. You can update quota statistics for all users and all buckets manually to retrieve the latest quota stats. :

```
radosgw-admin user stats --uid=<uid> --sync-stats
```

3.5.7. Get User Quota Usage Stats

To see how much of the quota a user has consumed, execute the following:

```
radosgw-admin user stats --uid=<uid>
```

Note

You should execute **`radosgw-admin user stats`** with the **`--sync-stats`** option to receive the latest data.

3.5.8. Reading and Writing Global Quotas

You can read and write quota settings in a region map. To get a region map:

```
radosgw-admin regionmap get > regionmap.json
```

To set quota settings for the entire region, modify the quota settings in the region map. Then, use the **`regionmap set`** command to update the region map:

```
radosgw-admin regionmap set < regionmap.json
```

Note

After updating the region map, you must restart the gateway.

3.6. USAGE

The Ceph Object Gateway logs usage for each user. You can track user usage within date ranges too.

Options include:

- **Start Date:** The **`--start-date`** option allows you to filter usage stats from a particular start date (**format: `yyyy-mm-dd[HH:MM:SS]`**).
- **End Date:** The **`--end-date`** option allows you to filter usage up to a particular date (**format: `yyyy-mm-dd[HH:MM:SS]`**).
- **Log Entries:** The **`--show-log-entries`** option allows you to specify whether or not to include log entries with the usage stats (options: **`true`** | **`false`**).

Note

You may specify time with minutes and seconds, but it is stored with 1 hour resolution.

3.6.1. Show Usage

To show usage statistics, specify the **`usage show`**. To show usage for a particular user, you must specify a user ID. You may also specify a start date, end date, and whether or not to show log entries.:

```
radosgw-admin usage show --uid=johndoe --start-date=2012-03-01 --end-
date=2012-04-01
```

You may also show a summary of usage information for all users by omitting a user ID. :

```
radosgw-admin usage show --show-log-entries=false
```

3.6.2. Trim Usage

With heavy use, usage logs can begin to take up storage space. You can trim usage logs for all users and for specific users. You may also specify date ranges for trim operations. :

```
radosgw-admin usage trim --start-date=2010-01-01 --end-date=2010-12-31
radosgw-admin usage trim --uid=johndoe
radosgw-admin usage trim --uid=johndoe --end-date=2013-12-31
```

3.6.3. Finding Orphan Objects

Normally, in a healthy storage cluster you should not have any leaking objects, but in some cases leaky objects can occur. For example, if the RADOS Gateway goes down in the middle of an operation, this may cause some RADOS objects to become orphans. Also, unknown bugs may cause these orphan objects to occur. The **`radosgw-admin`** command provides you a tool to search for these orphan objects and clean them up. With the **`--pool`** option, you can specify which pool to scan for leaky RADOS objects. With the **`--num-shards`** option, you may specify the number of shards to use for keeping temporary scan data.

1. Create a new log pool:

   Example

   ```
   rados mkpool .log
   ```

2. Search for orphan objects:

   Syntax

   ```
   radosgw-admin orphans find --pool=<data_pool> --job-id=<job_name>
   [--num-shards=<num_shards>] [--orphan-stale-secs=<seconds>]
   ```

Example

```
radosgw-admin orphans find --pool=.rgw.buckets --job-id=abc123
```

3. Clean up the search data:

Syntax

```
radosgw-admin orphans finish --job-id=<job_name>
```

Example

```
radosgw-admin orphans finish --job-id=abc123
```

CHAPTER 4. OBJECT GATEWAY CONFIGURATION REFERENCE

The following settings may be added to the Ceph configuration file (i.e., usually **`ceph.conf`**) under the **`[client.rgw.{instance-name}]`** section. The settings may contain default values. If you do not specify each setting in the Ceph configuration file, the default value will be set automatically.

Name	Description	Type	Default
`rgw_data`	Sets the location of the data files for Ceph Object Gateway.	String	**`/var/lib/ceph/radosgw/$cluster-$id`**
`rgw_enable_apis`	Enables the specified APIs.	String	**`s3, swift, swift_auth, admin`** All APIs.
`rgw_cache_enabled`	Whether the Ceph Object Gateway cache is enabled.	Boolean	**`true`**
`rgw_cache_lru_size`	The number of entries in the Ceph Object Gateway cache.	Integer	**`10000`**
`rgw_socket_path`	The socket path for the domain socket. **`FastCgiExternalServer`** uses this socket. If you do not specify a socket path, Ceph Object Gateway will not run as an external server. The path you specify here must be the same as the path specified in the **`rgw.conf`** file.	String	N/A
`rgw_host`	The host for the Ceph Object Gateway instance. Can be an IP address or a hostname.	String	**`0.0.0.0`**
`rgw_port`	Port the instance listens for requests. If not specified, Ceph Object Gateway runs external FastCGI.	String	None
`rgw_dns_name`	The DNS name of the served domain. See also the **`hostnames`** setting within regions.	String	None

Name	Description	Type	Default
`rgw_script_uri`	The alternative value for the **`SCRIPT_URI`** if not set in the request.	String	None
`rgw_request_uri`	The alternative value for the **`REQUEST_URI`** if not set in the request.	String	None
`rgw_print_continue`	Enable **`100-continue`** if it is operational.	Boolean	**`true`**
`rgw_remote_addr_param`	The remote address parameter. For example, the HTTP field containing the remote address, or the **`X-Forwarded-For`** address if a reverse proxy is operational.	String	**`REMOTE_ADDR`**
`rgw_op_thread_timeout`	The timeout in seconds for open threads.	Integer	600
`rgw_op_thread_suicide_timeout`	The time **`timeout`** in seconds before a Ceph Object Gateway process dies. Disabled if set to **`0`**.	Integer	**`0`**
`rgw_thread_pool_size`	The size of the thread pool.	Integer	100 threads.
`rgw_num_control_oids`	The number of notification objects used for cache synchronization between different **`rgw`** instances.	Integer	**`8`**
`rgw_init_timeout`	The number of seconds before Ceph Object Gateway gives up on initialization.	Integer	**`30`**
`rgw_mime_types_file`	The path and location of the MIME types. Used for Swift auto-detection of object types.	String	**`/etc/mime.types`**
`rgw_gc_max_objs`	The maximum number of objects that may be handled by garbage collection in one garbage collection processing cycle.	Integer	**`32`**

Name	Description	Type	Default
`rgw_gc_obj_min_wait`	The minimum wait time before the object may be removed and handled by garbage collection processing.	Integer	**`2 * 3600`**
`rgw_gc_processor_max_time`	The maximum time between the beginning of two consecutive garbage collection processing cycles.	Integer	**`3600`**
`rgw_gc_processor_period`	The cycle time for garbage collection processing.	Integer	**`3600`**
`rgw_s3 success_create_obj_status`	The alternate success status response for **`create-obj`**.	Integer	**`0`**
`rgw_resolve_cname`	Whether **`rgw`** should use DNS CNAME record of the request hostname field (if hostname is not equal to **`rgw_dns name`**).	Boolean	**`false`**
`rgw_object_stripe_size`	The size of an object stripe for Ceph Object Gateway objects.	Integer	**`4 << 20`**
`rgw_extended_http_attrs`	Add new set of attributes that could be set on an object. These extra attributes can be set through HTTP header fields when putting the objects. If set, these attributes will return as HTTP fields when doing GET/HEAD on the object.	String	None. For example: "content_foo, content_bar"
`rgw_exit_timeout_secs`	Number of seconds to wait for a process before exiting unconditionally.	Integer	**`120`**

Name	Description	Type	Default
`rgw_get_obj_window_size`	The window size in bytes for a single object request.	Integer	**`16 << 20`**
`rgw_get_obj_max_req_size`	The maximum request size of a single get operation sent to the Ceph Storage Cluster.	Integer	**`4 << 20`**
`rgw_relaxed s3_bucket_names`	Enables relaxed S3 bucket names rules for US region buckets.	Boolean	**`false`**
`rgw_list buckets_max_chunk`	The maximum number of buckets to retrieve in a single operation when listing user buckets.	Integer	**`1000`**
`rgw_num_zone_opstate_shards`	The maximum number of shards for keeping inter-region copy progress information.	Integer	**`128`**
`rgw_opstate_ratelimit_sec`	The minimum time between opstate updates on a single upload. **`0`** disables the ratelimit.	Integer	**`30`**
`rgw_curl_wait_timeout_ms`	The timeout in milliseconds for certain **`curl`** calls.	Integer	**`1000`**
`rgw_copy_obj_progress`	Enables output of object progress during long copy operations.	Boolean	**`true`**
`rgw_copy_obj_progress_every_bytes`	The minimum bytes between copy progress output.	Integer	**`1024 * 1024`**
`rgw_admin_entry`	The entry point for an admin request URL.	String	**`admin`**
`rgw_content_length_compat`	Enable compatability handling of FCGI requests with both CONTENT_LENGTH AND HTTP_CONTENT_LENGTH set.	Boolean	**`false`**

4.1. REGIONS

The Ceph Object Gateway supports federated deployments and a global namespace via the notion of regions. A region defines the geographic location of one or more Ceph Object Gateway instances within one or more zones.

Configuring regions differs from typical configuration procedures, because not all of the settings end up in a Ceph configuration file. You can list regions, get a region configuration and set a region configuration.

4.1.1. List Regions

A Ceph cluster contains a list of regions. To list the regions, execute:

```
sudo radosgw-admin regions list
```

The **`radosgw-admin`** returns a JSON formatted list of regions.

```
{ "default_info": { "default_region": "default"},
  "regions": [
        "default"]}
```

4.1.2. Get a Region Map

To list the details of each region, execute:

```
sudo radosgw-admin region-map get
```

Note

If you receive a **`failed to read region map`** error, run **`sudo radosgw-admin region-map update`** first.

4.1.3. Get a Region

To view the configuration of a region, execute:

```
radosgw-admin region get [--rgw-region=<region>]
```

The **`default`** region looks like this:

```
{"name": "default",
 "api_name": "",
 "is_master": "true",
 "endpoints": [],
 "hostnames": [],
 "master_zone": "",
 "zones": [
   {"name": "default",
    "endpoints": [],
    "log_meta": "false",
    "log_data": "false"}
  ],
 "placement_targets": [
   {"name": "default-placement",
    "tags": [] }],
 "default_placement": "default-placement"}
```

4.1.4. Set a Region

Defining a region consists of creating a JSON object, specifying at least the required settings:

1. **name**: The name of the region. Required.
2. **api_name**: The API name for the region. Optional.
3. **is_master**: Determines if the region is the master region. Required. **note:** You can only have one master region.
4. **endpoints**: A list of all the endpoints in the region. For example, you may use multiple domain names to refer to the same region. Remember to escape the forward slashes (**\/**). You may also specify a port (**fqdn:port**) for each endpoint. Optional.
5. **hostnames**: A list of all the hostnames in the region. For example, you may use multiple domain names to refer to the same region. Optional. The **rgw dns name** setting will automatically be included in this list. You should restart the **radosgw** daemon(s) after changing this setting.
6. **master_zone**: The master zone for the region. Optional. Uses the default zone if not specified. **note:** You can only have one master zone per region.
7. **zones**: A list of all zones within the region. Each zone has a name (required), a list of endpoints (optional), and whether or not the gateway will log metadata and data operations (false by default).
8. **placement_targets**: A list of placement targets (optional). Each placement target contains a name (required) for the placement target and a list of tags (optional) so that only users with the tag can use the placement target (i.e., the user's **placement_tags** field in the user info).
9. **default_placement**: The default placement target for the object index and object data. Set to **default-placement** by default. You may also set a per-user default placement in the user info for each user.

To set a region, create a JSON object consisting of the required fields, save the object to a file (e.g., **region.json**); then, execute the following command:

```
sudo radosgw-admin region set --infile region.json
```

Where **region.json** is the JSON file you created.

Important

The **`default`** region **`is_master`** setting is **`true`** by default. If you create a new region and want to make it the master region, you must either set the **`default`** region **`is_master`** setting to **`false`**, or delete the **`default`** region.

Finally, update the map. :

```
sudo radosgw-admin region-map update
```

4.1.5. Set a Region Map

Setting a region map consists of creating a JSON object consisting of one or more regions, and setting the **`master_region`** for the cluster. Each region in the region map consists of a key/value pair, where the **`key`** setting is equivalent to the **`name`** setting for an individual region configuration, and the **`val`** is a JSON object consisting of an individual region configuration.

You may only have one region with **`is_master`** equal to **`true`**, and it must be specified as the **`master_region`** at the end of the region map. The following JSON object is an example of a default region map.

```
{
    "regions": [
        {
            "key": "default",
            "val": {
                "name": "default",
                "api_name": "",
                "is_master": "true",
                "endpoints": [],
                "hostnames": [],
                "master_zone": "",
                "zones": [
                    {
                        "name": "default",
                        "endpoints": [],
                        "log_meta": "false",
                        "log_data": "false",
                        "bucket_index_max_shards": 0
                    }
                ],
                "placement_targets": [
                    {
                        "name": "default-placement",
                        "tags": []
                    }
                ],
                "default_placement": "default-placement"
            }
        }
    ],
    "master_region": "default",
    "bucket_quota": {
            "enabled": false,
            "max_size_kb": -1,
            "max_objects": -1
        },
        "user_quota": {
            "enabled": false,
            "max_size_kb": -1,
            "max_objects": -1
        }
}
```

To set a region map, execute the following:

```
sudo radosgw-admin region-map set --infile regionmap.json
```

Where **`regionmap.json`** is the JSON file you created. Ensure that you have zones created for the ones specified in the region map. Finally, update the map.

```
sudo radosgw-admin regionmap update
```

4.2. ZONES

Ceph Object Gateway supports the notion of zones. A zone defines a logical group consisting of one or more Ceph Object Gateway instances.

Configuring zones differs from typical configuration procedures, because not all of the settings end up in a Ceph configuration file. You can list zones, get a zone configuration and set a zone configuration.

4.2.1. List Zones

To list the zones in a cluster, execute:

```
sudo radosgw-admin zone list
```

4.2.2. Get a Zone

To get the configuration of a zone, execute:

```
sudo radosgw-admin zone get [--rgw-zone=<zone>]
```

The **`default`** zone looks like this:

```
{ "domain_root": ".rgw",
  "control_pool": ".rgw.control",
  "gc_pool": ".rgw.gc",
  "log_pool": ".log",
  "intent_log_pool": ".intent-log",
  "usage_log_pool": ".usage",
  "user_keys_pool": ".users",
  "user_email_pool": ".users.email",
  "user_swift_pool": ".users.swift",
    "user_uid_pool": ".users.uid",
    "system_key": { "access_key": "", "secret_key": ""},
    "placement_pools": [
        {  "key": "default-placement",
           "val": { "index_pool": ".rgw.buckets.index",
                    "data_pool": ".rgw.buckets"}
        }
      ]
    }
```

4.2.3. Set a Zone

Configuring a zone involves specifying a series of Ceph Object Gateway pools. For consistency, we recommend using a pool prefix that is the same as the zone name. See Pools_ for details of configuring pools.

To set a zone, create a JSON object consisting of the pools, save the object to a file (e.g., **`zone.json`**); then, execute the following command, replacing **`{zone-name}`** with the name of the zone:

```
sudo radosgw-admin zone set --rgw-zone={zone-name} --infile zone.json
```

Where **`zone.json`** is the JSON file you created.

4.3. REGION AND ZONE SETTINGS

When configuring a default region and zone, the pool name typically leaves off the region and zone names, but you may use any naming convention you prefer. The only requirement is to include a leading period to the region or zone name. For example:

- **`.rgw.root`**
- **`.users.swift`**

To change the defaults, include the following settings in your Ceph configuration file under each **`[client.radosgw.{instance-name}]`** instance.

Name	Description	Type	Default
`rgw_zone`	The name of the zone for the gateway instance.	String	None
`rgw_region`	The name of the region for the gateway instance.	String	None
`rgw_default_region_info_oid`	The OID for storing the default region. We do not recommend changing this setting.	String	**`default.region`**

4.4. POOLS

Ceph zones map to a series of Ceph Storage Cluster pools.

Manually Created Pools vs. Generated Pools

If you provide write capabilities to the user key for the Ceph Object Gateway, the gateway has the ability to create pools automatically. This is convenient, but the Ceph Object Storage Cluster uses the default values for the number of placement groups, which doesn't have to be ideal, or the values you specified in the Ceph configuration file. If you allow the Ceph Object Gateway to create pools automatically, ensure that you have reasonable defaults for the number of placement groups.

The default pools for the Ceph Object Gateway's default zone include:

- **`.rgw`**
- **`.rgw.control`**
- **`.rgw.gc`**
- **`.log`**
- **`.intent-log`**
- **`.usage`**
- **`.users`**
- **`.users.email`**
- **`.users.swift`**
- **`.users.uid`**

You have significant discretion in determining how you want a zone to access pools. You can create pools on a per zone basis, or use the same pools for multiple zones. As a best practice, Red Hat recommends having a separate set of pools for the master zone and the secondary zones in each region. When creating pools for a specific zone, consider prepending the region name and zone name to the default pool names. For example:

- **`.region1-zone1.domain.rgw`**
- **`.region1-zone1.rgw.control`**
- **`.region1-zone1.rgw.gc`**
- **`.region1-zone1.log`**
- **`.region1-zone1.intent-log`**
- **`.region1-zone1.usage`**
- **`.region1-zone1.users`**
- **`.region1-zone1.users.email`**
- **`.region1-zone1.users.swift`**
- **`.region1-zone1.users.uid`**

Ceph Object Gateways store data for the bucket index (**`index_pool`**) and bucket data (**`data_pool`**) in placement pools. These might overlap; you can use the same pool for the index and the data. The index pool for default placement is **`.rgw.buckets.index`** and for the data pool for default placement is **`.rgw.buckets`**.

Name	Description	Type	Default
`rgw_region_root_pool`	The pool for storing all region-specific information.	String	**`.rgw.root`**
`rgw_zone_root_pool`	The pool for storing zone-specific information.	String	**`.rgw.root`**

4.5. SWIFT SETTINGS

Name	Description	Type	Default
`rgw_enforce_swift_acls`	Enforces the Swift Access Control List (ACL) settings.	Boolean	**`true`**
`rgw_swift_token_expiration`	The time in seconds for expiring a Swift token.	Integer	**`24 * 3600`**
`rgw_swift_url`	The URL for the Ceph Object Gateway Swift API.	String	None
`rgw_swift_url_prefix`	The URL prefix for the Swift API (e.g., **`http://fqdn.com/swift`**).	**`swift`**	N/A
`rgw_swift_auth_url`	Default URL for verifying v1 auth tokens (if not using internal Swift auth).	String	None
`rgw_swift_auth_entry`	The entry point for a Swift auth URL.	String	**`auth`**

4.6. LOGGING SETTINGS

Name	Description	Type	Default
`rgw_log_nonexistent_bucket`	Enables Ceph Object Gateway to log a request for a non-existent bucket.	Boolean	**`false`**
`rgw_log_object_name`	The logging format for an object name. See manpage date for details about format specifiers.	Date	**`%Y-%m-%d-%H-%i-%n`**
`rgw_log_object_name_utc`	Whether a logged object name includes a UTC time. If **`false`**, it uses the local time.	Boolean	**`false`**
`rgw_usage_max_shards`	The maximum number of shards for usage logging.	Integer	**`32`**
`rgw_usage_max_user_shards`	The maximum number of shards used for a single user's usage logging.	Integer	**`1`**
`rgw_enable_ops_log`	Enable logging for each successful Ceph Object Gateway operation.	Boolean	**`false`**
`rgw_enable_usage_log`	Enable the usage log.	Boolean	**`false`**
`rgw_ops_log_rados`	Whether the operations log should be written to the Ceph Storage Cluster backend.	Boolean	**`true`**
`rgw_ops_log_socket_path`	The Unix domain socket for writing operations logs.	String	None
`rgw_ops_log_data-backlog`	The maximum data backlog data size for operations logs written to a Unix domain socket.	Integer	**`5 << 20`**
`rgw_usage_log_flush_threshold`	The number of dirty merged entries in the usage log before flushing synchronously.	Integer	1024

Name	Description	Type	Default
`rgw_usage_log_tick_interval`	Flush pending usage log data every **`n`** seconds.	Integer	**`30`**
`rgw_intent_log_object_name`	The logging format for the intent log object name. See manpage date for details about format specifiers.	Date	**`%Y-%m-%d-%i-%n`**
`rgw_intent_log_object_name_utc`	Whether the intent log object name includes a UTC time. If **`false`**, it uses the local time.	Boolean	**`false`**
`rgw_data_log_window`	The data log entries window in seconds.	Integer	**`30`**
`rgw_data_log_changes_size`	The number of in-memory entries to hold for the data changes log.	Integer	**`1000`**
`rgw_data_log_num_shards`	The number of shards (objects) on which to keep the data changes log.	Integer	**`128`**
`rgw_data_log_obj_prefix`	The object name prefix for the data log.	String	**`data_log`**
`rgw_replica_log_obj_prefix`	The object name prefix for the replica log.	String	**`replica log`**
`rgw_md_log_max_shards`	The maximum number of shards for the metadata log.	Integer	**`64`**

4.7. KEYSTONE SETTINGS

Name	Description	Type	Default
`rgw_keystone_url`	The URL for the Keystone server.	String	None

Name	Description	Type	Default
`rgw_keystone_admin_token`	The Keystone admin token (shared secret).	String	None
`rgw_keystone_accepted_roles`	The roles requires to serve requests.	String	**`Member, admin`**
`rgw_keystone_token_cache_size`	The maximum number of entries in each Keystone token cache.	Integer	**`10000`**
`rgw_keystone_revocation_interval`	The number of seconds between token revocation checks.	Integer	**`15 * 60`**

CHAPTER 5. OBJECT GATEWAY ADMIN API

The Ceph Object Gateway exposes features of the **`radosgw-admin`** CLI in a RESTful API too. We recommend using the CLI interface when setting up your Ceph Object Gateway. When you want to manage users, data, quotas and usage, the Ceph Object Gateway's Admin API provides a RESTful interface that you can integrate with your management platform(s). Typical Admin API features include:

- Create/Get/Modify/Delete User/Subuser
- User Capabilities Management
- Create/Delete Key
- Get/Trim Usage
- Bucket Operations
  - Get Bucket Info
  - Check Bucket Index
  - Remove Bucket
  - Link/Unlink Bucket
- Object Operations
- Quotas

5.1. CREATING AN ADMIN USER

To use the Ceph Object Gateway Admin API, you must first:

1. Create an object gateway user.

   ```
   radosgw-admin user create --uid="{user-name}" --display-name="
   {Display Name}"
   ```

   For example:

   ```
   radosgw-admin user create --uid="admin-api-user" --display-
   name="Admin API User"
   ```

   The **`radosgw-admin`** CLI will return the user. It will look something like this:

   ```
   {
       "user_id": "admin-api-user",
       "display_name": "Admin API User",
       "email": "",
       "suspended": 0,
       "max_buckets": 1000,
       "auid": 0,
       "subusers": [],
       "keys": [
   ```

```
        {
            "user": "admin-api-user",
            "access_key": "NRWGT19TWMYOB1YDBV1Y",
            "secret_key":
"gr1VEGIV7rxcP3xvXDFCo4UDwwl2YoNrmtRlIAty"
        }
    ],
    "swift_keys": [],
    "caps": [],
    "op_mask": "read, write, delete",
    "default_placement": "",
    "placement_tags": [],
    "bucket_quota": {
        "enabled": false,
        "max_size_kb": -1,
        "max_objects": -1
    },
    "user_quota": {
        "enabled": false,
        "max_size_kb": -1,
        "max_objects": -1
    },
    "temp_url_keys": []
}
```

2. Assign administrative capabilities to the user you create.

```
radosgw-admin caps add --uid="{user-name}" --caps="users=*"
```

For example:

```
radosgw-admin caps add --uid=admin-api-user --caps="users=*"
```

The **`radosgw-admin`** CLI will return the user. The **`"caps":`** will have the capabilities you assigned to the user:

```
{
    "user_id": "admin-api-user",
    "display_name": "Admin API User",
    "email": "",
    "suspended": 0,
    "max_buckets": 1000,
    "auid": 0,
    "subusers": [],
    "keys": [
        {
            "user": "admin-api-user",
            "access_key": "NRWGT19TWMYOB1YDBV1Y",
            "secret_key":
"gr1VEGIV7rxcP3xvXDFCo4UDwwl2YoNrmtRlIAty"
        }
    ],
    "swift_keys": [],
    "caps": [
        {
                "type": "users",
                "perm": "*"
            }
        ],
        "op_mask": "read, write, delete",
        "default_placement": "",
        "placement_tags": [],
        "bucket_quota": {
            "enabled": false,
            "max_size_kb": -1,
            "max_objects": -1
        },
        "user_quota": {
            "enabled": false,
            "max_size_kb": -1,
            "max_objects": -1
        },
        "temp_url_keys": []
    }
```

Now you have a user with administrative privileges.

5.2. AUTHENTICATING REQUESTS

Amazon's S3 service uses the access key and a hash of the request header and the secret key to authenticate the request, which has the benefit of providing an authenticated request (especially large uploads) without SSL overhead.

Most use cases for the S3 API involve using open source S3 clients such as the **`AmazonS3Client`** in the Amazon SDK for Java or Python Boto. These libraries do not support the Ceph Object Gateway Admin API. You can subclass and extend these libraries to support the Ceph Admin API. Alternatively, you may create your own Gateway client.

The CephAdminAPI example class in this section illustrates how to create an **`execute()`** method that can take request parameters, authenticate the request, call the Ceph Admin API and receive a response. **The `CephAdminAPI` class example is not supported or intended for commercial use. It is for illustrative purposes only.** The client code contains five calls to the Ceph Object Gateway to demonstrate CRUD operations:

- Create a User
- Get a User
- Modify a User
- Create a Subuser
- Delete a User

To use this example, you will have to get the Apache HTTP Components, unzip the tar file, navigate to its **`lib`** directory and copy the contents to the **`/jre/lib/ext`** directory of your **`JAVA_HOME`** directory (or a classpath of your choosing).

As you examine the CephAdminAPI class example, notice that the **`execute()`** method takes an HTTP method, a request path, an optional subresource (**`null`** if not specified) and a map of parameters. To execute with subresources (e.g., **`subuser`**, **`key`**, etc.), you will need to specify the subresource as an argument in the **`execute()`** method.

The example method:

1. Builds a URI.
2. Builds an HTTP header string.
3. Instantiates an HTTP request (e.g., **`PUT`**, **`POST`**, **`GET`**, **`DELETE`**).
4. Adds the **`Date`** header to the HTTP header string and the request header.
5. Adds the **`Authorization`** header to the HTTP request header.
6. Instantiates an HTTP client and passes it the instantiated HTTP request.
7. Makes a request.
8. Returns a response.

Building the header string is the portion of the process that involves Amazon’s S3 authentication procedure. Specifically, the example method does the following:

1. Adds a request type (e.g., **`PUT`**, **`POST`**, **`GET`**, **`DELETE`**)
2. Adds the date.
3. Adds the requestPath.

The request type should be upper case with no leading or trailing white space. If you do not trim white space, authentication will fail. The date MUST be expressed in GMT, or authentication will fail.

The exemplary method does not have any other headers. The Amazon S3 authentication procedure sorts **`x-amz`** headers lexicographically. So if you are adding **`x-amz`** headers, be sure to add them lexicographically. See S3 Authentication in this guide for additional details. For a more extensive explanation of the Amazon S3 authentication procedure, consult the Signing and Authenticating REST Requests section of Amazon Simple Storage Service documentation.

Once you have built your header string, the next step is to instantiate an HTTP request and pass it the URI. The examplary method uses **`PUT`** for creating a user and subuser, **`GET`** for getting a user, **`POST`** for modifying a user and **`DELETE`** for deleting a user.

Once you instantiate a request, add the **`Date`** header followed by the **`Authorization`** header. Amazon’s S3 authentication uses the standard **`Authorization`** header, and has the following structure:

```
Authorization: AWS {access-key}:{hash-of-header-and-secret}
```

The CephAdminAPI example class has a **`base64Sha1Hmac()`** method, which takes the header string and the secret key for the admin user, and returns a SHA1 HMAC as a base-64 encoded string. Each **`execute()`** call will invoke the same line of code to build the **`Authorization`** header:

```
httpRequest.addHeader("Authorization", "AWS " + this.getAccessKey() +
":" + base64Sha1Hmac(headerString.toString(), this.getSecretKey()));
```

The following **`CephAdminAPI`** example class requires you to pass the access key, secret key and an endpoint to the constructor. The class provides accessor methods to change them at runtime.

```
import java.io.IOException;
import java.net.URI;
import java.net.URISyntaxException;
import java.time.OffsetDateTime;
import java.time.format.DateTimeFormatter;
import java.time.ZoneId;

import org.apache.http.HttpEntity;
import org.apache.http.NameValuePair;
import org.apache.http.Header;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.client.methods.HttpRequestBase;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.client.methods.HttpPut;
import org.apache.http.client.methods.HttpDelete;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.message.BasicNameValuePair;
import org.apache.http.util.EntityUtils;
import org.apache.http.client.utils.URIBuilder;

import java.util.Base64;
import java.util.Base64.Encoder;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import javax.crypto.spec.SecretKeySpec;
import javax.crypto.Mac;

import java.util.Map;
import java.util.Iterator;
import java.util.Set;
import java.util.Map.Entry;

public class CephAdminAPI {

 /*
  * Each call must specify an access key, secret key, endpoint and
format.
  */
 String accessKey;
 String secretKey;
 String endpoint;
 String scheme = "http"; //http only.
 int port = 80;

 /*
  * A constructor that takes an access key, secret key, endpoint and
format.
  */
 public CephAdminAPI(String accessKey, String secretKey, String
endpoint){
  this.accessKey = accessKey;
  this.secretKey = secretKey;
  this.endpoint = endpoint;
 }
```

```
  /*
   * Accessor methods for access key, secret key, endpoint and format.
   */
  public String getEndpoint(){
   return this.endpoint;
  }

  public void setEndpoint(String endpoint){
   this.endpoint = endpoint;
  }

  public String getAccessKey(){
   return this.accessKey;
  }

  public void setAccessKey(String accessKey){
   this.accessKey = accessKey;
  }

  public String getSecretKey(){
   return this.secretKey;
  }

  public void setSecretKey(String secretKey){
   this.secretKey = secretKey;
  }

  /*
   * Takes an HTTP Method, a resource and a map of arguments and
   * returns a CloseableHTTPResponse.
   */
  public CloseableHttpResponse execute(String HTTPMethod, String
resource,
                                        String subresource, Map
arguments) {

   String httpMethod = HTTPMethod;
   String requestPath = resource;
   StringBuffer request = new StringBuffer();
   StringBuffer headerString = new StringBuffer();
   HttpRequestBase httpRequest;
   CloseableHttpClient httpclient;
   URI uri;
   CloseableHttpResponse httpResponse = null;

   try {

    uri = new URIBuilder()
     .setScheme(this.scheme)
     .setHost(this.getEndpoint())
     .setPath(requestPath)
     .setPort(this.port)
     .build();
     if (subresource != null){
      uri = new URIBuilder(uri)
       .setCustomQuery(subresource)
       .build();
     }


     for (Iterator iter = arguments.entrySet().iterator();
     iter.hasNext();) {
      Entry entry = (Entry)iter.next();
      uri = new URIBuilder(uri)
       .setParameter(entry.getKey().toString(),
                              entry.getValue().toString())
       .build();

     }

     request.append(uri);

     headerString.append(HTTPMethod.toUpperCase().trim() + "\n\n\n");

     OffsetDateTime dateTime = OffsetDateTime.now(ZoneId.of("GMT"));
     DateTimeFormatter formatter = DateTimeFormatter.RFC_1123_DATE_TIME;
     String date = dateTime.format(formatter);

     headerString.append(date + "\n");
     headerString.append(requestPath);

     if (HTTPMethod.equalsIgnoreCase("PUT")){
      httpRequest = new HttpPut(uri);
     } else if (HTTPMethod.equalsIgnoreCase("POST")){
      httpRequest = new HttpPost(uri);
     } else if (HTTPMethod.equalsIgnoreCase("GET")){
      httpRequest = new HttpGet(uri);
     } else if (HTTPMethod.equalsIgnoreCase("DELETE")){
      httpRequest = new HttpDelete(uri);
     } else {
      System.err.println("The HTTP Method must be PUT,
      POST, GET or DELETE.");
      throw new IOException();
     }

     httpRequest.addHeader("Date", date);
     httpRequest.addHeader("Authorization", "AWS " + this.getAccessKey()
     + ":" + base64Sha1Hmac(headerString.toString(),
     this.getSecretKey()));

     httpclient = HttpClients.createDefault();
     httpResponse = httpclient.execute(httpRequest);

    }  catch  (URISyntaxException e){
     System.err.println("The URI is not formatted properly.");
     e.printStackTrace();
    }  catch (IOException e){
     System.err.println("There was an error making the request.");
     e.printStackTrace();
    }
     return httpResponse;
   }

   /*
    * Takes a uri and a secret key and returns a base64-encoded
    * SHA-1 HMAC.
    */
   public String base64Sha1Hmac(String uri, String secretKey) {
    try {

     byte[] keyBytes = secretKey.getBytes("UTF-8");
     SecretKeySpec signingKey = new SecretKeySpec(keyBytes, "HmacSHA1");

     Mac mac = Mac.getInstance("HmacSHA1");
     mac.init(signingKey);

     byte[] rawHmac = mac.doFinal(uri.getBytes("UTF-8"));

     Encoder base64 = Base64.getEncoder();
     return base64.encodeToString(rawHmac);

    } catch (Exception e) {
     throw new RuntimeException(e);
    }
   }

  }
```

The subsequent **`CephAdminAPIClient`** example illustrates how to instantiate the **`CephAdminAPI`** class, build a map of request parameters, and use the **`execute()`** method to create, get, update and delete a user.

```
import java.io.IOException;
import org.apache.http.client.methods.CloseableHttpResponse;
import org.apache.http.HttpEntity;
import org.apache.http.util.EntityUtils;
import java.util.*;


public class CephAdminAPIClient {

 public static void main (String[] args){

  CephAdminAPI adminApi = new CephAdminAPI ("FFC6ZQ6EMIF64194158N",

"Xac39eCAhlTGcCAUreuwe1ZuH5oVQFa51lbEMVoT",
                              "ceph-client");

  /*
   * Create a user
   */
  Map requestArgs = new HashMap();
  requestArgs.put("access", "usage=read, write; users=read, write");
  requestArgs.put("display-name", "New User");
  requestArgs.put("email", "new-user@email.com");
  requestArgs.put("format", "json");
  requestArgs.put("uid", "new-user");

  CloseableHttpResponse response =
   adminApi.execute("PUT", "/admin/user", null, requestArgs);

  System.out.println(response.getStatusLine());
  HttpEntity entity = response.getEntity();

  try {
   System.out.println("\nResponse Content is: "
    + EntityUtils.toString(entity, "UTF-8") + "\n");
   response.close();
  } catch (IOException e){
   System.err.println ("Encountered an I/O exception.");
   e.printStackTrace();
  }

  /*
   * Get a user
   */
  requestArgs = new HashMap();
  requestArgs.put("format", "json");
  requestArgs.put("uid", "new-user");

  response = adminApi.execute("GET", "/admin/user", null, requestArgs);

  System.out.println(response.getStatusLine());
  entity = response.getEntity();

  try {
   System.out.println("\nResponse Content is: "
    + EntityUtils.toString(entity, "UTF-8") + "\n");
   response.close();
  } catch (IOException e){
   System.err.println ("Encountered an I/O exception.");
   e.printStackTrace();
  }

  /*
   * Modify a user
   */
  requestArgs = new HashMap();
  requestArgs.put("display-name", "John Doe");
  requestArgs.put("email", "johndoe@email.com");
  requestArgs.put("format", "json");
  requestArgs.put("uid", "new-user");
  requestArgs.put("max-buckets", "100");

  response = adminApi.execute("POST", "/admin/user", null,
requestArgs);

  System.out.println(response.getStatusLine());
  entity = response.getEntity();

  try {
     System.out.println("\nResponse Content is: "
      + EntityUtils.toString(entity, "UTF-8") + "\n");
     response.close();
    } catch (IOException e){
     System.err.println ("Encountered an I/O exception.");
     e.printStackTrace();
    }


    /*
     * Create a subuser
     */
    requestArgs = new HashMap();
    requestArgs.put("format", "json");
    requestArgs.put("uid", "new-user");
    requestArgs.put("subuser", "foobar");

    response = adminApi.execute("PUT", "/admin/user", "subuser",
  requestArgs);
    System.out.println(response.getStatusLine());
    entity = response.getEntity();

    try {
     System.out.println("\nResponse Content is: "
      + EntityUtils.toString(entity, "UTF-8") + "\n");
     response.close();
    } catch (IOException e){
     System.err.println ("Encountered an I/O exception.");
     e.printStackTrace();
    }


    /*
     * Delete a user
     */
    requestArgs = new HashMap();
    requestArgs.put("format", "json");
    requestArgs.put("uid", "new-user");

    response = adminApi.execute("DELETE", "/admin/user", null,
  requestArgs);
    System.out.println(response.getStatusLine());
    entity = response.getEntity();

    try {
     System.out.println("\nResponse Content is: "
      + EntityUtils.toString(entity, "UTF-8") + "\n");
     response.close();
    } catch (IOException e){
     System.err.println ("Encountered an I/O exception.");
     e.printStackTrace();
    }
   }
  }
```

5.3. ADMIN OPERATIONS API

5.3. ADMIN OPERATIONS API

An admin API request will be done on a URI that starts with the configurable 'admin' resource entry point. Authorization for the admin API duplicates the S3 authorization mechanism. Some operations require that the user holds special administrative capabilities. The response entity type (XML or JSON) may be specified as the 'format' option in the request and defaults to JSON if not specified.

5.3.1. Get Usage

Request bandwidth usage information.

caps

`usage=read`

5.3.1.1. Syntax

```
GET /admin/usage?format=json HTTP/1.1
Host: {fqdn}
```

5.3.1.2. Request Parameters

Name	Description	Type	Required
`uid`	The user for which the information is requested.	String.	Yes
`start`	Date and (optional) time that specifies the start time of the requested data. E.g., **`2012-09-25 16:00:00`**	String	No
`end`	Date and (optional) time that specifies the end time of the requested data (non-inclusive). E.g., **`2012-09-25 16:00:00`**	String	No
`show-entries`	Specifies whether data entries should be returned.	Boolean	No
`show-summary`	Specifies whether data summary should be returned.	Boolean	No

5.3.1.3. Response Entities

If successful, the response contains the requested information.

Name	Description	Type
`usage`	A container for the usage information.	Container
`entries`	A container for the usage entries information.	Container
`user`	A container for the user data information.	Container
`owner`	The name of the user that owns the buckets.	String
`bucket`	The bucket name.	String
`time`	Time lower bound for which data is being specified (rounded to the beginning of the first relevant hour).	String
`epoch`	The time specified in seconds since `1/1/1970`.	String
`categories`	A container for stats categories.	Container
`entry`	A container for stats entry.	Container
`category`	Name of request category for which the stats are provided.	String
`bytes_sent`	Number of bytes sent by the Ceph Object Gateway.	Integer
`bytes_received`	Number of bytes received by the Ceph Object Gateway.	Integer
`ops`	Number of operations.	Integer
`successful_ops`	Number of successful operations.	Integer

Name	Description	Type
`summary`	A container for stats summary.	Container
`total`	A container for stats summary aggregated total.	Container

5.3.2. Trim Usage

Remove usage information. With no dates specified, removes all usage information.

caps

   `usage=write`

5.3.2.1. Syntax

```
DELETE /admin/usage?format=json HTTP/1.1
Host: {fqdn}
```

5.3.2.2. Request Parameters

Name	Description	Type	Example	Required
`uid`	The user for which the information is requested.	String	**`foo_user`**	No
`start`	Date and (optional) time that specifies the start time of the requested data.	String	**`2012-09-25 16:00:00`**	No
`end`	Date and (optional) time that specifies the end time of the requested data (none inclusive).	String	**`2012-09-25 16:00:00`**	No
`remove-all`	Required when **`uid`** is not specified, in order to acknowledge multi-user data removal.	Boolean	True [False]	No

5.3.3. Get User Information

Get user information.

caps

users=read

5.3.3.1. Syntax

```
GET /admin/user?format=json HTTP/1.1
Host: {fqdn}
```

5.3.3.2. Request Parameters

Name	Description	Type	Example	Required
`uid`	The user for which the information is requested.	String	`foo_user`	Yes

5.3.3.3. Response Entities

If successful, the response contains the user information.

Name	Description	Type	Parent
`user`	A container for the user data information.	Container	N/A
`user_id`	The user ID.	String	`user`
`display_name`	Display name for the user.	String	`user`
`suspended`	True if the user is suspended.	Boolean	`user`
`max_buckets`	The maximum number of buckets to be owned by the user.	Integer	`user`
`subusers`	Subusers associated with this user account.	Container	`user`

Name	Description	Type	Parent
`keys`	S3 keys associated with this user account.	Container	**`user`**
`swift_keys`	Swift keys associated with this user account.	Container	**`user`**
`caps`	User capabilities.	Container	**`user`**

5.3.3.4. Special Error Responses

None.

5.3.4. Creating a User

Create a new user. By Default, a S3 key pair will be created automatically and returned in the response. If only one of **`access-key`** or **`secret-key`** is provided, the omitted key will be automatically generated. By default, a generated key is added to the keyring without replacing an existing key pair. If **`access-key`** is specified and refers to an existing key owned by the user then it will be modified.

caps

`users=write`

5.3.4.1. Syntax

```
PUT /admin/user?format=json HTTP/1.1
Host: {fqdn}
```

5.3.4.2. Request Parameters

Name	Description	Type	Example	Required
`uid`	The user ID to be created.	String	**`foo_user`**	Yes
`display-name`	The display name of the user to be created.	String	**`foo user`**	Yes

Name	Description	Type	Example	Required
`email`	The email address associated with the user.	String	`foo@bar.com`	No
`key-type`	Key type to be generated, options are: swift, s3 (default).	String	`s3` [`s3`]	No
`access-key`	Specify access key.	String	`ABCD0EF12GHIJ2K34LMN`	No
`secret-key`	Specify secret key.	String	`0AbCDEFg1h2i34JklM5nop6QrSTUV+WxyzaBC7D8`	No
`user-caps`	User capabilities.	String	`usage=read, write; users=read`	No
`generate-key`	Generate a new key pair and add to the existing keyring.	Boolean	True [True]	No
`max-buckets`	Specify the maximum number of buckets the user can own.	Integer	500 [1000]	No
`suspended`	Specify whether the user should be suspended.	Boolean	False [False]	No

5.3.4.3. Response Entities

If successful, the response contains the user information.

Name	Description	Type	Parent
`user`	A container for the user data information.	Container	N/A

Name	Description	Type	Parent
`user_id`	The user ID.	String	`user`
`display_name`	Display name for the user.	String	`user`
`suspended`	True if the user is suspended.	Boolean	`user`
`max_buckets`	The maximum number of buckets to be owned by the user.	Integer	`user`
`subusers`	Subusers associated with this user account.	Container	`user`
`keys`	S3 keys associated with this user account.	Container	`user`
`swift_keys`	Swift keys associated with this user account.	Container	`user`
`caps`	User capabilities.	Container	`user`

5.3.4.4. Special Error Responses

Name	Description	Code
`UserExists`	Attempt to create existing user.	409 Conflict
`InvalidAccessKey`	Invalid access key specified.	400 Bad Request

Name	Description	Code
`InvalidKeyType`	Invalid key type specified.	400 Bad Request
`InvalidSecretKey`	Invalid secret key specified.	400 Bad Request
`InvalidKeyType`	Invalid key type specified.	400 Bad Request
`KeyExists`	Provided access key exists and belongs to another user.	409 Conflict
`EmailExists`	Provided email address exists.	409 Conflict
`InvalidCap`	Attempt to grant invalid admin capability.	400 Bad Request

5.3.5. Modifying a User

Modify a user.

caps

`users=write`

5.3.5.1. Syntax

```
POST /admin/user?format=json HTTP/1.1
Host: {fqdn}
```

5.3.5.2. Request Parameters

Name	Description	Type	Example	Required
`uid`	The user ID to be modified.	String	**`foo_user`**	Yes

Name	Description	Type	Example	Required
`display-name`	The display name of the user to be modified.	String	`foo user`	No
`email`	The email address to be associated with the user.	String	`foo@bar.com`	No
`generate-key`	Generate a new key pair and add to the existing keyring.	Boolean	True [False]	No
`access-key`	Specify access key.	String	`ABCD0EF12GHIJ2K34LMN`	No
`secret-key`	Specify secret key.	String	`0AbCDEFg1h2i34JklM5nop6QrSTUV+WxyzaBC7D8`	No
`key-type`	Key type to be generated, options are: swift, s3 (default).	String	`s3`	No
`user-caps`	User capabilities.	String	`usage=read, write; users=read`	No
`max-buckets`	Specify the maximum number of buckets the user can own.	Integer	500 [1000]	No
`suspended`	Specify whether the user should be suspended.	Boolean	False [False]	No

5.3.5.3. Response Entities

If successful, the response contains the user information.

Name	Description	Type	Parent
`user`	A container for the user data information.	Container	N/A
`user_id`	The user ID.	String	**`user`**
`display_name`	Display name for the user.	String	**`user`**
`suspended`	True if the user is suspended.	Boolean	**`user`**
`max_buckets`	The maximum number of buckets to be owned by the user.	Integer	**`user`**
`subusers`	Subusers associated with this user account.	Container	**`user`**
`keys`	S3 keys associated with this user account.	Container	**`user`**
`swift_keys`	Swift keys associated with this user account.	Container	**`user`**
`caps`	User capabilities.	Container	**`user`**

5.3.5.4. Special Error Responses

Name	Description	Code
`InvalidAccessKey`	Invalid access key specified.	400 Bad Request

Name	Description	Code
`InvalidKeyType`	Invalid key type specified.	400 Bad Request
`InvalidSecretKey`	Invalid secret key specified.	400 Bad Request
`KeyExists`	Provided access key exists and belongs to another user.	409 Conflict
`EmailExists`	Provided email address exists.	409 Conflict
`InvalidCap`	Attempt to grant invalid admin capability.	400 Bad Request

5.3.6. Removing a User

Remove an existing user.

caps

  `users=write`

5.3.6.1. Syntax

```
DELETE /admin/user?format=json HTTP/1.1
Host: {fqdn}
```

5.3.6.2. Request Parameters

Name	Description	Type	Example	Required
`uid`	The user ID to be removed.	String	**`foo_user`**	Yes.
`purge-data`	When specified the buckets and objects belonging to the user will also be removed.	Boolean	True	No

5.3.6.3. Response Entities

None

5.3.6.4. Special Error Responses

None.

5.3.7. Creating a Subuser

Create a new subuser (primarily useful for clients using the Swift API). Note that either **`gen-subuser`** or **`subuser`** is required for a valid request. Note that in general for a subuser to be useful, it must be granted permissions by specifying **`access`**. As with user creation if **`subuser`** is specified without **`secret`**, then a secret key will be automatically generated.

caps

`users=write`

5.3.7.1. Syntax

```
PUT /admin/user?subuser&format=json HTTP/1.1
Host {fqdn}
```

5.3.7.2. Request Parameters

Name	Description	Type	Example	Required
`uid`	The user ID under which a subuser is to be created.	String	**`foo_user`**	Yes
`subuser`	Specify the subuser ID to be created.	String	**`sub_foo`**	Yes (or **`gen-subuser`**)
`gen-subuser`	Specify the subuser ID to be created.	String	**`sub_foo`**	Yes (or **`subuser`**)
`secret-key`	Specify secret key.	String	**`0AbCDEFg1h2i34JklM5nop6QrSTUVWxyzaBC7D8`**	No

Name	Description	Type	Example	Required
`key-type`	Key type to be generated, options are: swift (default), s3.	String	**`swift`** [**`swift`**]	No
`access`	Set access permissions for sub-user, should be one of **`read, write, readwrite, full`**.	String	**`read`**	No
`generate-secret`	Generate the secret key.	Boolean	True [False]	No

5.3.7.3. Response Entities

If successful, the response contains the subuser information.

Name	Description	Type	Parent
`subusers`	Subusers associated with the user account.	Container	N/A
`id`	Subuser ID.	String	**`subusers`**
`permissions`	Subuser access to user account.	String	**`subusers`**

5.3.7.4. Special Error Responses

Name	Description	Code
`SubuserExists`	Specified subuser exists.	409 Conflict

Name	Description	Code
`InvalidKeyType`	Invalid key type specified.	400 Bad Request
`InvalidSecretKey`	Invalid secret key specified.	400 Bad Request
`InvalidAccess`	Invalid subuser access specified.	400 Bad Request

5.3.8. Modifying a Subuser

Modify an existing subuser

caps

`users=write`

5.3.8.1. Syntax

```
POST /admin/user?subuser&format=json HTTP/1.1
Host {fqdn}
```

5.3.8.2. Request Parameters

Name	Description	Type	Example	Required
`uid`	The user ID under which the subuser is to be modified.	String	**`foo_user`**	Yes
`subuser`	The subuser ID to be modified.	String	**`sub_foo`**	Yes
`generate-secret`	Generate a new secret key for the subuser, replacing the existing key.	Boolean	True [False]	No

Name	Description	Type	Example	Required
`secret`	Specify secret key.	String	**`0AbCDEFg1h2i34JklM5nop6QrSTUV+WxyzaBC7D8`**	No
`key-type`	Key type to be generated, options are: swift (default), s3.	String	**`swift`** [**`swift`**]	No
`access`	Set access permissions for sub-user, should be one of **`read, write, readwrite, full`**.	String	**`read`**	No

5.3.8.3. Response Entities

If successful, the response contains the subuser information.

Name	Description	Type	Parent
`subusers`	Subusers associated with the user account.	Container	N/A
`id`	Subuser ID.	String	**`subusers`**
`permissions`	Subuser access to user account.	String	**`subusers`**

5.3.8.4. Special Error Responses

Name	Description	Code
`InvalidKeyType`	Invalid key type specified.	400 Bad Request

Name	Description	Code
`InvalidSecretKey`	Invalid secret key specified.	400 Bad Request
`InvalidAccess`	Invalid subuser access specified.	400 Bad Request

5.3.9. Removing a Subuser

Remove an existing subuser

`caps`

`users=write`

5.3.9.1. Syntax

```
DELETE /admin/user?subuser&format=json HTTP/1.1
Host {fqdn}
```

5.3.9.2. Request Parameters

Name	Description	Type	Example	Required
`uid`	The user ID under which the subuser is to be removed.	String	**`foo_user`**	Yes
`subuser`	The subuser ID to be removed.	String	**`sub_foo`**	Yes
`purge-keys`	Remove keys belonging to the subuser.	Boolean	True [True]	No

5.3.9.3. Response Entities

None.

5.3.9.4. Special Error Responses

None.

5.3.10. Creating a Key

Create a new key. If a **`subuser`** is specified then by default created keys will be swift type. If only one of **`access-key`** or **`secret-key`** is provided the committed key will be automatically generated, that is if only **`secret-key`** is specified then **`access-key`** will be automatically generated. By default, a generated key is added to the keyring without replacing an existing key pair. If **`access-key`** is specified and refers to an existing key owned by the user then it will be modified. The response is a container listing all keys of the same type as the key created. Note that when creating a swift key, specifying the option **`access-key`** will have no effect. Additionally, only one swift key may be held by each user or subuser.

caps

`users=write`

5.3.10.1. Syntax

```
PUT /admin/user?key&format=json HTTP/1.1
Host {fqdn}
```

5.3.10.2. Request Parameters

Name	Description	Type	Example	Required
`uid`	The user ID to receive the new key.	String	**`foo_user`**	Yes
`subuser`	The subuser ID to receive the new key.	String	**`sub_foo`**	No
`key-type`	Key type to be generated, options are: swift, s3 (default).	String	**`s3`** [**`s3`**]	No
`access-key`	Specify the access key.	String	**`AB01C2D3EF45G6H7IJ8K`**	No
`secret-key`	Specify the secret key.	String	**`0ab/CdeFGhij1klmnopqRSTUv1WxyZabcDEFgHij`**	No

Name	Description	Type	Example	Required
`generate-key`	Generate a new key pair and add to the existing keyring.	Boolean	True [`True`]	No

5.3.10.3. Response Entities

Name	Description	Type	Parent
`keys`	Keys of type created associated with this user account.	Container	N/A
`user`	The user account associated with the key.	String	`keys`
`access-key`	The access key.	String	`keys`
`secret-key`	The secret key	String	`keys`

5.3.10.4. Special Error Responses

Name	Description	Code
`InvalidAccessKey`	Invalid access key specified.	400 Bad Request
`InvalidKeyType`	Invalid key type specified.	400 Bad Request
`InvalidSecretKey`	Invalid secret key specified.	400 Bad Request

Name	Description	Code
InvalidKeyType	Invalid key type specified.	400 Bad Request
KeyExists	Provided access key exists and belongs to another user.	409 Conflict

5.3.11. Removing a Key

Remove an existing key.

caps

  users=write

5.3.11.1. Syntax

```
DELETE /admin/user?key&format=json HTTP/1.1
Host {fqdn}
```

5.3.11.2. Request Parameters

Name	Description	Type	Example	Required
access-key	The S3 access key belonging to the S3 key pair to remove.	String	**AB01C2D3EF45G6H7IJ8K**	Yes
uid	The user to remove the key from.	String	**foo_user**	No
subuser	The subuser to remove the key from.	String	**sub_foo**	No
key-type	Key type to be removed, options are: swift, s3. NOTE: Required to remove swift key.	String	**swift**	No

5.3.11.3. Special Error Responses

None.

5.3.11.4. Response Entities

None.

5.3.12. Getting Bucket Information

Get information about a subset of the existing buckets. If **`uid`** is specified without **`bucket`** then all buckets beloning to the user will be returned. If **`bucket`** alone is specified, information for that particular bucket will be retrieved.

caps

`buckets=read`

5.3.12.1. Syntax

```
GET /admin/bucket?format=json HTTP/1.1
Host {fqdn}
```

5.3.12.2. Request Parameters

Name	Description	Type	Example	Required
`bucket`	The bucket to return info on.	String	**`foo_bucket`**	No
`uid`	The user to retrieve bucket information for.	String	**`foo_user`**	No
`stats`	Return bucket statistics.	Boolean	True [False]	No

5.3.12.3. Response Entities

If successful the request returns a buckets container containing the desired bucket information.

Name	Description	Type	Parent
`stats`	Per bucket information.	Container	N/A

Name	Description	Type	Parent
`buckets`	Contains a list of one or more bucket containers.	Container	**`bucket`**
Container for single bucket information.	Container	**`buckets`**	**`name`**
The name of the bucket.	String	**`bucket`**	**`pool`**
The pool the bucket is stored in.	String	**`bucket`**	**`id`**
The unique bucket ID.	String	**`bucket`**	**`marker`**
Internal bucket tag.	String	**`bucket`**	**`owner`**
The user ID of the bucket owner.	String	**`bucket`**	**`usage`**
Storage usage information.	Container	**`bucket`**	**`index`**

5.3.12.4. Special Error Responses

Name	Description	Code
`IndexRepairFailed`	Bucket index repair failed.	409 Conflict

5.3.13. Checking a Bucket Index

Check the index of an existing bucket. NOTE: to check multipart object accounting with **`check-objects`**, **`fix`** must be set to True.

caps

`buckets=write`

5.3.13.1. Syntax

```
GET /admin/bucket?index&format=json HTTP/1.1
Host {fqdn}
```

5.3.13.2. Request Parameters

Name	Description	Type	Example	Required
`bucket`	The bucket to return info on.	String	**`foo_bucket`**	Yes
`check-objects`	Check multipart object accounting.	Boolean	True [False]	No
`fix`	Also fix the bucket index when checking.	Boolean	False [False]	No

5.3.13.3. Response Entities

Name	Description	Type
`index`	Status of bucket index.	String

5.3.13.4. Special Error Responses

Name	Description	Code
`IndexRepairFailed`	Bucket index repair failed.	409 Conflict

5.3.14. Removing a Bucket

Delete an existing bucket.

caps

`buckets=write`

5.3.14.1. Syntax

```
DELETE /admin/bucket?format=json HTTP/1.1
Host {fqdn}
```

5.3.14.2. Request Parameters

Name	Description	Type	Example	Required
`bucket`	The bucket to remove.	String	**`foo_bucket`**	Yes
`purge-objects`	Remove a buckets objects before deletion.	Boolean	True [False]	No

5.3.14.3. Response Entities

None.

5.3.14.4. Special Error Responses

Name	Description	Code
`BucketNotEmpty`	Attempted to delete non-empty bucket.	409 Conflict
`ObjectRemovalFailed`	Unable to remove objects.	409 Conflict

5.3.15. Unlinking a Bucket

Unlink a bucket from a specified user. Primarily useful for changing bucket ownership.

caps

`buckets=write`

5.3.15.1. Syntax

```
POST /admin/bucket?format=json HTTP/1.1
Host {fqdn}
```

5.3.15.2. Request Parameters

Name	Description	Type	Example	Required
`bucket`	The bucket to unlink.	String	**`foo_bucket`**	Yes
`uid`	The user ID to unlink the bucket from.	String	**`foo_user`**	Yes

5.3.15.3. Response Entities

None.

5.3.15.4. Special Error Responses

Name	Description	Code
`BucketUnlinkFailed`	Unable to unlink bucket from specified user.	409 Conflict

5.3.16. Linking a Bucket

Link a bucket to a specified user, unlinking the bucket from any previous user.

caps

`buckets=write`

5.3.16.1. Syntax

```
PUT /admin/bucket?format=json HTTP/1.1
Host {fqdn}
```

5.3.16.2. Request Parameters

Name	Description	Type	Example	Required
`bucket`	The bucket to unlink.	String	**`foo_bucket`**	Yes
`uid`	The user ID to link the bucket to.	String	**`foo_user`**	Yes

5.3.16.3. Response Entities

Name	Description	Type	Parent
`bucket`	Container for single bucket information.	Container	N/A
`name`	The name of the bucket.	String	**`bucket`**
`pool`	The pool the bucket is stored in.	String	**`bucket`**
`id`	The unique bucket ID.	String	**`bucket`**
`marker`	Internal bucket tag.	String	**`bucket`**
`owner`	The user ID of the bucket owner.	String	**`bucket`**
`usage`	Storage usage information.	Container	**`bucket`**
`index`	Status of bucket index.	String	**`bucket`**

5.3.16.4. Special Error Responses

Name	Description	Code
BucketUnlinkFailed	Unable to unlink bucket from specified user.	409 Conflict
BucketLinkFailed	Unable to link bucket to specified user.	409 Conflict

5.3.17. Removing an Object

Remove an existing object. NOTE: Does not require owner to be non-suspended.

caps

`buckets=write`

5.3.17.1. Syntax

```
DELETE /admin/bucket?object&format=json HTTP/1.1
Host {fqdn}
```

5.3.17.2. Request Parameters

Name	Description	Type	Example	Required
bucket	The bucket containing the object to be removed.	String	**foo_bucket**	Yes
object	The object to remove.	String	**foo.txt**	Yes

5.3.17.3. Response Entities

None.

5.3.17.4. Special Error Responses

Name	Description	Code
NoSuchObject	Specified object does not exist.	404 Not Found

Name	Description	Code
`ObjectRemovalFailed`	Unable to remove objects.	409 Conflict

5.3.18. Getting Bucket or Object Policy

Read the policy of an object or bucket.

caps

`buckets=read`

5.3.18.1. Syntax

```
GET /admin/bucket?policy&format=json HTTP/1.1
Host {fqdn}
```

5.3.18.2. Request Parameters

Name	Description	Type	Example	Required
`bucket`	The bucket to read the policy from.	String	**`foo_bucket`**	Yes
`object`	The object to read the policy from.	String	**`foo.txt`**	No

5.3.18.3. Response Entities

If successful, returns the object or bucket policy

|**`policy`** | Access control policy.|Container

5.3.18.4. Special Error Responses

Name	Description	Code

Name	Description	Code
IncompleteBody	Either bucket was not specified for a bucket policy request or bucket and object were not specified for an object policy request.	400 Bad Request

5.3.19. Adding a Capability to an Existing User

Add an administrative capability to a specified user.

caps

`users=write`

5.3.19.1. Syntax

```
PUT /admin/user?caps&format=json HTTP/1.1
Host {fqdn}
```

5.3.19.2. Request Parameters

Name	Description	Type	Example	Required
uid	The user ID to add an administrative capability to.	String	**foo_user**	Yes
user-caps	The administrative capability to add to the user.	String	**usage=read, write**	Yes

5.3.19.3. Response Entities

If successful, the response contains the user’s capabilities.

Name	Description	Type	Parent
user	A container for the user data information.	Container	N/A
user_id	The user ID.	String	**user**

Name	Description	Type	Parent
`caps`	User capabilities.	Container	**`user`**

5.3.19.4. Special Error Responses

Name	Description	Code
`InvalidCap`	Attempt to grant invalid admin capability.	400 Bad Request

5.3.19.5. Example Request

```
PUT /admin/user?caps&format=json HTTP/1.1
Host: {fqdn}
Content-Type: text/plain
Authorization: {your-authorization-token}

usage=read
```

5.3.20. Removing a Capability from an Existing User

Remove an administrative capability from a specified user.

caps

`users=write`

5.3.20.1. Syntax

```
DELETE /admin/user?caps&format=json HTTP/1.1
Host {fqdn}
```

5.3.20.2. Request Parameters

Name	Description	Type	Example	Required
`uid`	The user ID to remove an administrative capability from.	String	**`foo_user`**	Yes

Name	Description	Type	Example	Required
`user-caps`	The administrative capabilities to remove from the user.	String	`usage=read, write`	Yes

5.3.20.3. Response Entities

If successful, the response contains the user's capabilities.

Name	Description	Type	Parent
`user`	A container for the user data information.	Container	N/A
`user_id`	The user ID.	String	`user`
`caps`	User capabilities.	Container	`user`

5.3.20.4. Special Error Responses

Name	Description	Code
`InvalidCap`	Attempt to remove an invalid admin capability.	400 Bad Request
`NoSuchCap`	User does not possess specified capability.	404 Not Found

5.3.20.5. Special Error Responses

None.

5.3.21. Quotas

The Admin Operations API enables you to set quotas on users and on bucket owned by users. See Quota Management for additional details. Quotas include the maximum number of objects in a bucket and the maximum storage size in megabytes.

To view quotas, the user must have a **`users=read`** capability. To set, modify or disable a quota, the user must have **`users=write`** capability. See the Administration (CLI) for details.

Valid parameters for quotas include:

- **Bucket:** The **`bucket`** option allows you to specify a quota for buckets owned by a user.
- **Maximum Objects:** The **`max-objects`** setting allows you to specify the maximum number of objects. A negative value disables this setting.
- **Maximum Size:** The **`max-size`** option allows you to specify a quota for the maximum number of bytes. A negative value disables this setting.
- **Quota Scope:** The **`quota-scope`** option sets the scope for the quota. The options are **`bucket`** and **`user`**.

5.3.21.1. Getting User Quota

To get a quota, the user must have **`users`** capability set with **`read`** permission.

```
GET /admin/user?quota&uid=<uid>&quota-type=user
```

5.3.21.2. Setting User Quota

To set a quota, the user must have **`users`** capability set with **`write`** permission.

```
PUT /admin/user?quota&uid=<uid>&quota-type=user
```

The content must include a JSON representation of the quota settings as encoded in the corresponding read operation.

5.3.21.3. Getting Bucket Quota

To get a quota, the user must have **`users`** capability set with **`read`** permission.

```
GET /admin/user?quota&uid=<uid>&quota-type=bucket
```

5.3.21.4. Setting Bucket Quota

To set a quota, the user must have **`users`** capability set with **`write`** permission.

```
PUT /admin/user?quota&uid=<uid>&quota-type=bucket
```

The content must include a JSON representation of the quota settings as encoded in the corresponding read operation.

5.3.22. Standard Error Responses

Name	Description	Code
`AccessDenied`	Access denied.	403 Forbidden
`InternalError`	Internal server error.	500 Internal Server Error
`NoSuchUser`	User does not exist.	404 Not Found
`NoSuchBucket`	Bucket does not exist.	404 Not Found
`NoSuchKey`	No such access key.	404 Not Found

CHAPTER 6. OBJECT GATEWAY S3 API

Red Hat Ceph Object Gateway supports a RESTful API that is compatible with the basic data access model of the Amazon S3 API.

6.1. API

6.1.1. Feature Support

The following table describes the support status for current Amazon S3 functional features:

Feature	Status	Remarks
List Buckets	Supported	
Create Bucket	Supported	Different set of canned ACLs
Get Bucket	Supported	
Get Bucket Location	Supported	
Delete Bucket	Supported	
Bucket ACLs (Get, Put)	Supported	Different set of canned ACLs
Bucket Object Versions	Supported	
Get Bucket Info (HEAD)	Supported	
List Bucket Multipart Uploads	Supported	
Bucket Lifecycle	Not Supported	
Policy (Buckets, Objects)	Not Supported	ACLs are supported

Feature	Status	Remarks
Bucket Website	Not Supported	
Bucket Notification	Not Supported	
Bucket Request Payment	Not Supported	
Put Object	Supported	
Delete Object	Supported	
Get Object	Supported	
Object ACLs (Get, Put)	Supported	
Get Object Info (HEAD)	Supported	
Copy Object	Supported	
Initiate Multipart Upload	Supported	
Initiage Multipart Upload Part	Supported	
List Multipart Upload Parts	Supported	
Complete Multipart Upload	Supported	
Abort Multipart Upload	Supported	
Multipart Uploads	Supported	(missing Copy Part)

6.1.2. Unsupported Header Fields

The following common request header fields are not supported:

Name	Type
x-amz-security-token	Request
Server	Response
x-amz-delete-marker	Response
x-amz-id-2	Response
x-amz-request-id	Response
x-amz-version-id	Response

6.2. COMMON

6.2.1. Bucket and Host Name

There are two different modes of accessing the buckets. The first (preferred) method identifies the bucket as the top-level directory in the URI.

```
GET /mybucket HTTP/1.1
Host: cname.domain.com
```

The second method identifies the bucket via a virtual bucket host name. For example:

```
GET / HTTP/1.1
Host: mybucket.cname.domain.com
```

Tip

We prefer the first method, because the second method requires expensive domain certification and DNS wild cards.

6.2.2. Common Request Headers

Request Header	Description
`CONTENT_LENGTH`	Length of the request body.
`DATE`	Request time and date (in UTC).
`HOST`	The name of the host server.
`AUTHORIZATION`	Authorization token.

6.2.3. Common Response Status

HTTP Status	Response Code
`100`	Continue
`200`	Success
`201`	Created
`202`	Accepted
`204`	NoContent
`206`	Partial content
`304`	NotModified
`400`	InvalidArgument
`400`	InvalidDigest

HTTP Status	Response Code
400	BadDigest
400	InvalidBucketName
400	InvalidObjectName
400	UnresolvableGrantByEmailAddress
400	InvalidPart
400	InvalidPartOrder
400	RequestTimeout
400	EntityTooLarge
403	AccessDenied
403	UserSuspended
403	RequestTimeTooSkewed
404	NoSuchKey
404	NoSuchBucket
404	NoSuchUpload
405	MethodNotAllowed

HTTP Status	Response Code
408	RequestTimeout
409	BucketAlreadyExists
409	BucketNotEmpty
411	MissingContentLength
412	PreconditionFailed
416	InvalidRange
422	UnprocessableEntity
500	InternalError

6.3. AUTHENTICATION AND ACLS

Requests to the Ceph Object Gateway can be either authenticated or unauthenticated. Ceph Object Gateway assumes unauthenticated requests are sent by an anonymous user. Ceph Object Gateway supports canned ACLs.

6.3.1. Authentication

For most use cases, clients use existing open source libraries like the Amazon SDK's **AmazonS3Client** for Java, Python Boto, etc. where you simply pass in the access key and secret key, and the library builds the request header and authentication signature for you. However, you can create your own requests and sign them too.

Authenticating a request requires including an access key and a base 64-encoded Hash-based Message Authentication Code (HMAC) in the request before it is sent to the Ceph Object Gateway server. Ceph Object Gateway uses an S3-compatible authentication approach.

```
HTTP/1.1
PUT /buckets/bucket/object.mpeg
Host: cname.domain.com
Date: Mon, 2 Jan 2012 00:01:01 +0000
```

```
Content-Encoding: mpeg
Content-Length: 9999999

Authorization: AWS {access-key}:{hash-of-header-and-secret}
```

In the foregoing example, replace **`{access-key}`** with the value for your access key ID followed by a colon (**`:`**). Replace **`{hash-of-header-and-secret}`** with a hash of a canonicalized header string and the secret corresponding to the access key ID.

To generate the hash of the header string and secret, you must:

1. Get the value of the header string.
2. Normalize the request header string into canonical form.
3. Generate an HMAC using a SHA-1 hashing algorithm.
4. Encode the **`hmac`** result as base-64.

To normalize the header into canonical form:

1. Get all **`content-`** headers.
2. Remove all **`content-`** headers except for **`content-type`** and **`content-md5`**.
3. Ensure the **`content-`** header names are lowercase.
4. Sort the **`content-`** headers lexicographically.
5. Ensure you have a **`Date`** header AND ensure the specified date uses GMT and not an offset.
6. Get all headers beginning with **`x-amz-`**.
7. Ensure that the **`x-amz-`** headers are all lowercase.
8. Sort the **`x-amz-`** headers lexicographically.
9. Combine multiple instances of the same field name into a single field and separate the field values with a comma.
10. Replace white space and line breaks in header values with a single space.
11. Remove white space before and after colons.
12. Append a new line after each header.
13. Merge the headers back into the request header.

Replace the **`{hash-of-header-and-secret}`** with the base-64 encoded HMAC string.

For additional details, consult the Signing and Authenticating REST Requests section of Amazon Simple Storage Service documentation.

6.3.2. Access Control Lists (ACLs)

Ceph Object Gateway supports S3-compatible ACL functionality. An ACL is a list of access grants that specify which operations a user can perform on a bucket or on an object. Each grant has a different meaning when applied to a bucket versus applied to an object:

Permission	Bucket	Object
`READ`	Grantee can list the objects in the bucket.	Grantee can read the object.
`WRITE`	Grantee can write or delete objects in the bucket.	N/A
`READ_ACP`	Grantee can read bucket ACL.	Grantee can read the object ACL.
`WRITE_ACP`	Grantee can write bucket ACL.	Grantee can write to the object ACL.
`FULL_CONTROL`	Grantee has full permissions for object in the bucket.	Grantee can read or write to the object ACL.

6.4. SERVICE OPERATIONS

6.4.1. List Buckets

`GET /` returns a list of buckets created by the user making the request. **`GET /`** only returns buckets created by an authenticated user. You cannot make an anonymous request.

6.4.1.1. Syntax

```
GET / HTTP/1.1
Host: cname.domain.com

Authorization: AWS {access-key}:{hash-of-header-and-secret}
```

6.4.1.2. Response Entities

Name	Type	Description
`Buckets`	Container	Container for list of buckets.
`Bucket`	Container	Container for bucket information.

Name	Type	Description
`Name`	String	Bucket name.
`CreationDate`	Date	UTC time when the bucket was created.
`ListAllMyBucketsResult`	Container	A container for the result.
`Owner`	Container	A container for the bucket owner's **`ID`** and **`DisplayName`**.
`ID`	String	The bucket owner's ID.
`DisplayName`	String	The bucket owner's display name.

6.5. BUCKET OPERATIONS

6.5.1. PUT Bucket

Creates a new bucket. To create a bucket, you must have a user ID and a valid AWS Access Key ID to authenticate requests. You may not create buckets as an anonymous user.

6.5.1.1. Constraints

In general, bucket names should follow domain name constraints.

- Bucket names must be unique.
- Bucket names must begin and end with a lowercase letter.
- Bucket names may contain a dash (-).

6.5.1.2. Syntax

```
PUT /{bucket} HTTP/1.1
Host: cname.domain.com
x-amz-acl: public-read-write

Authorization: AWS {access-key}:{hash-of-header-and-secret}
```

6.5.1.3. Parameters

Name	Description	Valid Values	Required
`x-amz-acl`	Canned ACLs.	**`private`**, **`public-read`**, **`public-read-write`**, **`authenticated-read`**	No

6.5.1.4. HTTP Response

If the bucket name is unique, within constraints and unused, the operation will succeed. If a bucket with the same name already exists and the user is the bucket owner, the operation will succeed. If the bucket name is already in use, the operation will fail.

HTTP Status	Status Code	Description
`409`	BucketAlreadyExists	Bucket already exists under different user’s ownership.

6.5.2. DELETE Bucket

Deletes a bucket. You can reuse bucket names following a successful bucket removal.

6.5.2.1. Syntax

```
DELETE /{bucket} HTTP/1.1
Host: cname.domain.com

Authorization: AWS {access-key}:{hash-of-header-and-secret}
```

6.5.2.2. HTTP Response

HTTP Status	Status Code	Description
`204`	No Content	Bucket removed.

6.5.3. GET Bucket

Returns a list of bucket objects.

6.5.3.1. Syntax

```
GET /{bucket}?max-keys=25 HTTP/1.1
Host: cname.domain.com
```

6.5.3.2. Parameters

Name	Type	Description
`prefix`	String	Only returns objects that contain the specified prefix.
`delimiter`	String	The delimiter between the prefix and the rest of the object name.
`marker`	String	A beginning index for the list of objects returned.
`max-keys`	Integer	The maximum number of keys to return. Default is 1000.

6.5.3.3. HTTP Response

HTTP Status	Status Code	Description
`200`	OK	Buckets retrieved

6.5.3.4. Bucket Response Entities

`GET /{bucket}` returns a container for buckets with the following fields.

Name	Type	Description
`ListBucketResult`	Entity	The container for the list of objects.
`Name`	String	The name of the bucket whose contents will be returned.
`Prefix`	String	A prefix for the object keys.
`Marker`	String	A beginning index for the list of objects returned.
`MaxKeys`	Integer	The maximum number of keys returned.

Name	Type	Description
`Delimiter`	String	If set, objects with the same prefix will appear in the **`CommonPrefixes`** list.
`IsTruncated`	Boolean	If **`true`**, only a subset of the bucket's contents were returned.
`CommonPrefixes`	Container	If multiple objects contain the same prefix, they will appear in this list.

6.5.3.5. Object Response Entities

The **`ListBucketResult`** contains objects, where each object is within a **`Contents`** container.

Name	Type	Description
`Contents`	Object	A container for the object.
`Key`	String	The object's key.
`LastModified`	Date	The object's last-modified date/time.
`ETag`	String	An MD-5 hash of the object. (entity tag)
`Size`	Integer	The object's size.
`StorageClass`	String	Should always return **`STANDARD`**.

6.5.4. Get Bucket Location

Retrieves the bucket's region. The user needs to be the bucket owner to call this. A bucket can be constrained to a region by providing **`LocationConstraint`** during a PUT request.

6.5.4.1. Syntax

Add the **`location`** subresource to bucket resource as shown below.

```
GET /{bucket}?location HTTP/1.1
Host: cname.domain.com

Authorization: AWS {access-key}:{hash-of-header-and-secret}
```

6.5.4.2. Response Entities

Name	Type	Description
`LocationConstraint`	String	The region where bucket resides, empty string for defult region

6.5.5. Get Bucket ACLs

Retrieves the bucket access control list. The user needs to be the bucket owner or to have been granted **`READ_ACP`** permission on the bucket.

6.5.5.1. Syntax

Add the **`acl`** subresource to the bucket request as shown below.

```
GET /{bucket}?acl HTTP/1.1
Host: cname.domain.com

Authorization: AWS {access-key}:{hash-of-header-and-secret}
```

6.5.5.2. Response Entities

Name	Type	Description
`AccessControlPolicy`	Container	A container for the response.
`AccessControlList`	Container	A container for the ACL information.
`Owner`	Container	A container for the bucket owner’s **`ID`** and **`DisplayName`**.
`ID`	String	The bucket owner’s ID.

Name	Type	Description
`DisplayName`	String	The bucket owner’s display name.
`Grant`	Container	A container for **`Grantee`** and **`Permission`**.
`Grantee`	Container	A container for the **`DisplayName`** and **`ID`** of the user receiving a grant of permission.
`Permission`	String	The permission given to the **`Grantee`** bucket.

6.5.6. PUT Bucket ACLs

Sets an access control to an existing bucket. The user needs to be the bucket owner or to have been granted **`WRITE_ACP`** permission on the bucket.

6.5.6.1. Syntax

Add the **`acl`** subresource to the bucket request as shown below.

```
PUT /{bucket}?acl HTTP/1.1
```

6.5.6.2. Request Entities

Name	Type	Description
`AccessControlPolicy`	Container	A container for the request.
`AccessControlList`	Container	A container for the ACL information.
`Owner`	Container	A container for the bucket owner’s **`ID`** and **`DisplayName`**.
`ID`	String	The bucket owner’s ID.

Name	Type	Description
`DisplayName`	String	The bucket owner’s display name.
`Grant`	Container	A container for **`Grantee`** and **`Permission`**.
`Grantee`	Container	A container for the **`DisplayName`** and **`ID`** of the user receiving a grant of permission.
`Permission`	String	The permission given to the **`Grantee`** bucket.

6.5.7. List Bucket Object Versions

Returns a list of metadata about all the version of objects within a bucket. Requires READ access to the bucket.

6.5.7.1. Syntax

Add the **`versions`** subresource to the bucket request as shown below.

```
GET /{bucket}?versions HTTP/1.1
Host: cname.domain.com

Authorization: AWS {access-key}:{hash-of-header-and-secret}
```

6.5.7.2. Parameters

You may specify parameters for **`GET /{bucket}?versions`**, but none of them are required.

Name	Type	Description
`prefix`	String	Returns in-progress uploads whose keys contains the specified prefix.
`delimiter`	String	The delimiter between the prefix and the rest of the object name.
`key-marker`	String	The beginning marker for the list of uploads.

Name	Type	Description
`max-keys`	Integer	The maximum number of in-progress uploads. The default is 1000.
`version-id-marker`	String	Specifies the object version to begin the list.

6.5.7.3. Response Entities

Name	Type	Description
`KeyMarker`	String	The key marker specified by the **`key-marker`** request parameter (if any).
`NextKeyMarker`	String	The key marker to use in a subsequent request if **`IsTruncated`** is **`true`**.
`NextUploadIdMarker`	String	The upload ID marker to use in a subsequent request if **`IsTruncated`** is **`true`**.
`IsTruncated`	Boolean	If **`true`**, only a subset of the bucket's upload contents were returned.
`Size`	Integer	The size of the uploaded part.
`DisplayName`	String	The owners's display name.
`ID`	String	The owners's ID.
`Owner`	Container	A container for the **`ID`** and **`DisplayName`** of the user who owns the object.

Name	Type	Description
`StorageClass`	String	The method used to store the resulting object. **`STANDARD`** or **`REDUCED_REDUNDANCY`**
`Version`	Container	Container for the version information.
`versionId`	String	Version ID of an object.
`versionIdMarker`	String	The last version of the key in a truncated response.

6.5.8. List Bucket Multipart Uploads

`GET /?uploads` returns a list of the current in-progress multipart uploads—i.e., the application initiates a multipart upload, but the service hasn't completed all the uploads yet.

6.5.8.1. Syntax

```
GET /{bucket}?uploads HTTP/1.1
```

6.5.8.2. Parameters

You may specify parameters for **`GET /{bucket}?uploads`**, but none of them are required.

Name	Type	Description
`prefix`	String	Returns in-progress uploads whose keys contains the specified prefix.
`delimiter`	String	The delimiter between the prefix and the rest of the object name.
`key-marker`	String	The beginning marker for the list of uploads.
`max-keys`	Integer	The maximum number of in-progress uploads. The default is 1000.

Name	Type	Description
`max-uploads`	Integer	The maximum number of multipart uploads. The range from 1-1000. The default is 1000.
`version-id-marker`	String	Ignored if **`key-marker`** isn't specified. Specifies the **`ID`** of first upload to list in lexicographical order at or following the **`ID`**.

6.5.8.3. Response Entities

Name	Type	Description
`ListMultipartUploadsResult`	Container	A container for the results.
`ListMultipartUploadsResult.Prefix`	String	The prefix specified by the **`prefix`** request parameter (if any).
`Bucket`	String	The bucket that will receive the bucket contents.
`KeyMarker`	String	The key marker specified by the **`key-marker`** request parameter (if any).
`UploadIdMarker`	String	The marker specified by the **`upload-id-marker`** request parameter (if any).
`NextKeyMarker`	String	The key marker to use in a subsequent request if **`IsTruncated`** is **`true`**.
`NextUploadIdMarker`	String	The upload ID marker to use in a subsequent request if **`IsTruncated`** is **`true`**.
`MaxUploads`	Integer	The max uploads specified by the **`max-uploads`** request parameter.

Name	Type	Description
`Delimiter`	String	If set, objects with the same prefix will appear in the **`CommonPrefixes`** list.
`IsTruncated`	Boolean	If **`true`**, only a subset of the bucket's upload contents were returned.
`Upload`	Container	A container for **`Key`**, **`UploadId`**, **`InitiatorOwner`**, **`StorageClass`**, and **`Initiated`** elements.
`Key`	String	The key of the object once the multipart upload is complete.
`UploadId`	String	The **`ID`** that identifies the multipart upload.
`Initiator`	Container	Contains the **`ID`** and **`DisplayName`** of the user who initiated the upload.
`DisplayName`	String	The initiator's display name.
`ID`	String	The initiator's ID.
`Owner`	Container	A container for the **`ID`** and **`DisplayName`** of the user who owns the uploaded object.
`StorageClass`	String	The method used to store the resulting object. **`STANDARD`** or **`REDUCED_REDUNDANCY`**
`Initiated`	Date	The date and time the user initiated the upload.
`CommonPrefixes`	Container	If multiple objects contain the same prefix, they will appear in this list.

Name	Type	Description
`CommonPrefixes.Prefix`	String	The substring of the key after the prefix as defined by the **`prefix`** request parameter.

6.6. OBJECT OPERATIONS

6.6.1. PUT Object

Adds an object to a bucket. You must have write permissions on the bucket to perform this operation.

6.6.1.1. Syntax

```
PUT /{bucket}/{object} HTTP/1.1
```

6.6.1.2. Request Headers

Name	Description	Valid Values	Required
content-md5	A base64 encoded MD-5 hash of the message.	A string. No defaults or constraints.	No
content-type	A standard MIME type.	Any MIME type. Default: **`binary/octet-stream`**	No
x-amz-meta-<...>	User metadata. Stored with the object.	A string up to 8kb. No defaults.	No
x-amz-acl	A canned ACL.	**`private`**, **`public-read`**, **`public-read-write`**, **`authenticated-read`**	No

6.6.1.3. Response Headers

Name	Description
x-amz-version-id	Returns the version ID or null.

6.6.2. Copy Object

To copy an object, use **`PUT`** and specify a destination bucket and the object name.

6.6.2.1. Syntax

```
PUT /{dest-bucket}/{dest-object} HTTP/1.1
x-amz-copy-source: {source-bucket}/{source-object}
```

6.6.2.2. Request Headers

Name	Description	Valid Values	Required
x-amz-copy-source	The source bucket name + object name.	{bucket}/{obj}	Yes
x-amz-acl	A canned ACL.	**`private`**, **`public-read`**, **`public-read-write`**, **`authenticated-read`**	No
x-amz-copy-if-modified-since	Copies only if modified since the timestamp.	Timestamp	No
x-amz-copy-if-unmodified-since	Copies only if unmodified since the timestamp.	Timestamp	No
x-amz-copy-if-match	Copies only if object ETag matches ETag.	Entity Tag	No
x-amz-copy-if-none-match	Copies only if object ETag doesn’t match.	Entity Tag	No

6.6.2.3. Response Entities

Name	Type	Description
CopyObjectResult	Container	A container for the response elements.
LastModified	Date	The last modified date of the source object.
Etag	String	The ETag of the new object.

6.6.3. Remove Object

Removes an object. Requires WRITE permission set on the containing bucket.

6.6.3.1. Syntax

Deletes an object. If object versioning is on, it creates a marker.

```
DELETE /{bucket}/{object} HTTP/1.1
```

To delete an object when versioning is on, you must specify the **`versionId`** subresource and the version of the object to delete.

```
DELETE /{bucket}/{object}?versionId={versionID} HTTP/1.1
```

6.6.4. Get Object

Retrieves an object from a bucket.

6.6.4.1. Syntax

```
GET /{bucket}/{object} HTTP/1.1
```

Add the **`versionId`** subresource to retrieve a particular version of the object.

```
GET /{bucket}/{object}?versionId={versionID} HTTP/1.1
```

6.6.4.2. Request Headers

Name	Description	Valid Values	Required
range	The range of the object to retrieve.	Range: bytes=beginbyte-endbyte	No
if-modified-since	Gets only if modified since the timestamp.	Timestamp	No
if-unmodified-since	Gets only if not modified since the timestamp.	Timestamp	No
if-match	Gets only if object ETag matches ETag.	Entity Tag	No
if-none-match	Gets only if object ETag matches ETag.	Entity Tag	No

6.6.4.3. Response Headers

Name	Description
Content-Range	Data range, will only be returned if the range header field was specified in the request
x-amz-version-id	Returns the version ID or null.

6.6.5. Get Object Information

Returns information about an object. This request will return the same header information as with the Get Object request, but will include the metadata only, not the object data payload.

6.6.5.1. Syntax

Retrieves the current version of the object.

```
HEAD /{bucket}/{object} HTTP/1.1
```

Add the **`versionId`** subresource to retrieve info for a particular version.

```
HEAD /{bucket}/{object}?versionId={versionID} HTTP/1.1
```

6.6.5.2. Request Headers

Name	Description	Valid Values	Required
range	The range of the object to retrieve.	Range: bytes=beginbyte-endbyte	No
if-modified-since	Gets only if modified since the timestamp.	Timestamp	No
if-unmodified-since	Gets only if not modified since the timestamp.	Timestamp	No
if-match	Gets only if object ETag matches ETag.	Entity Tag	No
if-none-match	Gets only if object ETag matches ETag.	Entity Tag	No

6.6.5.3. Response Headers

Name	Description
x-amz-version-id	Returns the version ID or null.

6.6.6. Get Object ACL

6.6.6.1. Syntax

Returns the ACL for the current version of the object.

```
GET /{bucket}/{object}?acl HTTP/1.1
```

Add the **`versionId`** subresource to retrieve the ACL for a particular version.

```
GET /{bucket}/{object}versionId={versionID}&acl HTTP/1.1
```

6.6.6.2. Response Headers

Name	Description
x-amz-version-id	Returns the version ID or null.

6.6.6.3. Response Entities

Name	Type	Description
`AccessControlPolicy`	Container	A container for the response.
`AccessControlList`	Container	A container for the ACL information.
`Owner`	Container	A container for the object owner’s **`ID`** and **`DisplayName`**.
`ID`	String	The object owner’s ID.
`DisplayName`	String	The object owner’s display name.
`Grant`	Container	A container for **`Grantee`** and **`Permission`**.
`Grantee`	Container	A container for the **`DisplayName`** and **`ID`** of the user receiving a grant of permission.
`Permission`	String	The permission given to the **`Grantee`** object.

6.6.7. Set Object ACL

Sets an object ACL for the current version of the object.

6.6.7.1. Syntax

```
PUT /{bucket}/{object}?acl
```

6.6.7.2. Request Entities

Name	Type	Description
`AccessControlPolicy`	Container	A container for the response.
`AccessControlList`	Container	A container for the ACL information.
`Owner`	Container	A container for the object owner's **`ID`** and **`DisplayName`**.
`ID`	String	The object owner's ID.
`DisplayName`	String	The object owner's display name.
`Grant`	Container	A container for **`Grantee`** and **`Permission`**.
`Grantee`	Container	A container for the **`DisplayName`** and **`ID`** of the user receiving a grant of permission.
`Permission`	String	The permission given to the **`Grantee`** object.

6.6.8. Initiate Multipart Upload

Initiates a multi-part upload process. Returns a **`UploadId`**, which you may specify when adding additional parts, listing parts, and completing or abandoning a multi-part upload.

6.6.8.1. Syntax

```
POST /{bucket}/{object}?uploads
```

6.6.8.2. Request Headers

Name	Description	Valid Values	Required
`content-md5`	A base64 encoded MD-5 hash of the message.	A string. No defaults or constraints.	No
`content-type`	A standard MIME type.	Any MIME type. Default: **`binary/octet-stream`**	No
`x-amz-meta-<… >`	User metadata. Stored with the object.	A string up to 8kb. No defaults.	No
`x-amz-acl`	A canned ACL.	**`private`**, **`public-read`**, **`public-read-write`**, **`authenticated-read`**	No

6.6.8.3. Response Entities

Name	Type	Description
`InitiatedMultipartUploadsResult`	Container	A container for the results.
`Bucket`	String	The bucket that will receive the object contents.
`Key`	String	The key specified by the **`key`** request parameter (if any).
`UploadId`	String	The ID specified by the **`upload-id`** request parameter identifying the multipart upload (if any).

6.6.9. Multipart Upload Part

Adds a part to a multi-part upload.

6.6.9.1. Syntax

Specify the **`uploadId`** subresource and the upload ID to add a part to a multi-part upload.

```
PUT /{bucket}/{object}?partNumber=&uploadId={upload-id} HTTP/1.1
```

6.6.9.2. HTTP Response

The following HTTP response may be returned:

HTTP Status	Status Code	Description
`404`	NoSuchUpload	Specified upload-id does not match any initiated upload on this object

6.6.10. List Multipart Upload Parts

6.6.10.1. Syntax

Specify the **`uploadId`** subresource and the upload ID to list the parts of a multi-part upload.

```
GET /{bucket}/{object}?uploadId={upload-id} HTTP/1.1
```

6.6.10.2. Response Entities

Name	Type	Description
`InitiatedMultipartUploadsResult`	Container	A container for the results.
`Bucket`	String	The bucket that will receive the object contents.
`Key`	String	The key specified by the **`key`** request parameter (if any).
`UploadId`	String	The ID specified by the **`upload-id`** request parameter identifying the multipart upload (if any).
`Initiator`	Container	Contains the **`ID`** and **`DisplayName`** of the user who initiated the upload.

Name	Type	Description
`ID`	String	The initiator's ID.
`DisplayName`	String	The initiator's display name.
`Owner`	Container	A container for the **`ID`** and **`DisplayName`** of the user who owns the uploaded object.
`StorageClass`	String	The method used to store the resulting object. **`STANDARD`** or **`REDUCED_REDUNDANCY`**
`PartNumberMarker`	String	The part marker to use in a subsequent request if **`IsTruncated`** is **`true`**. Precedes the list.
`NextPartNumberMarker`	String	The next part marker to use in a subsequent request if **`IsTruncated`** is **`true`**. The end of the list.
`MaxParts`	Integer	The max parts allowed in the response as specified by the **`max-parts`** request parameter.
`IsTruncated`	Boolean	If **`true`**, only a subset of the object's upload contents were returned.
`Part`	Container	A container for **`Key`**, **`Part`**, **`InitiatorOwner`**, **`StorageClass`**, and **`Initiated`** elements.
`PartNumber`	Integer	The identification number of the part.
`ETag`	String	The part's entity tag.
`Size`	Integer	The size of the uploaded part.

6.6.11. Complete Multipart Upload

Assembles uploaded parts and creates a new object, thereby completing a multipart upload.

6.6.11.1. Syntax

Specify the **`uploadId`** subresource and the upload ID to complete a multi-part upload.

```
POST /{bucket}/{object}?uploadId= HTTP/1.1
```

6.6.11.2. Request Entities

Name	Type	Description	Required
`CompleteMultipartUpload`	Container	A container consisting of one or more parts.	Yes
`Part`	Container	A container for the **`PartNumber`** and **`ETag`**.	Yes
`PartNumber`	Integer	The identifier of the part.	Yes
`ETag`	String	The part’s entity tag.	Yes

6.6.11.3. Response Entities

Name	Type	Description
`CompleteMultipartUploadResult`	Container	A container for the response.
`Location`	URI	The resource identifier (path) of the new object.
`Bucket`	String	The name of the bucket that contains the new object.
`Key`	String	The object’s key.

Name	Type	Description
`ETag`	String	The entity tag of the new object.

6.6.12. Abort Multipart Upload

Aborts a multipart upload.

6.6.12.1. Syntax

Specify the **`uploadId`** subresource and the upload ID to abort a multi-part upload.

```
DELETE /{bucket}/{object}?uploadId={upload-id} HTTP/1.1
```

6.7. ACCESSING THE GATEWAY

You can use various programming languages to create a connection with the gateway server and do the bucket management tasks. There are different open source libraries available for these programming languages that are used for authentication with the gateway.

The sections mentioned below will describe the procedure for some of the popular programmimg languages.

6.7.1. Prerequisites

You have to follow some pre-requisites on your **`gateway node`** before attempting to access the gateway server. The pre-requisites are as follows:

1. Set up your gateway server properly by following the instructions mentioned in Install RHCS v1.3 Ceph Object Gateway for RHEL.

2. **DO NOT** modify the Ceph configuration file to use port **`80`** and let **`Civetweb`** use the default port **`7480`**. Port **`80`** is required by **`Apache`** and it needs to be running and enabled.

3. Install and start **`Apache`**.

   ```
   sudo yum install httpd -y
   sudo systemctl start httpd
   sudo systemctl enable httpd.service
   ```

4. Open port **`80`** on firewall.

   ```
   sudo firewall-cmd --zone=public --add-port=80/tcp --permanent
   sudo firewall-cmd --reload
   ```

5. Disable **`ssl`**.

   If you have **`ssl`** enabled in your **`gateway node`**, please follow the steps mentioned in Disable SSL to disable it.

6. Add a wildcard to your DNS server that you are using for your gateway as mentioned in Add wildcard to DNS.

   You can also set up your gateway node for local DNS caching. To do so, execute the following steps:

   - Install and setup **dnsmasq**.

     ```
     sudo yum install dnsmasq -y
     echo "address=/.<FQDN_of_gateway_node>/<IP_of_gateway_node>" |
     sudo tee --append /etc/dnsmasq.conf
     sudo systemctl start dnsmasq
     sudo systemctl enable dnsmasq
     ```

     Replace **<IP_of_gateway_node>** and **<FQDN_of_gateway_node>** with the IP address and FQDN of your gateway node.

   - Stop NetworkManager.

     ```
     sudo systemctl stop NetworkManager
     sudo systemctl disable NetworkManager
     ```

   - Set your gateway server's IP as the nameserver.

     ```
     echo "DNS1=<IP_of_gateway_node>" | sudo tee --append
     /etc/sysconfig/network-scripts/ifcfg-eth0
     echo "<IP_of_gateway_node> <FQDN_of_gateway_node>" | sudo tee -
     -append /etc/hosts
     sudo systemctl restart network
     sudo systemctl enable network
     sudo systemctl restart dnsmasq
     ```

     Replace **<IP_of_gateway_node>** and **<FQDN_of_gateway_node>** with the IP address and FQDN of your gateway node.

   - Verify subdomain requests:

     ```
     ping mybucket.<FQDN_of_gateway_node>
     ```

     Replace **<FQDN_of_gateway_node>** with the FQDN of your gateway node.

     Warning

     Setting up your gateway server for local DNS caching is for testing purposes only. You won't be able to access outside network after doing this. **It is strongly recommended to use a proper DNS server for your Ceph cluster and gateway node.**

7. Create the **radosgw** user for **S3** access carefully as mentioned in Create Radosgw user for S3 access and copy the generated **access_key** and **secret_key**. You will need these keys for **S3** access and subsequent bucket management tasks.

6.7.2. Ruby AWS::S3 Examples (aws-s3 gem)

You can use **`Ruby`** programming language alongwith **`aws-s3`** gem for **`S3`** access. Execute the steps mentioned below in your **`gateway node`** for accessing the gateway with **`Ruby AWS::S3`**.

6.7.2.1. Setup Ruby

Execute the following steps to setup **`Ruby`**:

1. Install **`Ruby`**:

```
sudo yum install ruby -y
```

Note

The above command will install **`ruby`** and it's essential dependencies like **`rubygems`** and **`ruby-libs`** too. If somehow the command doesn't install all the dependencies, install them separately.

2. Install **`aws-s3`**:

```
sudo gem install aws-s3
```

6.7.2.2. Creating a connection

1. Create a project directory:

```
mkdir ruby_aws_s3
cd ruby_aws_s3
```

2. Create the connection file:

```
vim conn.rb
```

3. Paste the following contents in the **`conn.rb`** file:

```
#!/usr/bin/env ruby

require 'aws/s3'
require 'resolv-replace'

AWS::S3::Base.establish_connection!(
        :server            => '<FQDN_of_gateway_node>',
        :port           => '7480',
        :access_key_id     => 'my-access-key',
        :secret_access_key => 'my-secret-key'
)
```

Replace **`<FQDN_of_gateway_node>`** with the FQDN of you gateway node. Replace **`my-access-key`** and **`my-secret-key`** with the **`access_key`** and **`secret_key`** that was generated when you created the **`radosgw`** user for **`S3`** access as mentioned in Create Radosgw user for S3 access.

An example connection file will look like the following:

```
#!/usr/bin/env ruby

require 'aws/s3'

require 'resolv-replace'

AWS::S3::Base.establish_connection!(
        :server            =>
'testclient.englab.pnq.redhat.com',
        :port           => '7480',
        :access_key_id     => '98J4R9P22P5CDL65HKP8',
        :secret_access_key =>
'6C+jcaP0dp0+FZfrRNgyGA9EzRy25pURldwje049'
)
```

Save the file and exit the editor.

4. Make the file executable:

```
chmod +x conn.rb
```

5. Run the file:

```
./conn.rb | echo $?
```

If you have provided the values correctly in the file, the output of the command will be **`0`**.

6.7.2.3. Creating a bucket

1. Create a new file:

```
vim create_bucket.rb
```

Paste the following contents into the file:

```
#!/usr/bin/env ruby

load 'conn.rb'

AWS::S3::Bucket.create('my-new-bucket1')
```

Save the file and exit the editor.

2. Make the file executable:

```
chmod +x create_bucket.rb
```

3. Run the file:

```
./create_bucket.rb
```

If the output of the command is **`true`** it would mean that bucket **`my-new-bucket1`** was created successfully.

6.7.2.4. Listing owned buckets

1. Create a new file:

```
vim list_owned_buckets.rb
```

Paste the following content into the file:

```
#!/usr/bin/env ruby

load 'conn.rb'

AWS::S3::Service.buckets.each do |bucket|
        puts "#{bucket.name}\t#{bucket.creation_date}"
end
```

Save the file and exit the editor.

2. Make the file executable:

```
chmod +x list_owned_buckets.rb
```

3. Run the file:

```
./list_owned_buckets.rb
```

The output should look something like this:

```
my-new-bucket1 2016-01-21 10:33:19 UTC
```

6.7.2.5. Creating an object

1. Create a new file:

```
vim create_object.rb
```

Paste the following contents into the file:

```
#!/usr/bin/env ruby

load 'conn.rb'

AWS::S3::S3Object.store(
        'hello.txt',
        'Hello World!',
        'my-new-bucket1',
        :content_type => 'text/plain'
)
```

Save the file and exit the editor.

2. Make the file executable:

```
chmod +x create_object.rb
```

3. Run the file:

```
./create_object.rb
```

This will create a file **`hello.txt`** with the string **`Hello World!`**.

6.7.2.6. Listing a Bucket's Content

1. Create a new file:

```
vim list_bucket_content.rb
```

Paste the following content into the file:

```
#!/usr/bin/env ruby

load 'conn.rb'

new_bucket = AWS::S3::Bucket.find('my-new-bucket1')
new_bucket.each do |object|
        puts "#{object.key}\t#{object.about['content-
length']}\t#{object.about['last-modified']}"
end
```

Save the file and exit the editor.

2. Make the file executable.

```
chmod +x list_bucket_content.rb
```

3. Run the file:

```
./list_bucket_content.rb
```

The output will look something like this:

```
hello.txt    12    Fri, 22 Jan 2016 15:54:52 GMT
```

6.7.2.7. Deleting a empty bucket

1. Create a new file:

```
vim del_empty_bucket.rb
```

Paste the following contents into the file:

```
#!/usr/bin/env ruby

load 'conn.rb'

AWS::S3::Bucket.delete('my-new-bucket1')
```

Save the file and exit the editor.

2. Make the file executable:

```
chmod +x del_empty_bucket.rb
```

3. Run the file:

```
./del_empty_bucket.rb | echo $?
```

If the bucket is successfully deleted, the command will return **0** as output.

Note

Please edit the **create_bucket.rb** file to create empty buckets like **my-new-bucket9**, **my-new-bucket10** etc and edit the above mentioned **del_empty_bucket.rb** file accordingly before trying to delete empty buckets.

6.7.2.8. Deleting a non-empty bucket (forcefully)

1. Create a new file:

```
vim del_non_empty_bucket.rb
```

Paste the following contents into the file:

```
#!/usr/bin/env ruby

load 'conn.rb'

AWS::S3::Bucket.delete('my-new-bucket1', :force => true)
```

Save the file and exit the editor.

2. Make the file executable:

```
chmod +x del_non_empty_bucket.rb
```

3. Run the file:

```
./del_non_empty_bucket.rb | echo $?
```

If the bucket is successfully deleted, the command will return **`0`** as output.

6.7.2.9. Deleting an object

1. Create a new file:

   ```
   vim delete_object.rb
   ```

   Paste the following contents into the file:

   ```
   #!/usr/bin/env ruby

   load 'conn.rb'

   AWS::S3::S3Object.delete('hello.txt', 'my-new-bucket1')
   ```

   Save the file and exit the editor.

2. Make the file executable:

   ```
   chmod +x delete_object.rb
   ```

3. Run the file:

   ```
   ./delete_object.rb
   ```

   This will delete the object **`hello.txt`**.

CHAPTER 7. OBJECT GATEWAY SWIFT API

Ceph supports a RESTful API that is compatible with the basic data access model of the Swift API.

7.1. FEATURES SUPPORT

The following table describes the support status for current Swift functional features:

Feature	Status	Remarks
Authentication	Supported	
Get Account Metadata	Supported	No custom metadata
Swift ACLs	Supported	Supports a subset of Swift ACLs
List Containers	Supported	
Delete Container	Supported	
Create Container	Supported	
Get Container Metadata	Supported	
Update Container Metadata	Supported	
Delete Container Metadata	Supported	
List Objects	Supported	
Static Website	Not Supported	
Create/Update an Object	Supported	

Feature	Status	Remarks
Create Large Object	Supported	
Delete Object	Supported	
Get Object	Supported	
Copy Object	Supported	
Get Object Metadata	Supported	
Add/Update Object Metadata	Supported	
Temp URL Operations	Supported	
Expiring Objects	Not Supported	
Object Versioning	Not Supported	
CORS	Not Supported	

7.2. AUTHENTICATION

Swift API requests that require authentication must contain an **`X-Storage-Token`** authentication token in the request header. The token may be retrieved from Ceph Object Gateway, or from another authenticator. To obtain a token from Ceph Object Gateway, you must create a user. For example:

```
sudo radosgw-admin user create --uid="{username}" --display-name="
{Display Name}"
```

7.2.1. Authentication GET

To authenticate a user, make a request containing an **`X-Auth-User`** and a **`X-Auth-Key`** in the header.

7.2.1.1. Syntax

```
GET /auth HTTP/1.1
Host: swift.radosgwhost.com
X-Auth-User: johndoe
X-Auth-Key: R7UUOLFDI2ZI9PRCQ53K
```

7.2.1.2. Request Headers

Name	Description	Type	Required
`X-Auth-User`	The key Ceph Object Gateway username to authenticate.	String	Yes
`X-Auth-Key`	The key associated to a Ceph Object Gateway username.	String	Yes

7.2.1.3. Response Headers

The response from the server should include an **`X-Auth-Token`** value. The response may also contain a **`X-Storage-Url`** that provides the **`{api version}/{account}`** prefix that is specified in other requests throughout the API documentation.

Name	Description	Type
`X-Storage-Token`	The authorization token for the **`X-Auth-User`** specified in the request.	String
`X-Storage-Url`	The URL and **`{api version}/{account}`** path for the user.	String

A typical response looks like this:

```
HTTP/1.1 204 No Content
Date: Mon, 16 Jul 2012 11:05:33 GMT
Server: swift
X-Storage-Url: https://swift.radosgwhost.com/v1/ACCT-12345
X-Auth-Token: UOlCCC8TahFKlWuv9DB09TWHF0nDjpPElha0kAa
Content-Length: 0
Content-Type: text/plain; charset=UTF-8
```

7.3. SERVICE OPERATIONS

To retrieve data about our Swift-compatible service, you may execute **`GET`** requests using the **`X-Storage-Url`** value retrieved during authentication.

7.3.1. List Containers

A **`GET`** request that specifies the API version and the account will return a list of containers for a particular user account. Since the request returns a particular user's containers, the request requires an authentication token. The request cannot be made anonymously.

7.3.1.1. Syntax

```
GET /{api version}/{account} HTTP/1.1
Host: {fqdn}
X-Auth-Token: {auth-token}
```

7.3.1.2. Request Parameters

Name	Description	Type	Required	Valid Values
`limit`	Limits the number of results to the specified value.	Integer	No	N/A
`format`	Defines the format of the result.	String	No	**`json`** or **`xml`**
`marker`	Returns a list of results greater than the marker value.	String	No	N/A

7.3.1.3. Response Entities

The response contains a list of containers, or returns with an HTTP 204 response code

Name	Description	Type
`account`	A list for account information.	Container
`container`	The list of containers.	Container

Name	Description	Type
`name`	The name of a container.	String
`bytes`	The size of the container.	Integer

7.4. CONTAINER OPERATIONS

A container is a mechanism for storing data objects. An account may have many containers, but container names must be unique. This API enables a client to create a container, set access controls and metadata, retrieve a container’s contents, and delete a container. Since this API makes requests related to information in a particular user’s account, all requests in this API must be authenticated unless a container’s access control is deliberately made publicly accessible (i.e., allows anonymous requests).

Note

The Amazon S3 API uses the term 'bucket' to describe a data container. When you hear someone refer to a 'bucket' within the Swift API, the term 'bucket' may be construed as the equivalent of the term 'container.'

One facet of object storage is that it does not support hierarchical paths or directories. Instead, it supports one level consisting of one or more containers, where each container may have objects. The RADOS Gateway’s Swift-compatible API supports the notion of 'pseudo-hierarchical containers,' which is a means of using object naming to emulate a container (or directory) hierarchy without actually implementing one in the storage system. You may name objects with pseudo-hierarchical names (e.g., photos/buildings/empire-state.jpg), but container names cannot contain a forward slash (`/`) character.

7.4.1. Create a Container

To create a new container, make a **`PUT`** request with the API version, account, and the name of the new container. The container name must be unique, must not contain a forward-slash (/) character, and should be less than 256 bytes. You may include access control headers and metadata headers in the request. You may also include a storage policy identifying a key for a set of placement pools (e.g., execute **`radosgw-admin zone get`** to see a list of available keys under **`placement_pools`**). A storage policy enables you to specify a special set of pools for the container (e.g., SSD-based storage). The operation is idempotent; that is, if you make a request to create a container that already exists, it will return with a HTTP 202 return code, but will not create another container.

7.4.1.1. Syntax

```
PUT /{api version}/{account}/{container} HTTP/1.1
Host: {fqdn}
X-Auth-Token: {auth-token}
X-Container-Read: {comma-separated-uids}
```

```
X-Container-Write: {comma-separated-uids}
X-Container-Meta-{key}: {value}
X-Storage-Policy: {placement-pools-key}
```

7.4.1.2. Headers

Name	Description	Type	Required
`X-Container-Read`	The user IDs with read permissions for the container.	Comma-separated string values of user IDs.	No
`X-Container-Write`	The user IDs with write permissions for the container.	Comma-separated string values of user IDs.	No
`X-Container-Meta-{key}`	A user-defined meta data key that takes an arbitrary string value.	String	No
`X-Storage-Policy`	The key that identifies the storage policy under **`placement_pools`** for the Ceph Object Gateway. Execute **`radosgw-admin zone get`** for available keys.	String	No

7.4.1.3. HTTP Response

If a container with the same name already exists, and the user is the container owner then the operation will succeed. Otherwise the operation will fail.

Name	Description	Status Code

Name	Description	Status Code
`409`	The container already exists under a different user's ownership.	**`BucketAlreadyExists`**

7.4.2. List a Container's Objects

To list the objects within a container, make a **`GET`** request with the with the API version, account, and the name of the container. You can specify query parameters to filter the full list, or leave out the parameters to return a list of the first 10,000 object names stored in the container.

7.4.2.1. Syntax

```
GET /{api version}/{container} HTTP/1.1
 Host: {fqdn}
 X-Auth-Token: {auth-token}
```

7.4.2.2. Parameters

Name	Description	Type	Valid Values	Required
`format`	Defines the format of the result.	String	**`json`** or **`xml`**	No
`prefix`	Limits the result set to objects beginning with the specified prefix.	String	N/A	No
`marker`	Returns a list of results greater than the marker value.	String	N/A	No
`limit`	Limits the number of results to the specified value.	Integer	0 - 10,000	No

Name	Description	Type	Valid Values	Required
`delimiter`	The delimiter between the prefix and the rest of the object name.	String	N/A	No
`path`	The pseudo-hierarchical path of the objects.	String	N/A	No

7.4.2.3. Response Entities

Name	Description	Type
`container`	The container.	Container
`object`	An object within the container.	Container
`name`	The name of an object within the container.	String
`hash`	A hash code of the object’s contents.	String
`last_modified`	The last time the object’s contents were modified.	Date
`content_type`	The type of content within the object.	String

7.4.3. Update a Container’s ACLs

When a user creates a container, the user has read and write access to the container by default. To allow other users to read a container’s contents or write to a container, you must specifically enable the user. You may also specify **`*`** in the **`X-Container-Read`** or **`X-Container-Write`** settings, which effectively enables all users to either read from or write to the container. Setting **`*`** makes the container public. That is it enables anonymous users to either read from or write to the container.

7.4.3.1. Syntax

```
POST /{api version}/{account}/{container} HTTP/1.1
Host: {fqdn}
 X-Auth-Token: {auth-token}
 X-Container-Read: *
 X-Container-Write: {uid1}, {uid2}, {uid3}
```

7.4.3.2. Request Headers

Name	Description	Type	Required
`X-Container-Read`	The user IDs with read permissions for the container.	Comma-separated string values of user IDs.	No
`X-Container-Write`	The user IDs with write permissions for the container.	Comma-separated string values of user IDs.	No

7.4.4. Add/Update Container Metadata

To add metadata to a container, make a **`POST`** request with the API version, account, and container name. You must have write permissions on the container to add or update metadata.

7.4.4.1. Syntax

```
POST /{api version}/{account}/{container} HTTP/1.1
Host: {fqdn}
 X-Auth-Token: {auth-token}
 X-Container-Meta-Color: red
 X-Container-Meta-Taste: salty
```

7.4.4.2. Request Headers

Name	Description	Type	Required
`X-Container-Meta-{key}`	A user-defined meta data key that takes an arbitrary string value.	String	No

7.4.5. Delete a Container

To delete a container, make a **`DELETE`** request with the API version, account, and the name of the container. The container must be empty. If you’d like to check if the container is empty, execute a

HEAD request against the container. Once you've successfully removed the container, you'll be able to reuse the container name.

7.4.5.1. Syntax

```
DELETE /{api version}/{account}/{container} HTTP/1.1
Host: {fqdn}
X-Auth-Token: {auth-token}
```

7.4.5.2. HTTP Response

Name	Description	Status Code
204	The container was removed.	**NoContent**

7.5. OBJECT OPERATIONS

An object is a container for storing data and metadata. A container may have many objects, but the object names must be unique. This API enables a client to create an object, set access controls and metadata, retrieve an object's data and metadata, and delete an object. Since this API makes requests related to information in a particular user's account, all requests in this API must be authenticated unless the container or object's access control is deliberately made publicly accessible (i.e., allows anonymous requests).

7.5.1. Create/Update an Object

To create a new object, make a **PUT** request with the API version, account, container name and the name of the new object. You must have write permission on the container to create or update an object. The object name must be unique within the container. The **PUT** request is not idempotent, so if you do not use a unique name, the request will update the object. However, you may use pseudo-hierarchical syntax in your object name to distinguish it from another object of the same name if it is under a different pseudo-hierarchical directory. You may include access control headers and metadata headers in the request.

7.5.1.1. Syntax

```
PUT /{api version}/{account}/{container}/{object} HTTP/1.1
 Host: {fqdn}
 X-Auth-Token: {auth-token}
```

7.5.1.2. Request Headers

Name	Description	Type	Required	Valid Values
`ETag`	An MD5 hash of the object's contents. Recommended.	String	No	N/A
`Content-Type`	The type of content the object contains.	String	No	N/A
`Transfer-Encoding`	Indicates whether the object is part of a larger aggregate object.	String	No	**`chunked`**

7.5.2. Copy an Object

Copying an object allows you to make a server-side copy of an object, so that you don't have to download it and upload it under another container/name. To copy the contents of one object to another object, you may make either a **`PUT`** request or a **`COPY`** request with the API version, account, and the container name. For a **`PUT`** request, use the destination container and object name in the request, and the source container and object in the request header. For a **`Copy`** request, use the source container and object in the request, and the destination container and object in the request header. You must have write permission on the container to copy an object. The destination object name must be unique within the container. The request is not idempotent, so if you do not use a unique name, the request will update the destination object. However, you may use pseudo-hierarchical syntax in your object name to distinguish the destination object from the source object of the same name if it is under a different pseudo-hierarchical directory. You may include access control headers and metadata headers in the request.

7.5.2.1. Syntax

```
PUT /{api version}/{account}/{dest-container}/{dest-object} HTTP/1.1
X-Copy-From: {source-container}/{source-object}
Host: {fqdn}
X-Auth-Token: {auth-token}
```

or alternatively:

```
COPY /{api version}/{account}/{source-container}/{source-object}
HTTP/1.1
Destination: {dest-container}/{dest-object}
```

7.5.2.2. Request Headers

Name	Description	Type	Required
`X-Copy-From`	Used with a **`PUT`** request to define the source container/object path.	String	Yes, if using **`PUT`**
`Destination`	Used with a **`COPY`** request to define the destination container/object path.	String	Yes, if using **`COPY`**
`If-Modified-Since`	Only copies if modified since the date/time of the source object's **`last_modified`** attribute.	Date	No
`If-Unmodified-Since`	Only copies if not modified since the date/time of the source object's **`last_modified`** attribute.	Date	No
`Copy-If-Match`	Copies only if the ETag in the request matches the source object's ETag.	ETag.	No
`Copy-If-None-Match`	Copies only if the ETag in the request does not match the source object's ETag.	ETag.	No

7.5.3. Delete an Object

To delete an object, make a **`DELETE`** request with the API version, account, container and object name. You must have write permissions on the container to delete an object within it. Once you've successfully deleted the object, you'll be able to reuse the object name.

7.5.3.1. Syntax

```
DELETE /{api version}/{account}/{container}/{object} HTTP/1.1
Host: {fqdn}
X-Auth-Token: {auth-token}
```

7.5.4. Get an Object

To retrieve an object, make a **`GET`** request with the API version, account, container and object name. You must have read permissions on the container to retrieve an object within it.

7.5.4.1. Syntax

```
GET /{api version}/{account}/{container}/{object} HTTP/1.1
Host: {fqdn}
X-Auth-Token: {auth-token}
```

7.5.4.2. Request Headers

Name	Description	Type	Required
`range`	To retrieve a subset of an object's contents, you may specify a byte range.	Date	No
`If-Modified-Since`	Only copies if modified since the date/time of the source object's **`last_modified`** attribute.	Date	No
`If-Unmodified-Since`	Only copies if not modified since the date/time of the source object's **`last_modified`** attribute.	Date	No
`Copy-If-Match`	Copies only if the ETag in the request matches the source object's ETag.	ETag.	No
`Copy-If-None-Match`	Copies only if the ETag in the request does not match the source object's ETag.	ETag.	No

7.5.4.3. Response Headers

Name	Description
`Content-Range`	The range of the subset of object contents. Returned only if the range header field was specified in the request.

7.5.5. Get Object Metadata

To retrieve an object's metadata, make a **`HEAD`** request with the API version, account, container and object name. You must have read permissions on the container to retrieve metadata from an object within the container. This request returns the same header information as the request for the object itself, but it does not return the object's data.

7.5.5.1. Syntax

```
HEAD /{api version}/{account}/{container}/{object} HTTP/1.1
Host: {fqdn}
X-Auth-Token: {auth-token}
```

7.5.6. Add/Update Object Metadata

To add metadata to an object, make a **`POST`** request with the API version, account, container and object name. You must have write permissions on the parent container to add or update metadata.

7.5.6.1. Syntax

```
POST /{api version}/{account}/{container}/{object} HTTP/1.1
Host: {fqdn}
X-Auth-Token: {auth-token}
```

7.5.6.2. Request Headers

Name	Description	Type	Requir ed
`X-Object-Meta-{key}`	A user-defined meta data key that takes an arbitrary string value.	String	No

7.6. TEMP URL OPERATIONS

To allow temporary access (for eg for GET requests) to objects without the need to share credentials, temp url functionality is supported by swift endpoint of radosgw. For this functionality, initially the value of X-Account-Meta-Temp-URL-Key and optionally X-Account-Meta-Temp-URL-Key-2 should be set. The Temp URL functionality relies on a HMAC-SHA1 signature against these secret keys.

7.6.1. POST Temp-URL Keys

A **`POST`** request to the swift account with the required Key will set the secret temp url key for the account against which temporary url access can be provided to accounts. Up to two keys are supported, and signatures are checked against both the keys, if present, so that keys can be rotated without invalidating the temporary urls.

7.6.1.1. Syntax

```
POST /{api version}/{account} HTTP/1.1
Host: {fqdn}
X-Auth-Token: {auth-token}
```

7.6.1.2. Request Headers

Name	Description	Type	Required
`X-Account-Meta-Temp-URL-Key`	A user-defined key that takes an arbitrary string value.	String	Yes
`X-Account-Meta-Temp-URL-Key-2`	A user-defined key that takes an arbitrary string value.	String	No

7.6.2. GET Temp-URL Objects

Temporary URL uses a cryptographic HMAC-SHA1 signature, which includes the following elements:

1. The value of the Request method, "GET" for instance
2. The expiry time, in format of seconds since the epoch, ie Unix time
3. The request path starting from "v1" onwards

The above items are normalized with newlines appended between them, and a HMAC is generated using the SHA-1 hashing algorithm against one of the Temp URL Keys posted earlier.

A sample python script to demonstrate the above is given below:

```
import hmac
from hashlib import sha1
from time import time

method = 'GET'
host = 'https://objectstore.example.com'
duration_in_seconds = 300  # Duration for which the url is valid
expires = int(time() + duration_in_seconds)
path = '/v1/your-bucket/your-object'
key = 'secret'
hmac_body = '%s\n%s\n%s' % (method, expires, path)
hmac_body = hmac.new(key, hmac_body, sha1).hexdigest()
sig = hmac.new(key, hmac_body, sha1).hexdigest()
rest_uri = "{host}{path}?temp_url_sig={sig}&temp_url_expires=
{expires}".format(
     host=host, path=path, sig=sig, expires=expires)
print rest_uri

# Example Output
# https://objectstore.example.com/v1/your-bucket/your-object?
temp_url_sig=ff4657876227fc6025f04fcf1e82818266d022c6&temp_url_expires=
1423200992
```