CONFIGURING RADOS GATEWAY

Before you can start RADOS Gateway, you must modify your `ceph.conf` file to include a section for RADOS Gateway You must also create an `rgw.conf` file in the `/etc/apache2/sites-enabled` directory. The `rgw.conf` file configures Apache to interact with FastCGI.

ADD A RADOS GW CONFIGURATION TO `CEPH.CONF`

Add the RADOS Gateway configuration to your `ceph.conf` file. The RADOS Gateway configuration requires you to specify the host name where you installed RADOS Gateway, a keyring (for use with cephx), the socket path and a log file. For example:

```
[client.radosgw.gateway]
        host = {host-name}
        keyring = /etc/ceph/keyring.radosgw.gateway
        rgw socket path = /tmp/radosgw.sock
        log file = /var/log/ceph/radosgw.log
```

Note: `host` must be your machine hostname, not FQDN.

DEPLOY `CEPH.CONF`

If you deploy Ceph with `mkcephfs`, manually redeploy `ceph.conf` to the hosts in your cluster. For example:

```
cd /etc/ceph
ssh {host-name} sudo /etc/ceph/ceph.conf < ceph.conf
```

CREATE DATA DIRECTORY

The `mkcephfs` deployment script may not create the default RGW data directory. Create data directories for each instance of a `radosgw` daemon (if you haven't done so already). The `host` variables in the `ceph.conf` file determine which host runs each instance of a `radosgw` daemon. The typical form specifes the daemon `radosgw`, the cluster name and the daemon ID.

```
sudo mkdir -p /var/lib/ceph/radosgw/{$cluster}-{$id}
```

Using the exemplary `ceph.conf` settings above, you would execute the following:

```
sudo mkdir -p /var/lib/ceph/radosgw/ceph-radosgw.gateway
```

CREATE `RGW.CONF`

Create an `rgw.conf` file on the host where you installed RADOS Gateway under the `/etc/apache2/sites-available` directory.

We recommend deploying FastCGI as an external server, because allowing Apache to manage FastCGI sometimes introduces high latency. To manage FastCGI as an external server, use the `FastCgiExternalServer` directive. See FastCgiExternalServer for details on this directive. See Module mod_fastcgi for general details.

```
FastCgiExternalServer /var/www/s3gw.fcgi -socket /tmp/radosgw.sock
```

Once you have configured FastCGI as an external server, you must create the virtual host configuration within your `rgw.conf` file. See Apache Virtual Host documentation for details on `<VirtualHost>` format and settings. Replace the values in brackets.

```
<VirtualHost *:80>
```

```
        ServerName {fqdn}
        ServerAdmin {email.address}
        DocumentRoot /var/www
</VirtualHost>
```

RADOS Gateway requires a rewrite rule for the Amazon S3-compatible interface. It’s required for passing in the `HTTP_AUTHORIZATION` env for S3, which is filtered out by Apache. The rewrite rule is not necessary for the OpenStack Swift-compatible interface. Turn on the rewrite engine and add the following rewrite rule to your Virtual Host configuration.

```
RewriteEngine On
RewriteRule ^/([a-zA-Z0-9-_.]*)([/]?.*) /s3gw.fcgi?page=$1&params=$2&%{QUERY_STRING} [E=HTTP_
```

Since the `<VirtualHost>` is running `mod_fastcgi.c`, you must include a section in your `<VirtualHost>` configuration for the `mod_fastcgi.c` module.

```
<VirtualHost *:80>
        ...
        <IfModule mod_fastcgi.c>
                <Directory /var/www>
                        Options +ExecCGI
                        AllowOverride All
                        SetHandler fastcgi-script
                        Order allow,deny
                        Allow from all
                        AuthBasicAuthoritative Off
                </Directory>
        </IfModule>
        ...
</VirtualHost>
```

See <IfModule> Directive for additional details.

Finally, you should configure Apache to allow encoded slashes, provide paths for log files and to turn off server signatures.

```
<VirtualHost *:80>
...
        AllowEncodedSlashes On
        ErrorLog /var/log/apache2/error.log
        CustomLog /var/log/apache2/access.log combined
        ServerSignature Off
</VirtualHost>
```

Important: If you are using CentOS or similar, make sure that `FastCgiWrapper` is turned off in `/etc/httpd/conf.d/fastcgi.conf`.

ENABLE THE RADOS GATEWAY CONFIGURATION

Enable the site for `rgw.conf`.

```
sudo a2ensite rgw.conf
```

Disable the default site.

```
sudo a2dissite default
```

ADD A RADOS GW SCRIPT

Add a `s3gw.fcgi` file (use the same name referenced in the first line of `rgw.conf`) to `/var/www`. The contents of the file should include:

```
#!/bin/sh
exec /usr/bin/radosgw -c /etc/ceph/ceph.conf -n client.radosgw.gateway
```

Ensure that you apply execute permissions to `s3gw.fcgi`.

```
sudo chmod +x s3gw.fcgi
```

GENERATE A KEYRING AND KEY FOR RADOS GATEWAY

You must create a keyring for the RADOS Gateway. For example:

```
sudo ceph-authtool --create-keyring /etc/ceph/keyring.radosgw.gateway
sudo chmod +r /etc/ceph/keyring.radosgw.gateway
```

Generate a key so that RADOS Gateway can identify a user name and authenticate the user with the cluster. Then, add capabilities to the key. For example:

```
sudo ceph-authtool /etc/ceph/keyring.radosgw.gateway -n client.radosgw.gateway --gen-key
sudo ceph-authtool -n client.radosgw.gateway --cap osd 'allow rwx' --cap mon 'allow r' /etc/c
```

ADD TO CEPH KEYRING ENTRIES

Once you have created a keyring and key for RADOS GW, add it as an entry in the Ceph keyring. For example:

```
sudo ceph -k /etc/ceph/ceph.keyring auth add client.radosgw.gateway -i /etc/ceph/keyring.rado
```

RESTART SERVICES AND START THE RADOS GATEWAY

To ensure that all components have reloaded their configurations, we recommend restarting your ceph and apaches services. Then, start up the `radosgw` service. For example:

```
sudo service ceph restart
sudo service apache2 restart
sudo /etc/init.d/radosgw start
```

CREATE A RADOS GATEWAY USER

To use the REST interfaces, first create an initial RADOS Gateway user. The RADOS Gateway user is not the same user as the `client.rados.gateway` user, which identifies the RADOS Gateway as a user of the RADOS cluster. The RADOS Gateway user is a user of the RADOS Gateway.

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

For example:

```
radosgw-admin user create --uid=johndoe --display-name="John Doe" --email=john@example.com
```

```
{ "user_id": "johndoe",
  "rados_uid": 0,
  "display_name": "John Doe",
  "email": "john@example.com",
  "suspended": 0,
  "subusers": [],
  "keys": [
    { "user": "johndoe",
      "access_key": "QFAMEDSJP5DEKJO0DDXY",
      "secret_key": "iaSFLDVvDdQt6lkNzHyW4fPLZugBAI1g17LO0+87"}],
  "swift_keys": []}
```

Creating a user also creates an `access_key` and `secret_key` entry for use with any S3 API-compatible client. For details on RADOS Gateway administration, see radosgw-admin.

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

CONFIGURING THE OPERATIONS LOGGING

By default, the RADOS Gateway will log every successful operation in the RADOS backend. This means that every request, whether it is a read request or a write request will generate a RADOS operation that writes data. This does not come without cost, and may affect overall performance. Turning off logging completely can be done by adding the following config option to ceph.conf:

```
rgw enable ops log = false
```

Another way to reduce the logging load is to send operations logging data to a unix domain socket, instead of writing it to the RADOS backend:

```
rgw ops log rados = false
rgw enable ops log = true
rgw ops log socket path = <path to socket>
```

When specifying a unix domain socket, it is also possible to specify the maximum amount of memory that will be used to keep the data backlog:

```
rgw ops log data backlog = <size in bytes>
```

Any backlogged data in excess to the specified size will be lost, so socket needs to be constantly read.

ENABLING SWIFT ACCESS

Allowing access to the object store with Swift (OpenStack Object Storage) compatible clients requires an additional step, the creation of a subuser and a Swift access key.

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

```
{ "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": []}
```

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

```
{ "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"}]}
```

This step enables you to use any Swift client to connect to and use RADOS Gateway via the Swift-compatible API. As an example, you might use the `swift` command-line client utility that ships with the OpenStack Object Storage packages.

```
swift -V 1.0 -A http://radosgw.example.com/auth -U johndoe:swift -K E9T2rUZNu2gxUjcwUBO8n\/Ev
swift -V 1.0 -A http://radosgw.example.com/auth -U johndoe:swift -K E9T2rUZNu2gxUjcwUBO8n\/Ev
```

RGW’s `user:subuser` tuple maps to the `tenant:user` tuple expected by Swift.

Note: RGW’s Swift authentication service only supports built-in Swift authentication (`-V 1.0`). To make RGW authenticate users via OpenStack Identity Service (Keystone), see below.

INTEGRATING WITH OPENSTACK KEYSTONE

It is possible to integrate RGW with Keystone, the OpenStack identity service. This sets up RGW to accept Keystone as the users authority. A user that Keystone authorizes to access RGW will also be automatically created on RGW (if didn’t exist beforehand). A token that Keystone validates will be considered as valid by RGW.

The following config options are available for Keystone integration:

```
[client.radosgw.gateway]
        rgw keystone url = {keystone server url:keystone server admin port}
        rgw keystone admin token = {keystone admin token}
        rgw keystone accepted roles = {accepted user roles}
        rgw keystone token cache size = {number of tokens to cache}
        rgw keystone revocation interval = {number of seconds before checking revoked tickets
        nss db path = {path to nss db}
```

An RGW user is mapped into a Keystone `tenant`. A Keystone user has different roles assigned to it on possibly more than a single tenant. When RGW gets the ticket, it looks at the tenant, and the user roles that are assigned to that ticket, and accepts/rejects the request according to the `rgw keystone accepted roles` configurable.

Keystone itself needs to be configured to point to RGW as an object-storage endpoint:

```
keystone service-create --name swift --type-object-store
keystone endpoint-create --service-id <id> --publicurl http://radosgw.example.com/swift/v1 \
        --internalurl http://radosgw.example.com/swift/v1 --adminurl http://radosgw.example.c
```

The keystone url is the Keystone admin RESTful api url. The admin token is the token that is configured internally in Keystone for admin requests.

RGW will query Keystone periodically for a list of revoked tokens. These requests are encoded and signed. Also, Keystone may be configured to provide self signed tokens, which are also encoded and signed. RGW needs to be able to decode and verify these signed messages, and it requires it to be set up appropriately. Currently, RGW will be able to do it only if it was compiled with `--with-nss`. It also requires converting the OpenSSL certificates that Keystone uses for creating the requests to the nss db format, for example:

```
mkdir /var/ceph/nss

openssl x509 -in /etc/keystone/ssl/certs/ca.pem -pubkey | \
        certutil -d /var/ceph/nss -A -n ca -t "TCu,Cu,Tuw"
openssl x509 -in /etc/keystone/ssl/certs/signing_cert.pem -pubkey | \
        certutil -d /var/ceph/nss -A -n signing_cert -t "TCu,Cu,Tuw"
```

ENABLING SUBDOMAIN S3 CALLS

To use RADOS Gateway with subdomain S3 calls (e.g., `http://bucketname.hostname`), you must add the RADOS Gateway DNS name under the `[client.radosgw.gateway]` section of your Ceph configuration file:

```
[client.radosgw.gateway]
        ...
        rgw dns name = {hostname}
```

You should also consider installing Dnsmasq on your client machine(s) when using `http://{bucketname}.{hostname}` syntax. The `dnsmasq.conf` file should include the following settings:

```
address=/{hostname}/{host-ip-address}
listen-address={client-loopback-ip}
```

Then, add the `{client-loopback-ip}` IP address as the first DNS nameserver on client the machine(s).