RADOS GATEWAY DATA LAYOUT

Although the source code is the ultimate guide, this document helps new developers to get up to speed with the implementation details.

INTRODUCTION

Swift offers something called a container, that we use interchangeably with the term bucket. One may say that RGW's buckets implement Swift containers.

This document does not consider how RGW operates on these structures, e.g. the use of encode() and decode() methods for serialization and so on.

CONCEPTUAL VIEW

Although RADOS only knows about pools and objects with their xattrs and omap[1], conceptually RGW organizes its data into three different kinds: metadata, bucket index, and data.

METADATA

We have 3 'sections' of metadata: 'user', 'bucket', and 'bucket.instance'. You can use the following commands to introspect metadata entries:

```
$ radosgw-admin metadata list
$ radosgw-admin metadata list bucket
$ radosgw-admin metadata list bucket.instance
$ radosgw-admin metadata list user

$ radosgw-admin metadata get bucket:<bucket>
$ radosgw-admin metadata get bucket.instance:<bucket>:<bucket_id>
$ radosgw-admin metadata get user:<user>   # get or set
```

Some variables have been used in above commands, they are:

- user: Holds user information
- bucket: Holds a mapping between bucket name and bucket instance id
- bucket.instance: Holds bucket instance information[2]

Every metadata entry is kept on a single rados object. See below for implementation defails.

Note that the metadata is not indexed. When listing a metadata section we do a rados pgls operation on the containing pool.

BUCKET INDEX

It's a different kind of metadata, and kept separately. The bucket index holds a key-value map in rados objects. By default it is a single rados object per bucket, but it is possible since Hammer to shard that map over multiple rados objects. The map itself is kept in omap, associated with each rados object. The key of each omap is the name of the objects, and the value holds some basic metadata of that object – metadata that shows up when listing the bucket. Also, each omap holds a header, and we keep some bucket accounting metadata in that header (number of objects, total size, etc.).

Note that we also hold other information in the bucket index, and it's kept in other key namespaces. We can hold the bucket index log there, and for versioned objects there is more information that we keep on other keys.

DATA

Objects data is kept in one or more rados objects for each rgw object.

OBJECT LOOKUP PATH

When accessing objects, ReST APIs come to RGW with three parameters: account information (access key in S3 or account name in Swift), bucket or container name, and object name (or key). At present, RGW only uses account information to find out the user ID and for access control. Only the bucket name and object key are used to address the object in a pool.

The user ID in RGW is a string, typically the actual user name from the user credentials and not a hashed or mapped identifier.

When accessing a user's data, the user record is loaded from an object "<user_id>" in pool "default.rgw.meta" with namespace "users.uid".

Bucket names are represented in the pool "default.rgw.meta" with namespace "root". Bucket record is loaded in order to obtain so-called marker, which serves as a bucket ID.

The object is located in pool "default.rgw.buckets.data". Object name is "<marker>_<key>", for example "default.7593.4_image.png", where the marker is "default.7593.4" and the key is "image.png". Since these concatenated names are not parsed, only passed down to RADOS, the choice of the separator is not important and causes no ambiguity. For the same reason, slashes are permitted in object names (keys).

It is also possible to create multiple data pools and make it so that different users buckets will be created in different rados pools by default, thus providing the necessary scaling. The layout and naming of these pools is controlled by a 'policy' setting. [3]

An RGW object may consist of several RADOS objects, the first of which is the head that contains the metadata, such as manifest, ACLs, content type, ETag, and user-defined metadata. The metadata is stored in xattrs. The head may also contain up to 512 kilobytes of object data, for efficiency and atomicity. The manifest describes how each object is laid out in RADOS objects.

BUCKET AND OBJECT LISTING

Buckets that belong to a given user are listed in an omap of an object named "<user_id>.buckets" (for example, "foo.buckets") in pool "default.rgw.meta" with namespace "users.uid". These objects are accessed when listing buckets, when updating bucket contents, and updating and retrieving bucket statistics (e.g. for quota).

See the user-visible, encoded class 'cls_user_bucket_entry' and its nested class 'cls_user_bucket' for the values of these omap entires.

These listings are kept consistent with buckets in pool ".rgw".

Objects that belong to a given bucket are listed in a bucket index, as discussed in sub-section 'Bucket Index' above. The default naming for index objects is ".dir.<marker>" in pool "default.rgw.buckets.index".

FOOTNOTES

[1] Omap is a key-value store, associated with an object, in a way similar to how Extended Attributes associate with a POSIX file. An object's omap is not physically located in the object's storage, but its precise implementation is invisible and immaterial to RADOS Gateway. In Hammer, one LevelDB is used to store omap in each OSD.

[2] Before the Dumpling release, the 'bucket.instance' metadata did not exist and the 'bucket' metadata contained its information. It is possible to encounter such buckets in old installations.

[3] The pool names have been changed starting with the Infernalis release. If you are looking at an older setup, some details may be different. In particular there was a different pool for each of the namespaces that are now being used inside the default.root.meta pool.

APPENDIX: COMPENDIUM

Known pools:

.rgw.root
: Unspecified region, zone, and global information records, one per object.

<zone>.rgw.control
: notify.<N>

<zone>.rgw.meta
: Multiple namespaces with different kinds of metadata:

  namespace: root

<bucket> .bucket.meta.<bucket>:<marker> # see put_bucket_instance_info()

The tenant is used to disambiguate buckets, but not bucket instances. Example:

```
.bucket.meta.prodtx:test%25star:default.84099.6
.bucket.meta.testcont:default.4126.1
.bucket.meta.prodtx:testcont:default.84099.4
prodtx/testcont
prodtx/test%25star
testcont
```

namespace: users.uid

Contains _both_ per-user information (RGWUserInfo) in “<user>” objects and per-user lists of buckets in omaps of “<user>.buckets” objects. The “<user>” may contain the tenant if non-empty, for example:

```
prodtx$prodt
test2.buckets
prodtx$prodt.buckets
test2
```

namespace: users.email

Unimportant

namespace: users.keys

47UA98JSTJZ9YAN3OS3O

This allows radosgw to look up users by their access keys during authentication.

namespace: users.swift

test:tester

<zone>.rgw.buckets.index

Objects are named “.dir.<marker>”, each contains a bucket index. If the index is sharded, each shard appends the shard index after the marker.

<zone>.rgw.buckets.data

default.7593.4__shadow_.488urDFerTYXavx4yAd-Op8mxehnvTI_1 <marker>_<key>

An example of a marker would be “default.16004.1” or “default.7593.4”. The current format is “<zone>.<instance_id>.<bucket_id>”. But once generated, a marker is not parsed again, so its format may change freely in the future.