INSTALL CEPH OBJECT GATEWAY

As of *firefly* (v0.80), Ceph Object Gateway is running on Civetweb (embedded into the `ceph-radosgw` daemon) instead of Apache and FastCGI. Using Civetweb simplifies the Ceph Object Gateway installation and configuration.

Note: To run the Ceph Object Gateway service, you should have a running Ceph storage cluster, and the gateway host should have access to the public network.

Note: In version 0.80, 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.

EXECUTE THE PRE-INSTALLATION PROCEDURE

See Preflight and execute the pre-installation procedures on your Ceph Object Gateway node. Specifically, you should disable `requiretty` on your Ceph Deploy user, set SELinux to `Permissive` and 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.

INSTALL CEPH 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>
```

CREATE A GATEWAY INSTANCE

From the working directory of your administration server, create an instance of the Ceph Object Gateway on the Ceph Object Gateway. For example:

```
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 like this:

```
http://client-node:7480
```

If the gateway instance is working properly, you 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 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.

CHANGE THE 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 the working directory of your administration server. Add a section entitled `[client.rgw.<gateway-node>]`, replacing `<gateway-node>` with the short node name of your Ceph Object Gateway node (i.e., `hostname -s`).

Note: As of version 11.0.1, the Ceph Object Gateway **does** support SSL. See Using SSL with Civetweb for information on how to set that up.

For example, if your node name is `gateway-node1`, add a section like this after the `[global]` section:

```
[client.rgw.gateway-node1]
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-node1]` 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-node1`.

Push the updated configuration file to your Ceph Object Gateway node (and other Ceph nodes):

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

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

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

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. If you use the `firewalld` daemon, execute:

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

If you use `iptables`, execute:

```
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 the `root` user:

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

USING SSL WITH CIVETWEB

Before using SSL with civetweb, you will need a certificate that will match the host name that that will be used to access the Ceph Object Gateway. You may wish to obtain one that has *subject alternate name* fields for more flexibility. If you intend to use S3-style subdomains (Add Wildcard to DNS), you will need a *wildcard* certificate.

Civetweb requires that the server key, server certificate, and any other CA or intermediate certificates be supplied in one file. Each of these items must be in *pem* form. Because the combined file contains the secret key, it should be protected from unauthorized access.

To configure ssl operation, append `s` to the port number. For eg:

```
[client.rgw.gateway-node1]
rgw_frontends = civetweb port=443s ssl_certificate=/etc/ceph/private/keyandcert.pem
```

New in version Luminous.

Furthermore, civetweb can be made to bind to multiple ports, by separating them with `+` in the configuration. This allows for use cases where both ssl and non-ssl connections are hosted by a single rgw instance. For eg:

```
[client.rgw.gateway-node1]
rgw_frontends = civetweb port=80+443s ssl_certificate=/etc/ceph/private/keyandcert.pem
```

MIGRATING FROM APACHE TO CIVETWEB

If you are running the Ceph Object Gateway on Apache and FastCGI with Ceph Storage v0.80 or above, you are already running Civetweb–it starts with the `ceph-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 similar as the following:

On Red Hat Enterprise Linux:

```
[client.radosgw.gateway-node1]
host = {hostname}
keyring = /etc/ceph/ceph.client.radosgw.keyring
rgw socket path = ""
log file = /var/log/radosgw/client.radosgw.gateway-node1.log
rgw frontends = fastcgi socket\_port=9000 socket\_host=0.0.0.0
rgw print continue = false
```

On Ubuntu:

```
[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 settings such as `rgw_socket_path` and `rgw_print_continue`. 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-node1]
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. On Red Hat Enterprise Linux execute:

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

On Ubuntu execute:

```
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.

CONFIGURE BUCKET SHARDING

A Ceph Object Gateway stores bucket index data in the `index_pool`, which defaults to `.rgw.buckets.index`. Sometimes users like to put many objects (hundreds of thousands to millions of objects) in a single bucket. If you do not use the gateway administration interface to set quotas for the maximum number of objects per bucket, the bucket index can suffer significant performance degradation when users place large numbers of objects into a bucket.

In Ceph 0.94, you may shard bucket indices to help prevent performance bottlenecks when you allow a high number of objects per bucket. The `rgw_override_bucket_index_max_shards` setting allows you to set a maximum number of shards per bucket. The default value is `0`, which means bucket index sharding is off by default.

To turn bucket index sharding on, set `rgw_override_bucket_index_max_shards` to a value greater than `0`.

For simple configurations, you may add `rgw_override_bucket_index_max_shards` to your Ceph configuration file. Add it under `[global]` to create a system-wide value. You can also set it for each instance in your Ceph configuration file.

Once you have changed your bucket sharding configuration in your Ceph configuration file, restart your gateway. On Red Hat Enteprise Linux execute:

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

On Ubuntu execute:

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

For federated configurations, each zone may have a different `index_pool` setting for failover. To make the value consistent for a zonegroup's zones, you may set `rgw_override_bucket_index_max_shards` in a gateway's zonegroup configuration. For example:

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

Open the `zonegroup.json` file and edit the `bucket_index_max_shards` setting for each named zone. Save the `zonegroup.json` file and reset the zonegroup. For example:

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

Once you have updated your zonegroup, update and commit the period. For example:

```
radosgw-admin period update --commit
```

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

ADD 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 DNS configuration works as expected:

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

For example:

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

ADD DEBUGGING (IF NEEDED)

Once you finish the setup procedure, if you encounter issues with your configuration, you can add debugging to 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
```

USING THE 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.

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.

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": []
}
```

ACCESS VERIFICATION

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 yum install python-boto
```

2. Create the Python script:

```
vi s3test.py
```

3. Add the following contents to the file:

```
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='{hostname}', 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} {created}".format(
        name=bucket.name,
        created=bucket.creation_date,
    )
```

Replace `{hostname}` with the hostname of the host where you have configured the gateway service i.e., the `gateway host`. 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
```

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 commands. On Red Hat Enterprise Linux:

```
sudo yum install python-setuptools
sudo easy_install pip
sudo pip install --upgrade setuptools
sudo pip install --upgrade python-swiftclient
```

On Debian-based distributions:

```
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+fz2gSqoHwR3lYtSbIyomyPH
```

The output should be:

```
my-new-bucket
```