Understanding a Multi Site Ceph Gateway Installation

How to perform a Multi Site Ceph Gateway Installation and understand the configuration parameters including how to handle fail-over and fall-back in case of a disaster (Jewel or RHCS V2.0 and later)

Walter Graf

walter.graf@ts.fujitsu.com

Version: 19.01.2017

Abstract: With the major rework of the Ceph gateway software in the Jewel release it became necessary to revisit the installation and configuration process for S3 and Swift deployments. Although some documentation is already available on the Internet most of them do not convey a deeper understanding of the various configuration parameters. This applies in particular for the fail-over and fall-back process in case of a disaster. This white paper aims at improving this situation.

Keywords: Ceph, Jewel, Red Hat Ceph Storage V2.0, Ceph Gateway, Rados Gateway, Multi Site, Multi Cluster, Realm, Zonegroup, Zone, S3, Swift, Load-balancer, Fail-over, Fall-back

Content

1 Introduction

This white paper is arranged in three major parts. The first one deals with a short recipe style description of a sample Multi Site Ceph Gateway installation while the second one will convey a deeper understanding of the configuration parameters and underlying concepts so that we are better able to design different configurations. A third and last part deals with handling fail-over and fall-back processes in case of a disaster.

We assume enough Ceph knowledge to install required software packages, create Ceph users and pools etc. Therefore these topics are not covered in this paper. Also load-balancer and DNS network configurations are not covered in greater detail because the environments will change substantially from installation to installation.

We also assume some basic know how on the S3 protocol and how to use it.

Some tests have been performed on a Fujitsu Storage ETERNUS CD10000 system, a reference architecture based on Red Hat Ceph Storage V2.0 running on RHEL 7 at the time of testing.

Most of the commands are expected to work across multiple Linux distributions like CentOS, Ubuntu, and SLES. The most obvious differences are centred around the various system start-up methods like systemd, upstart etc.

2 Installing the sample configuration

Without further ado we will run now through the installation and configuration process of the sample system

2.1 The sample system

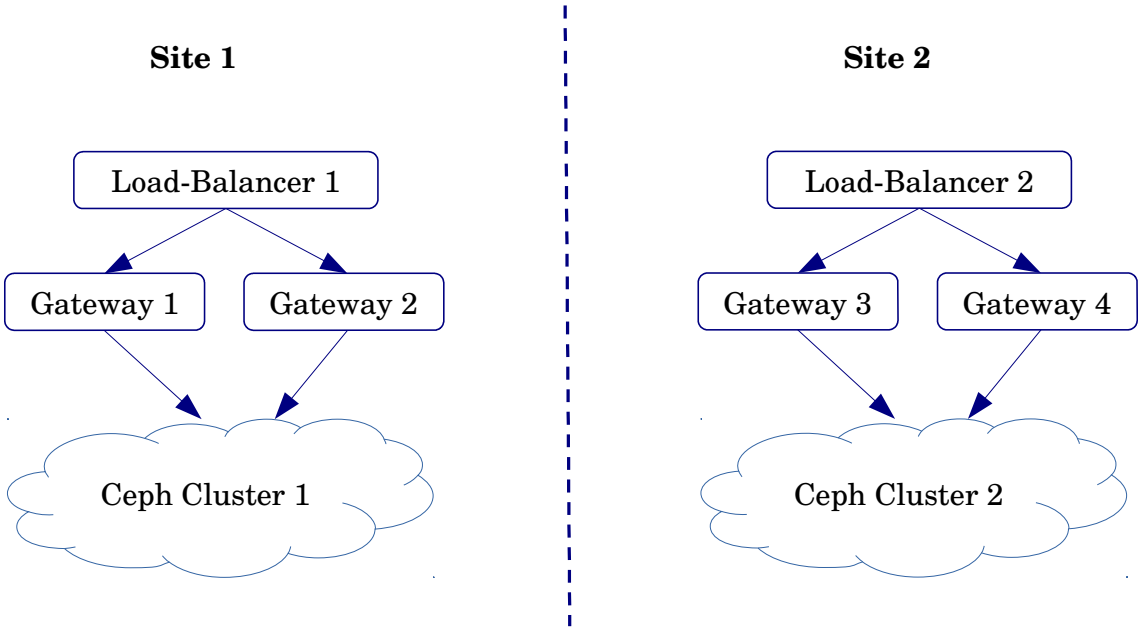

Figure 1: The sample system

Our sample system consists of two geographically dispersed and independent Ceph clusters. Every Ceph cluster is used by two gateway nodes (either implemented as physical or virtual machines). Always two gateway nodes (belonging to one cluster) will be driven by one load-balancer.

2.2 The sample configuration

From a software standpoint of view both sites form one entity. This concept is supported by the notion of a "realm" that forms a container for "zonegroups" (actually one zonegroup in our example). In turn the zonegroup[1] can consist of multiple zones. In our example it consists of two zones, one master zone and one secondary zone that are reachable via defined *http* endpoints as depicted in the next figure.

1 The notion of a zonegroup resembles the notion of a region in previous Ceph gateway implementations (i.e. before the Ceph Jewel release)

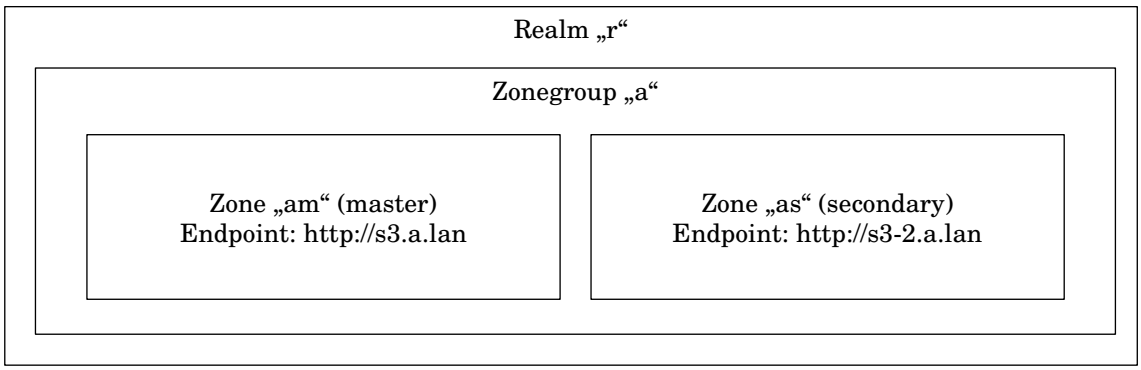

Figure 2: The notions of realm, zonegroup, and zones

The master zone is implemented on the two gateways of the first Ceph cluster, the secondary zone runs on the two gateways of the second cluster.

2.3 Preparing the environment

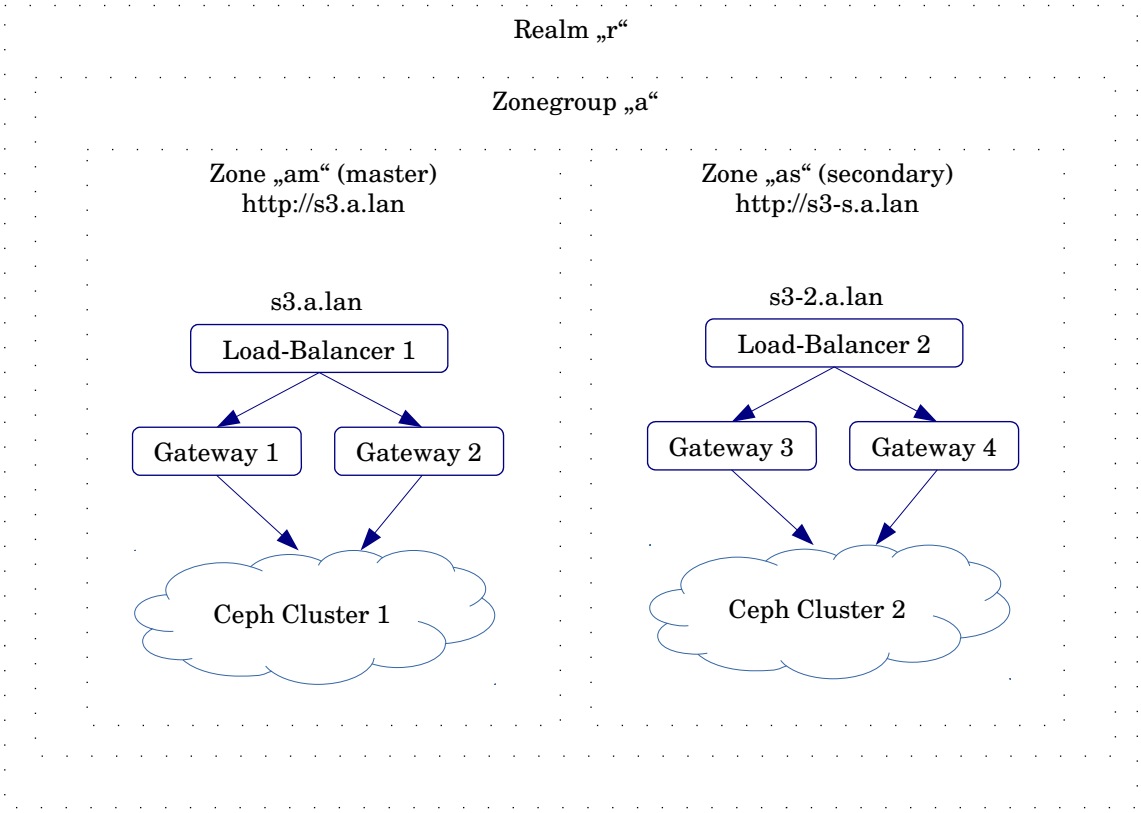

Figure 3: Environment overview

Gateways 1 and 2 should be able to access the Ceph cluster 1 as Ceph clients while gateways 3 and 4 should be able to access the Ceph cluster 2 as Ceph clients.

DNS name resolution[2] should direct all requests to *s3.a.lan* to load-balancer 1 while all requests to *s3-2.a.lan* should be directed to load-balancer 2. The

2 For a better understanding of name resolution requirements please refer to a later chapter

load-balancers should in turn direct *http* requests to the respective gateways as shown in the figure above.

2.4 Preparing the two Ceph clusters

We need a set of Ceph pools defined in both clusters. Although non existing pools are created automatically by the gateways as needed it is recommended to create the pools upfront in order to actively influence important parameters like number of placement groups and redundancy schemes like erasure coding. In particular the following pools should exist on both clusters:

Pools in Cluster 1	Pools in Cluster 2
.rgw.root	.rgw.root
a.root	a.root
am.rgw.data.root	as.rgw.data.root
am.rgw.control	as.rgw.control
am.rgw.gc	as.rgw.gc
am.rgw.log	as.rgw.log
am.rgw.intent-log	as.rgw.intent-log
am.rgw.usage	as.rgw.usage
am.rgw.users.keys	as.rgw.users.keys
am.rgw.users.email	as.rgw.users.email
am.rgw.users.swift	as.rgw.users.swift
am.rgw.users.uid	as.rgw.users.uid
am.rgw.buckets.index	as.rgw.buckets.index
am.rgw.buckets.data	as.rgw.buckets.data
am.rgw.buckets.non-ec	as.rgw.buckets.non-ec
am.rgw.meta	as.rgw.meta

In addition four Ceph users need to be created:

- *client.radosgw.am1* on cluster 1
- *client.radosgw.am2* on cluster 1
- *client.radosgw.as1* on cluster 2
- c*lient.radosgw.as2* on cluster 2

All Ceph users must be able to write and read to the pools on the respective clusters.

2.5 Preparing the gateways

In our example all gateways are assumed to have the Ceph gateway software installed. The following environment needs to be prepared for the next installation steps:

- On gateway 1 (cluster 1, zone *am*):
  - the output of `hostname -s` is *gw1*
  - the */etc/ceph/ceph.conf* file of cluster 1 is present
  - the keyring file of Ceph user *client.radosgw.am1* is installed in */etc/ceph/ceph.client.radosgw.am1.keyring*
- On gateway 2 (cluster 1, zone *am*):
  - the output of `hostname -s` is *gw2*
  - the */etc/ceph/ceph.conf* file of cluster 1 is present
  - the keyring file of Ceph user *client.radosgw.am2* is installed in */etc/ceph/ceph.client.radosgw.am2.keyring*
- On gateway 3 (cluster 2, zone *as*):
  - the output of `hostname -s` is *gw3*
  - the */etc/ceph/ceph.conf* file of cluster 2 is present
  - the keyring file of Ceph user *client.radosgw.as1* is installed in */etc/ceph/ceph.client.radosgw.as1.keyring*
- On gateway 4 (cluster 2, zone *as*):
  - the output of `hostname -s` is *gw4*
  - the */etc/ceph/ceph.conf* file of cluster 2 is present
  - the keyring file of Ceph user *client.radosgw.as2* is installed in */etc/ceph/ceph.client.radosgw.as2.keyring*

2.6 Adapting */etc/ceph/ceph.conf* on all gateways

In this step we will add a gateway section to the local copy of

```
/etc/ceph/ceph.conf
```

on every gateway as follows (example: gateway1 on cluster 1):

```
[client.radosgw.am1]
rgw data = /var/lib/ceph/radosgw/ceph-radosgw.am1[3]
rgw zonegroup = a
rgw zone = am
rgw zonegroup root pool = a.root
rgw zone root pool = a.root
keyring = /etc/ceph/ceph.client.radosgw.am1.keyring
host = gw1
rgw dns name = s3.a.lan
log file = /var/log/radosgw/client.radosgw.am1.log
rgw frontends = civetweb port=80 num_threads=150
rgw override bucket index max shards = 20
```

The parameters

- rgw data (*...am1*, *...am2*, *...as1*, *...as2*
- rgw zone (*am*, *am*, *as*, *as*)
- keyring (*...am1.keyring*, *...am2.keyring*, *...as1.keyring*, *...as2.keyring*)
- host (*gw1*, *gw2*, *gw3*, *gw4*)
- log file (*...am1.log*, *...am2.log*, *...as1.log*, *...as2.log*

must be adapted on the other gateways accordingly. In addition *rgw dns name* needs to be adapted as well:

- On gateways 1/2: *rgw dns name* = *s3.a.lan*
- On gateways 3/4: *rgw dns name* = *s3-2.a.lan*

Now and with all Ceph users, pools, and ceph.conf sections in place on every gateway we are ready for the actual configuration work.

3 This directory should already exist as part of the software installation. If not, please create it

2.7 Creating the master zone

In the following we will

- create the realm *r*
- create the zonegroup *a*
- create the master zone *am*
- create the system user *system*
- update the period

All of the following will be executed on gateway 1:

```
# id=radosgw.am1
```

create realm

```
# radosgw-admin --id $id realm create --rgw-realm=r --default
```

create zonegroup

```
# radosgw-admin --id $id zonegroup create --rgw-realm=r \
--rgw-zonegroup=a --endpoints=http://s3.a.lan:80 \
--master --default
```

create master zone

```
# radosgw-admin --id $id zone create --rgw-realm=r \
--rgw-zonegroup=a --rgw-zone=am --endpoints=http://s3.a.lan:80 \
--master --default \
--access-key=system-access –secret=do-not-tell-anybody
```

create system user

```
# radosgw-admin --id $id user create --uid="system" \
--display-name="System User" --access-key[4]=system-access \
--secret=do-not-tell-anybody --system
```

update the period

```
# radosgw-admin --id $id period update --commit
```

As a last step before starting the gateway daemon on both master zone gateways we define at least one normal S3 user, e.g.:

```
# radosgw-admin --id $id user create --uid="wall-e" \
--display-name="Robot Wall-E" --access-key=wall-e-access \
--secret=waste-allocation-load-lifter
```

Now we will enable and start the radosgw daemon on gateways 1 and 2, i.e. on gateway 1 we execute (on RHEL 7):

```
# systemctl enable ceph-radosgw@radosgw.am1
# systemctl enable ceph-radosgw.target
# systemctl start ceph-radosgw.target
```

4 For security reasons a different naming method for access and secret keys will be explained in the “Deep Dive” section of this paper

and on gateway 2 we execute

```
# systemctl enable ceph-radosgw@radosgw.am2
# systemctl enable ceph-radosgw.target
# systemctl start ceph-radosgw.target
```

At this point in time the system is already capable of serving S3 requests sent to *http://s3.a.lan*.

2.8 Creating the secondary zone

For the next steps it is important that the master zone is configured as described in the last chapter and that the gateway daemon on both gateways 1 and 2 is running and able to properly service S3 requests.

In the following we will

- pull the realm to cluster2
- pull the period to cluster 2
- create the secondary zone
- and update the period

In our example the secondary zone runs on the gateways 3 and 4 and stores its data in cluster 2. Therefore the following steps will all be performed on gateway 3:

```
# id=radosgw.as1
```

pull the realm

```
# radosgw-admin --id $id realm pull --rgw-realm=r \
--url=http://s3.a.lan --default
--access-key=system-access --secret=do-not-tell-anybody
```

pull the period

```
# radosgw-admin --id $id period pull --rgw-realm=r \
--url=http://s3.a.lan \
--access-key=system-access --secret=do-not-tell-anybody
```

create the secondary zone *as*

```
# radosgw-admin --id $id zone create --rgw-realm=r \
--rgw-zonegroup=a --rgw-zone=as --endpoints=http://s3-2.a.lan \
--access-key=system-access --secret=do-not-tell-anybody
```

update the period

```
# radosgw-admin --id $id period update --commit
```

Now we will enable and start the radosgw daemon on gateways 3 and 4, i.e. on gateway 3 we execute (on RHEL 7):

```
# systemctl enable ceph-radosgw@radosgw.as1
# systemctl enable ceph-radosgw.target
# systemctl start ceph-radosgw.target
```

and on gateway 2 we execute

```
# systemctl enable ceph-radosgw@radosgw.as2
# systemctl enable ceph-radosgw.target
# systemctl start ceph-radosgw.target
```

At this point in time the system is now also capable of serving S3 requests sent to *http://s3-2.a.lan*. The master zone and the secondary zone will synchronise data, i.e. data written to the master zone will be automatically replicated to the secondary zone and vice versa. The S3 users we defined in the master zone will also be propagated to the secondary zone.

3 The Deep Dive

3.1 Understanding network name resolution requirements

There are two ways to address a bucket in S3:

- via a virtual-host-style URL, e.g. *http://my_bucket.s3.a.lan*. If a request is made to the *http://my_bucket.s3.a.lan* endpoint, the DNS must have sufficient information to route the request directly to the S3 instance that services it.
- via a path-style URL, e.g. *http://s3.a.lan/my_bucket*. In this style the bucket name is not part of the hostname. Therefore the DNS is only responsible for resolving the hostname *s3.a.lan*.

Because applications use one of these address styles it is recommended to configure DNS for both. From a DNS perspective the virtual-host-style URL is the critical one, because the name service needs to translate every name of the style *{bucket}.s3.a.lan* to the same IP address.

In our load-balancer based example it will be the IP address of the respective load-balancer (and not the gateway!). However, the gateway's section in the *ceph.conf* must still contain the line

```
rgw dns name = s3.a.lan
```

(for gateways 1 and 2), although s3.a.lan resolves to the load-balancer. Otherwise the gateway daemon will not process the S3 request.

The easiest but less attractive way is editing the */etc/hosts file* on every involved S3 client system. A simple example to define name resolution in */etc/hosts* could look like:

```
10.172.144.210 s3.a.lan bucket1.s3.a.lan bucket2.s3.a.lan
10.172.144.211 s3-2.a.lan bucket1.s3-2.a.lan bucket2.s3-2.a.lan
```

The disadvantage is obvious: We have to explicitly define name resolution for every bucket. The next and better method is to define a wild-carding scheme with DNS. There are at least two ways of achieving this

- using *dnsmasq*
- using *bind*

The following example will configure, enable, and start the dnsmasq service on another system, e.g. on 172.17.33.193.

- edit */etc/dnsmasq.conf* and add:

```
address=/.s3.a.lan/10.172.144.210
address=/.s3-2.a.lan/10.172.144.211
listen-address=127.0.0.1
```

```
listen-address=172.17.33.193
```

- edit */etc/resolv.conf* on the dnsmasq nameserver and add

```
nameserver 127.0.0.1
```

- edit */etc/resolv.conf* on very involved system (gateways, S3 clients, etc.) and add:

```
nameserver 172.17.33.193
```

- enable and start the dnsmasq service on the nameserver

```
# systemctl enable dnsmasq
# systemctl start dnsmasq
```

Please note that it might be necessary to adapt the firewall on the involved systems (*firewalld*, *iptables*, etc.)

The most common solution will probably be to integrate the bucket name resolution into the company's standard *bind* service. In this situation the following wild-card DNS record needs to be added to *bind* (e.g. for *s3.a.lan*):

```
$TTL    604800
@       IN      SOA     s3.a.lan. root.s3.a.lan. (
                              2         ; Serial
                         604800         ; Refresh
                          86400         ; Retry
                        2419200         ; Expire
                         604800 )       ; Negative Cache TTL
;
@        IN      NS      s3.a.lan.
@        IN      A        10.172.144.210
*        IN      CNAME    @
```

Bucket names must be DNS compliant because they may be part of a host-name in case of a virtual-host-style URL. In particular this means

- Bucket names must be at least 3 and no more than 63 characters long.
- Bucket names can contain **lower-case** letters, numbers, and hyphens. They must start and end with a lower-case letter or a number.
- Bucket names **must not** be formatted as an IP address (e.g., 192.168.5.4).
- When using virtual-host-style buckets with SSL, the SSL wild-card certificate only matches buckets that do not contain periods. It is recommended not to use periods (".") in bucket names.

3.2 Configuring S3 clients

The following credentials are necessary and need to be configured in an S3 client application to successfully connect to an S3 service:

- the access key, typically a 20 character string
- the secret key, typically a 40 character string
- the web protocol, *http* or *https*
- the port name, e.g. 80 for *http* or 443 for *https*
- the endpoint's DNS name or IP address, e.g. *s3.a.lan* or 10.172.144.210
- some S3 clients expect a bucket to be created upfront (see next chapter how to use a simple S3 client for this purpose)

The last three parameters are typically summarised in one string like in *http://s3.a.lan:80*.

In our sample configuration the access and secret key have already been defined on one of the master zone gateways as part of the S3 user definition:

```
# id=radosgw.am1
# radosgw-admin --id $id user create --uid="wall-e" \
--display-name="Robot Wall-E" --access-key=wall-e-access \
--secret=waste-allocation-load-lifter
```

The access key *wall-e-access* and the secret key *waste-allocation-load-lifter* can be used by any S3 client to access the master zone or one of the secondary zones. For security reasons it is not recommended to use access and secret keys like in this example but instead use a random pattern generation scheme like

```
# cat /dev/urandom | tr -dc 'a-zA-Z0-9' | fold -w 20 | head -n 1
```

for the access key and

```
# cat /dev/urandom | tr -dc 'a-zA-Z0-9' | fold -w 40 | head -n 1
```

for the secret key. Since the Jewel version of Ceph it is now also safely[5] possible to have radosgw-admin create the keys like in

```
# id=radosgw.am1
# radosgw-admin --id $id user create --uid="wall-e" \
--display-name="Robot Wall-E" --gen-access-key --gen-secret \
```

Note: It is important to create S3 users in the master zone because only then they are automatically propagated to the secondary zones.

5 Former versions of radosgw-admin were also able to generate keys but the resulting keys very often needed escape characters as soon as they were used in the CLI or configuration files.

The uid *wall-e* is only used by the gateway software and doesn't need to be disclosed to the S3 client.

Note: Don't mix up S3 users with Ceph users. S3 users are only known to the gateway daemon (and the S3 client) and are not used to access the Ceph cluster.

In our sample configuration we are using a web server that is embedded in the gateway daemon, called *civetweb*. At the time of writing the civetweb version in use only supported *http*, but not *https*. What is possible, however, is to let the load-balancer perform a *https* to *http* conversion.

Note: Some S3 clients require public SSL certificates for *https* communication. This can cause a problem in test environments using privately generated certificates.

S3 clients using virtual-host-style URLs must be able to resolve all possible *{bucket}.{fully-qualified-domain-name}* combinations (see previous chapter).

S3 clients should be able to work with custom defined endpoints like *http://myvault.mycompany.com* instead of being restricted to Amazon hosts (e.g. *s3.amazonaws.com*).

Some S3 clients do also expect pre-defined buckets. These can easily be created using one of the simple S3 clients that are explained in the next chapter.

3.3 How to use the simple client s3cmd

A good way to demonstrate the insights of the last chapter is to install, configure, and use one of the simple S3 clients called *s3cmd*. It can be downloaded from *http://s3tools.org* in form of a zipped tar file, e.g.

```
http://sourceforge.net/projects/s3tools/files/s3cmd/1.6.0/s3cmd-
1.6.0.tar.gz
```

s3cmd is an CLI based S3 application for Linux that uses the virtual-host-type URL scheme. Therefore DNS must be set-up properly as described in the previous chapters.

The following steps will be performed on an S3 client system running Linux. Download the tar file file, e.g. using *wget* and install it with

```
# tar –zxvf s3cmd-1.6.1.tar.gz
# cd s3cmd-1.6.1/
# rsync -a s3cmd S3 $HOME/bin
```

In a next create and edit a configuration file *a.cfg* that *s3cmd* will use

```
# cd
# s3cmd --configure –c a.cfg
```

In this configuration file *a.cfg* we will adapt the following entries

```
access_key = wall-e-access
secret_key = waste-allocation-load-lifter
host_base = s3.a.lan
host_bucket = %(bucket)s.s3.a.lan
```

and mark the following lines as comments

```
#proxy_host =
#proxy_port = 0
#cloudfront_host =
#simpledb_host =
#website_endpoint =
```

According to whether *http* or *https* is used the entry

```
use_https = …
```

has to be set to *True* or *False*. In case *https* is chosen and a private certificate (see previous chapter) is in use the following entry needs to be configured

```
check_ssl_certificate = False
```

s3cmd is now ready to use and as a first test the following command sequence should execute without problems:

- list buckets

```
# cd
# s3cmd -c a.cfg ls
```

- make and list bucket

```
# s3cmd -c a.cfg mb s3://targaryen
# s3cmd -c a.cfg ls
```

- create object in bucket and list

```
# dd if=/dev/zero of=/tmp/10m.bin bs=1024k count=10
```

```
# s3cmd -c a.cfg put /tmp/10m.bin s3://targaryen/daenerys
# s3cmd -c a.cfg ls
```

- list all objects belonging to this S3 user

```
# s3cmd -c a.cfg la
```

- retrieve object and test (files should be identical)

```
# s3cmd -c a.cfg get s3://targaryen/daenerys
# diff daenerys /tmp/10m.bin
```

- clean up the test files, objects, and buckets

```
# rm daenerys /tmp/10m.bin
```

```
# s3cmd -c a.cfg rm s3://targaryen/daenerys
# s3cmd -c a.cfg rb s3://targaryen
# s3cmd -c a.cfg ls
# s3cmd -c a.cfg la
```

Using simple S3 applications like s3cmd and executing these short tests also help to quickly verify that an installation is ready for configuring more complex S3 applications.

Another simple application that uses path-style URLs is CloudBerry (*http://www.cloudberrylab.com/free-amazon-s3-explorer-cloudfront-IAM.aspx*) that runs on Windows.

3.4 Principles of operation for radosgw-admin and radosgw

In general *radosgw-admin* is the command line tool to configure the Ceph gateway environment and also retrieve information about it. The gateway daemon *radosgw* then provides the S3 service and acts according to this configuration.

Therefore the usual process is to first set-up the gateway using *radosgw-admin* and then start the gateway daemon.

Configuration data is stored in two places:

Fundamental data is stored in the respective gateway section in */etc/ceph/ceph.conf*. This fundamental data could also be viewed as bootstrap information. In fact this information is not only used by the gateway daemon but also by *radosgw-admin*. Therefore the very first step in configuring any gateway is manually editing the respective gateway section in */etc/ceph/ceph.conf* (even before calling *radosgw-admin* the first time). Every gateway (or better: every gateway daemon) has its own section in *ceph.conf*. In addition it is also important to perform any configuration work with *radosgw.admin* using the *--id* option, e.g.

```
# radosgw-admin --id radosgw.am1 zonegroup create …
```

The option *--id* instructs *radosgw-admin* to use the definitions in the *ceph.conf* gateway section *[client.radosgw.am1]* (the string *client* can be omitted while using --id). Later changes in ceph.conf gateway sections require a restart of the gateway daemon because the sections are only read during the start-up phase, e.g.

```
# systemctl restart ceph-radosgw@radosgw.am1
```

All other configuration data

- is stored by *radosgw-admin* in Ceph pools
- may be read again from the Ceph pools by *radosgw-admin*

- is read from the Ceph pools and used by the gateway daemon

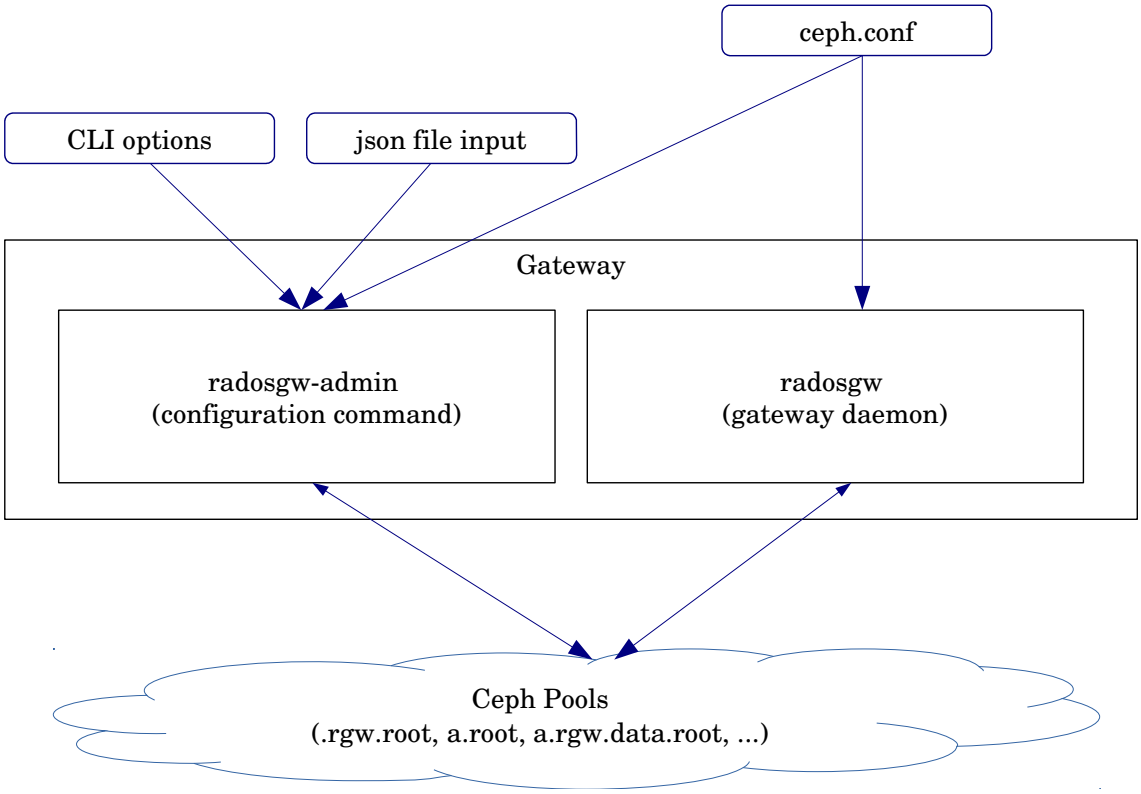

Figure 4: Gateway configuration data flow

Because the bulk configuration data is stored in Ceph pools data can be easily shared by multiple gateways operating on the same realm, zonegroup, and zone. For example in our sample configuration it is sufficient to perform the configuration for zone *am* just on gateway 1. Gateway 2 (belonging to the same zone) will access the same Ceph pools and is hereby configured automatically (except the initial set-up of its gateway section in */etc/ceph/ceph.conf*!)

radosgw-admin accepts two forms of input:

- via CLI options
- via json files

For example it is possible to change an existing zone configuration via

```
# id=radosgw.as1
# radosgw-admin --id $id zone modify --rgw-zone=as \
--master --default
```

(promoting zone *as* as new master like it is performed during a later described fail-over and fall-back procedure).

On the contrary it is also possible to manually create a json file, e.g. *zone.json* and *set* the zone using:

```
# id = radosgw.as1
# radosgw-admin --id $id zone set <zone.json
```

This is, however, a more complex process and modifying a zone based on CLI options is certainly the easier method. Of course it is also possible to retrieve the current zone configuration via

```
# radosgw-admin --id $id zone get >zone.json
```

and use the *zone.json* output as template for a subsequent *zone set* command. This is particularly useful to obtain a backup of existing zone configurations (as it will be used in a later chapter on fail-over / fall-back procedures). For normal configuration work, the preferred method should be using CLI options.

Regardless whether *radosgw-admin* is used via CLI options or json input files the resulting configuration output is stored in the form of objects in Ceph pools.

After the gateway daemon is started up it acts according to the configuration present in these Ceph pools. In addition the gateway daemon also stores all other data like S3 objects, buckets, log data etc. in Ceph pools. All this data is shared among all gateways that serve the same zone. Based on this collaboration scheme and together with a load-balancer it is possible to have multiple gateways serve the same zone via one endpoint (e.g. *s3.a.lan*).

3.5 User names and other naming conventions

In the chapter on configuring S3 clients we have already come across the notion of an S3 user. This user is not known to the Ceph cluster itself but only known to the gateway daemon and *radosgw-admin*.

In order that both instances, the gateway daemon and *radosgw-admin* can access the Ceph cluster they need the necessary authorisation in the form of a Ceph user (not an S3 user). As a general rule every gateway (or better: every gateway daemon) needs one dedicated Ceph user that will also be used by the accompanying *radosgw-admin* command. In our sample configuration we used the Ceph user *client.radosgw.am1* for gateway 1 (which serves the master zone *am*). That user was created via standard Ceph commands like

```
# user=client.radosgw.am1
# ceph-authtool --create-keyring /etc/ceph/ceph.$user.keyring \
--gen-key -n $user --cap osd 'allow rwx' --cap mon 'allow rwx'
# ceph auth add client.radosgw.$user -i /etc/ceph/ceph.$user.keyring
```

The whole gateway architecture now requires that this gateway specific Ceph user is also the name of the gateway section in */etc/ceph/ceph.conf*. This is the reason why every gateway needs its own Ceph user that is also

identical to the respective section in */etc/ceph.conf* that is used to load the initial settings at the gateway daemon start-up time (or while calling *radosgw-admin*).

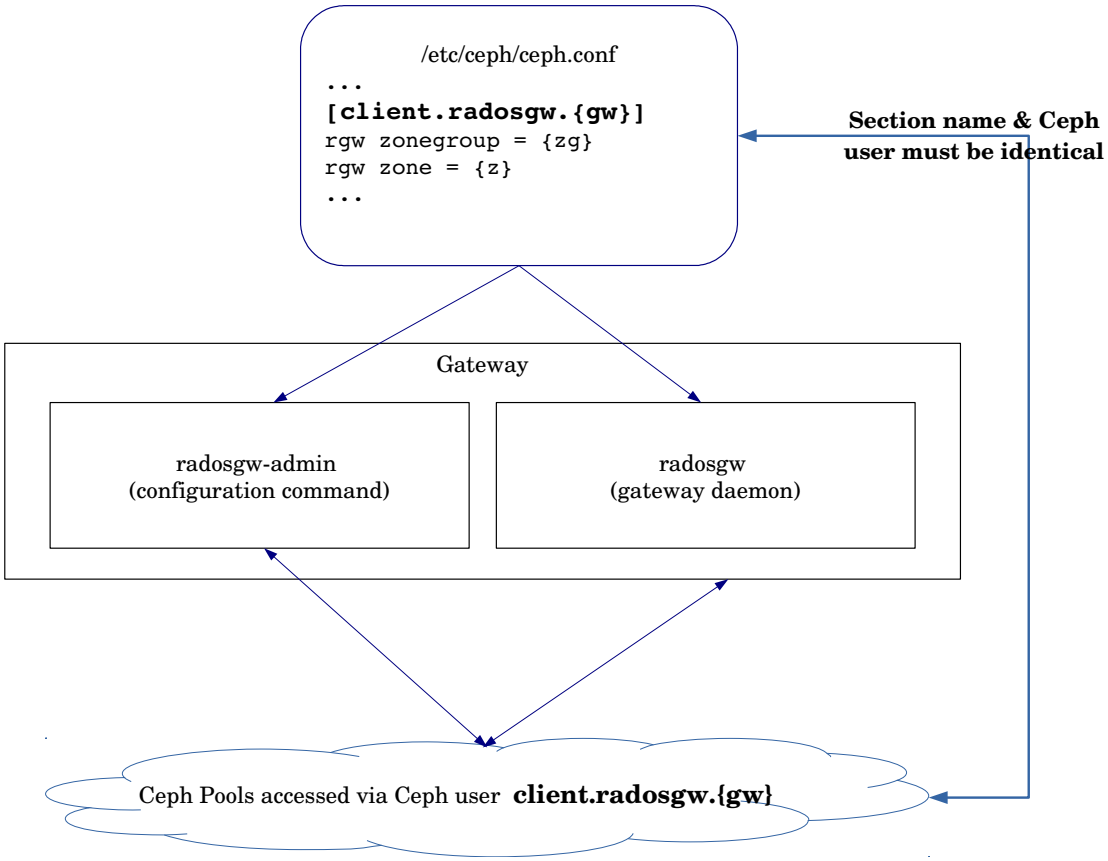

Figure 5: Gateway section name and Ceph user must be identical

In addition the following naming rule applies:

- The gateway section name (and therefore the user) must start with the string *client.radosgw*, i.e. must follow the convention *client.radosgw.{gw}*.

Background is that various Ceph mechanisms are based on this naming convention:

- Normal Ceph users always start with the string *client.*
- The start-up process on a gateway only scans sections in *ceph.conf* that start with the string *client.radosgw*.

As a consequence

radosgw-admin must always be called with either the option *--name client.radosgw.{gw}*[6] or *--id*[7] *radosgw.{gw}*, e.g. *radosgw-admin --id radosgw.am1*.

6 {gw} stands for any name of the gateway service, e.g. am1
7 --id is a short form of --name saving the specification of the string *client*

Using e.g. *systemd* a gateway daemon is directly started via *systemctl start ceph-radosgw@radosgw.{gw}*

A useful scheme for {gw} could be {zg}-{z}-{gwid}, e.g.

- *a-am-1* for the first gateway of the master zone
- *a-am-2* for the second gateway of the master zone
- *a-a1-1* for the first gateway of the first secondary zone
- *a-a1-2* for the second gateway of the second secondary zone
- etc.

In our sample configuration we used a shorter naming scheme defining {gw} as *am1, am2, as1, as2* respectively.

3.6 A closer look at ceph.conf

As outlined in the last chapter the gateway section in */etc/ceph/ceph.conf* contains the initial bootstrap information for the gateway daemon and *radosgw-admin*. Let's have a closer look at the gateway section of gateway one in our sample configuration:

```
[client.radosgw.am1]
rgw data = /var/lib/ceph/radosgw/ceph-radosgw.am1[8]
rgw zonegroup = a
rgw zone = am
rgw zonegroup root pool = a.root
rgw zone root pool = a.root
keyring = /etc/ceph/ceph.client.radosgw.am1.keyring
host = gw1
rgw dns name = s3.a.lan
log file = /var/log/radosgw/client.radosgw.am1.log
rgw frontends = civetweb port=80 num_threads=150
rgw override bucket index max shards = 20
```

The meaning of the section name *client.radosgw.am1* and its naming rules have already been explained in the last chapter. As stated there the section name is also the Ceph user name the gateway and *radosgw-admin* use to access the Ceph cluster. The *keyring* parameter must therefore contain the corresponding keyring information for this Ceph user.

rgw data points to a data directory that should be created upfront if it doesn't already exist.

rgw zonegroup and *rgw zone* define the zonegroup and zone the gateway daemon is supposed to live in and that are defined in more detail using radosgw-admin (see also the sample configuration process).

8 This directory should already exist as part of the software installation. If not, please create it

rgw zonegroup root pool and *rgw zone root pool* define the Ceph pools the gateway daemon reads its configuration data. radosgw-admin will also store the zonegroup and zone definitions in these pools. It is common practice to use just one pool for both parameters, e.g. a.root in our sample configuration. Although there is no specific naming convention required it is good practice to stick to the convention used in our sample configuration.

rgw dns name is the DNS name that resolves to the IP address under which the S3 service of the gateway daemon can be reached. In case a load-balancer is involved it must resolve to the IP address of the load-balancer and not the gateway. For more details on network name resolution please refer to the respective chapter. *rgw dns name* is not necessarily identical to the gateway daemon *hostname* (see also next parameter).

host is the exact output of the command *hostname -s* on the gateway. For example in our sample configuration *hostname -s* outputs *gw1* on the first gateway of the master zone. When using e.g. *systemd* to start up the gateway daemon using *systemctl start ceph-radosgw.target* the file */etc/ceph/ceph.conf* is searched for all gateway daemon sections starting with *client.radosgw*. In a second step the *host* parameter is checked ensuring that only daemons will start that belong to the host where the start-up process is actually executed.

rgw frontends specifies the web server in use. In our case it is civetweb, a built-in web server that runs as part of the gateway daemon. An alternative in former versions was to use Apache which required a different parameter set. The parameter *num_threads* and also the parameter *rgw override bucket index max shards* are the result of performance tests that have been carried out. The parameter *rgw override bucket index max shards* should be sized according to the expectation of the maximum number of objects per bucket. The recommended value is one shard per 100k objects in a bucket.

By default all realm and period information is stored in the pool *.rgw.root* and we have not changed this in our sample configuration. However, it is possible to reconfigure this location by defining the parameters *rgw realm root pool* and *rgw period root pool*.

3.7 The concept of realm, zonegroup, zone, and period

As already explained during the installation of our sample configuration some new notions have been introduced to the Ceph gateway architecture as part of the Ceph *Jewel* release.

The new top level structure is now called *realm*, which may contain one or more *zonegroups*. Every *zonegroup* may contain one or more *zones*.

The previously known term *region* (as in pre-*Jewel* releases) has been replaced by the term *zonegroup* in order to avoid misunderstandings regarding the normally anticipated geographical meaning of a *region*. A *zonegroup* can easily span multiple locations and is simply a collection of *zones*. One or more gateways (gateway daemons) provide the S3 services for a particular zone.

In our sample configuration we defined a realm *r* containing a zonegroup *a*. The zonegroup *a* consists of the zones *am* and *as*. The zone *am* is served by *gw1* and *gw2* while zone *as* is served by *gw3* and *gw4*.

New since the *Jewel* release is that multiple zones belonging to the same zonegroup can now work in an active-active synchronisation mode, i.e. S3 clients can write to every zone with all newly written data being synchronised with the remaining zones.

In addition configuration changes can be propagated more easily across multiple gateways even if they belong to different Ceph clusters. There is no need to manually edit, copy, and inject complex json files as it was the case in older Ceph versions.

The *master zone* of the *master zonegroup* is responsible for propagating configuration changes. Every *realm* has one *master zonegroup* and every *zonegroup* has a *master zone*.

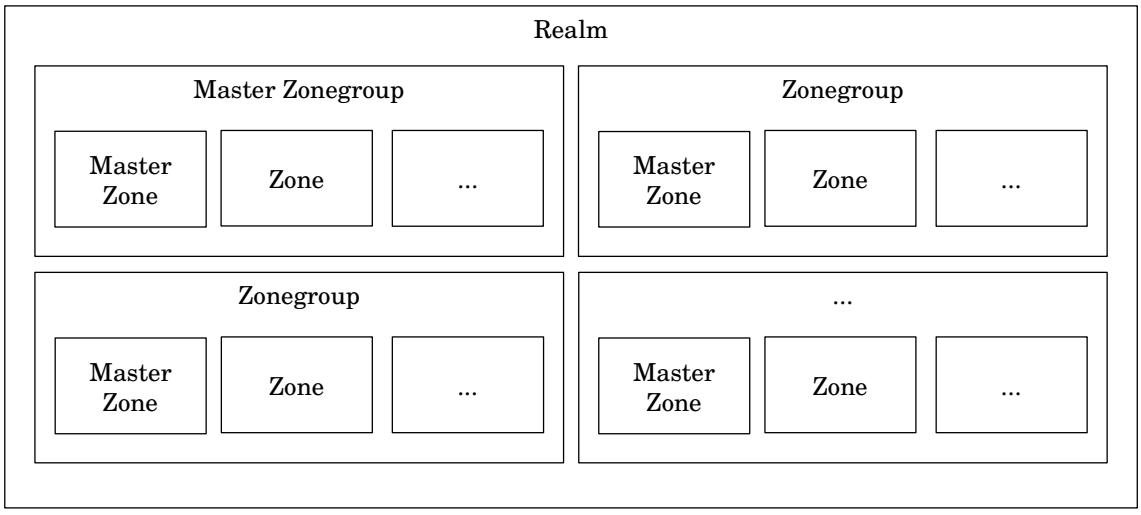

Figure 6: Realm, Master Zonegroup, Zonegroups, Master Zone, Zones

As a consequence the whole process[9] of fail-over and fall-back is much simpler now.

9 For details, please refer to the chapter dealing with fail-over and fall-back

All this has been made possible by the notion of a *period*. A *period* is a configuration state of the *realm* including its *zonegroups* and *zones* that is valid for a certain period of time.

If for example the *master zone* changes as part of a fail-over, a new *period* is defined and propagated. The new *period* also contains a reference to the previous *period*.

If for example just a new *zone* is added or an *endpoint* is modified the *period* remains the same but it will increment its *epoch*, i.e. a new version of the same *period* is activated.

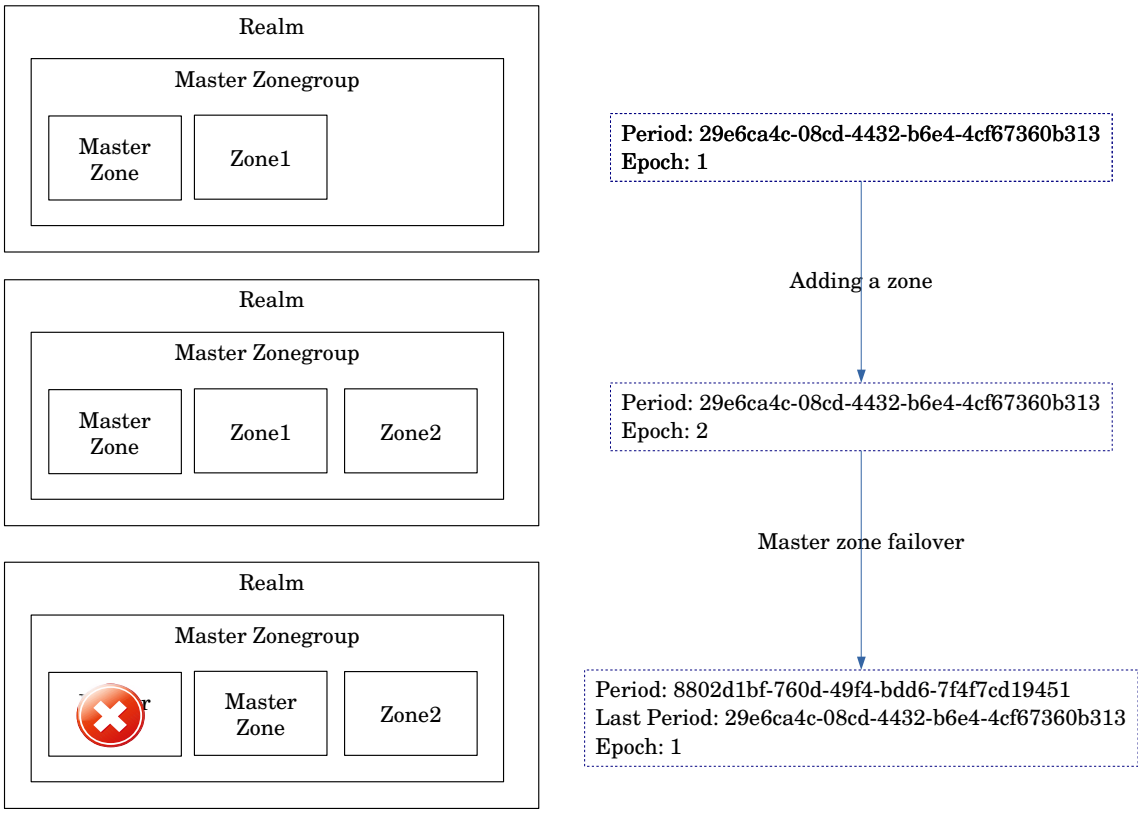

Figure 7: Periods and Epochs

In order to activate a new *period* or *epoch increment* after initiating configuration changes like

```
# id=radosgw.am1
# radosgw-admin --id $id zone create …
```

or

```
# radosgw-admin --id $id zone modify …
```

etc. you have to update and commit the period with

```
# radosgw-admin --id $id --rgw-realm=r[10] period update --commit
```

10 --rgw-realm can be omitted in case *r* is the default realm

As a result the new configuration will be propagated by the master zone to all of its peers.

All *realm* and *period* information is stored in the Ceph pool *.rgw.root*[11]. All gateway daemons or *radosgw-admin* instances running on the same Ceph cluster can easily access this information. As soon as gateway daemons (and their zones) use different Ceph clusters the situation becomes slightly more complicated.

When we look back to the installation process of our sample configuration we notice that before we created the secondary zone we issued the two commands

```
# id=radosgw.as1
# radosgw-admin --id $id realm pull --rgw-realm=r \
--url=http://s3.a.lan --default
--access-key=system-access --secret=do-not-tell-anybody
```

and

```
# radosgw-admin --id $id period pull --rgw-realm=r \
--url=http://s3.a.lan \
--access-key=system-access --secret=do-not-tell-anybody
```

These two commands *pull* the *realm* and *period* related information from the cluster where the master zone resides to the other cluster where the secondary zone resides. Please note the options *--url*, *--access-key*, and *--secret*. *--url* specifies the endpoint that is served by the master zone and *--access-key* and *--secret* specify the system credentials to authenticate ourselves with the master zone.

Once *realm* and *period* informations are established on the new cluster's *.rgw.root* pool, the secondary zone is created, and the *period* is updated and committed. No further pull is necessary because all future configuration changes are propagated automatically by the master zone to its peers.

In addition to configuration changes that always require a *period update* and *commit* operation there are a number of other operations that are automatically propagated or replicated to all zones of a zonegroup.

For example when an S3 user is created with

```
# id=radosgw.am1
# radosgw-admin --id $id user create --uid=...
```

the user is also propagated to the remaining zones of the zonegroup.

Note: User propagation will only happen if the command is executed on one of the master zone gateways.

11 or in the pool defined by the parameters *rgw realm root pool* and *rgw period root pool* in the respective gateway section of *ceph.conf*

Although *bucket* creation is possible in all zones it needs the master zone committing this operation. This handshake is automatically handled by the zones, but if the master zone fails and no other zone has been promoted to master in the meantime no *bucket* creation is possible.

S3 objects, however, can be created even in the situation of a failed master zone.

3.8 Starting Ceph gateway daemons using systemctl

In RHEL 7.x *systemd* is used as system start-up method and also Ceph gateway daemons are started using the command *systemctl*. The common method is to use

```
# systemctl enable …
```

to enable the service and use

```
# systemctl start …
```

to actually start a daemon and

```
# systemctl stop …
```

to stop it.

It is either possible to enable, start, stop, etc. a daemon directly by using the naming convention

```
ceph-radosgw@radosgw.{gw}
```

where the string *radosgw.{gw}* corresponds to the section in */etc/ceph/ceph.conf* as explained earlier. For example

```
# systemctl start ceph-radosgw@radosgw.am1
```

starts the Ceph gateway daemon on *gw1* in our sample configuration.

Another way would be to start all Ceph gateway daemons belonging to a particular gateway system using

```
# systemctl start ceph-radosgw.target
```

This command will scan all gateway sections in */etc/ceph/ceph.conf* that belong to the particular *host* (i.e. all sections with a *host* entry that corresponds to the *hostname -s* output) and start it.

Note: While retrieving the status of a Ceph gateway daemon using

```
# systemctl status …
```

the output of e.g.

```
# systemctl status ceph-radosgw.am1
```

is more precise than using

```
# systemctl status ceph-radosgw.target
```

3.9 Two Ceph clusters vs. one Ceph cluster

In our sample configuration we established zones in different Ceph clusters. This guarantees maximum data segregation and resilience against data or system outages. It should be the recommended way of operating a multi zone / multi site installation.

However, it is also possible to implement the architecture in just one cluster. The following recommendations apply in this case:

- The Ceph cluster should span multiple fire zones or even data centres, i.e. so called *failure domains*.
- These *failure domains* should be implemented in the *crush map* hierarchy accordingly.
- All Ceph pools belonging to one zone should be mapped to *crush rules* ending up in distinct *failure domains* without overlapping with pools belonging to other zones.

This leads to a design in which data belonging to different pools are completely segregated and a data outage in one zone will have no impact on another zone. Care should be taken that latencies between the *failure domains* do not become a bottleneck. In case the distances between the *failure domains* become too large (and therefore the latencies grow accordingly) a multi cluster design should be preferred over an implementation in a single cluster.

4 Fail-over and Fall-back

When looking at the sample configuration we can state that most failures do not require any form of fail-over and fall-back procedures:

- Load-balancers are typically highly available and resilient against hardware failures.
- Single gateways can fail (or be restored) without causing any system interruption.
- All data (even the more complex realm, zonegroup, and zone configuration data) are stored in the Ceph cluster with no single point of failure. Any Ceph cluster component can fail (or be restored) without causing any interruption or data loss.

The only disaster that is worth a fail-over / fall-back consideration is the outage of a complete site. We will therefore have a closer look at two main scenarios:

- The temporary outage of one site
- A longer (more permanent) outage of one site

4.1 Managing a temporary site outage

A typical reason for a temporary site outage is a complete loss of power. It is realistic here to assume that normal operation can be resumed in a matter of hours (or days in the worst case).

Since the new functionality in the Ceph *Jewel* release (and its enterprise derivatives from Red Hat etc.) it is not necessary to consider a fail-over / fall-back procedure in this case at all.

If the site containing the master[12] zone goes down the secondary sites continue to run and are able to serve read and write requests as usual. Only the creation of new buckets and users is not possible while the master zone is not available.

It maybe necessary to reconfigure the application instances using the *http* address of the failed master zone (*http://s3.a.lan* in our sample configuration) to use the *http* address of one of the secondary zones (*http://s3-2.a.lan* in our sample configuration) during the outage.

After the failed site has come back again it is sufficient to simply restart the gateway instances. In our sample configuration we would then execute

```
# systemctl start ceph-radosgw.target
```

12 The outage of the secondary (i.e. non-master) site is even more trivial and is not covered here.

on gateways *gw1* and *gw2*.

Please note that we assume that the Ceph cluster has not suffered any data loss (at the site containing the master zone in our sample configuration). The newly started daemons on *gw1* and *gw2* will simply use the existing (unchanged) configuration and re-synchronise with *gw3* and *gw4* from the secondary zone.

Please do also note that no changes have been applied to the configuration during the system outage. In particular we assume that the secondary zone has not been promoted as new master zone during the outage.

Because the vast majority of applications using the S3 API will use a static set of buckets and users that rarely changes this is the best way to deal with a temporary site outage.

The more disastrous and longer outage of a site with major data loss is covered in the following chapter.

4.2 Managing a longer site outage including data loss

Even in case of a longer site outage that involves a major data loss the surviving sites will continue to run and stay operational.

If the lost site hosts a secondary (i.e. non-master) zone there is no fail-over necessary because the surviving sites are able to perform all read and write operations including the creation of new buckets and S3 users (because the master zone is still active and operational).

If the lost site hosts the master zone the surviving sites are also able to perform all read and write operations. Only new buckets and users cannot be created.

Please note that application instances using the endpoints of the failed zone do no longer work. For example in our sample configuration requests to *http://s3.a.lan* are no longer processed in case the zone *am* is no longer available. As a consequence the application instances using the endpoints of the failed zone should be re-configured.

As a next step it should now be carefully considered if the downtime of the failed site and the usage scenario really requires one of the secondary zones to be promoted as the new master in order to allow the creation of new buckets and users. The recommendation would be to wait with this activity until it becomes inevitable.

Let's assume we decided for this in order to restore bucket and user capabilities. In our sample configuration we will then promote[13] the only remaining zone *as* as new master zone with

```
# id=radosgw.as1
# radosgw-admin --id $id zone modify --rgw-zone=as \
--master --default --read-only[14]=False
```

and we do not forget to update and commit the period with

```
# radosgw-admin --id $id period update --commit
```

Because we performed a master zone change this will introduce a new period.

Please note that a restart of the daemon is not necessary this time and we can immediately work with the new master zone, e.g. create new buckets or users.

Now, let's have a look at the fall-back case, that can start as soon as the failed site comes up again. The following instructions (valid for our sample configuration) will be easier to follow if a regular backup of the zone configuration files has been been performed, e.g. executing the following commands on *gw1* (for zone *am*)

```
# id=radosgw.am1
# radosgw-admin --id $id zone get --rgw-zone=am >zone-am.json
```

and storing *zone-am.json* at a save place (and performing the same procedure on e.g. *gw3* for zone *as* with file *zone-as.json*).

At the time we start our fall-back process we have a recovered site with

- the Ceph cluster at the failed site being up and running again (in the worst case with all data lost, i.e. all pools newly defined but empty)
- all gateways in place with the necessary software installed but not yet configured
- all Ceph users in place on every gateway
- the */etc/ceph/ceph.conf* configuration file reconstructed with the original gateway section on every gateway

For the following we assume that we have lost the original master zone *am* in our sample configuration:

13 To be executed e.g. on *gw3*

14 This option is only necessary if the zone has been set to read-only before

In a first step we pull the realm and period to the still empty and new Ceph cluster on *gw1*

```
# id=radosgw.am1
# radosgw-admin --id $id realm pull \
--url=http://s3-2.a.lan --default \
--access=system-access --secret=do-not-tell-anybody
# radosgw-admin --id $id period pull \
--url=http://s3-2.a.lan \
--access=system-access --secret=do-not-tell-anybody
```

Please note the *--url*, *--access-key*, and *--secret* options that describe from where to pull the information and how to authenticate.

Next, we set the zone and update / commit the period:

```
# id=radosgw.am1
# radosgw-admin --id $id zone set --rgw-zone=am <zone-am.json[15]
# radosgw-admin --id $id period update --commit
```

Now we start the gateway daemon on both gateways *gw1* and *gw2* with:

```
# systemctl start ceph-radosgw.target
```

Please note that we did not yet restore the zone am *as* master zone. It is still coming up as secondary zone (assuming we had the zone *as* promoted as master before).

Before we restore the status of *am* as the original master we wait until the resynchronisation is complete by watching the output of

```
# id=radosgw.am1
# radosgw-admin --id $id sync status
```

After synchronisation is complete we restore the original role of *am* as master zone by executing (on e.g. *gw1*)

```
# id=radosgw.am1
# radosgw-admin --id $id zone modify --rgw-zone=am \
--master --default --read-only[16]=False
```

and we do not forget to update and commit the period with

```
# radosgw-admin --id $id period update --commit
```

With zone *am* being the master zone again we remove the master zone status of the previous master zone on *gw3* accordingly:

```
# id=radosgw.as1
# radosgw-admin --id $id zone modify --rgw-zone=as \
--master=false
# radosgw-admin --id $id period update --commit
```

15 This is the file we previously backed up
16 This option is only necessary if the zone has been set to read-only before

As a very last step we can reconfigure those application instances that are supposed to work with zone *am* and have previously been re-routed to the secondary zone.

Now, we have safely recovered from a major disaster even involving a major data loss and longer outages.

5 Acknowledgements

I would like to thank Dieter Meyer from Fujitsu for sharing some early insights of how to set-up a multi site Ceph gateway environment in Ceph Jewel and Ursula Menke from Fujitsu for providing the test environment.

I would also like to thank Orit Wasserman from Red Hat for providing precious input that helped me to increase the quality of this paper.