ERASURE CODE

A Ceph pool is associated to a type to sustain the loss of an OSD (i.e. a disk since most of the time there is one OSD per disk). The default choice when creating a pool is *replicated*, meaning every object is copied on multiple disks. The Erasure Code pool type can be used instead to save space.

CREATING A SAMPLE ERASURE CODED POOL

The simplest erasure coded pool is equivalent to RAID5 and requires at least three hosts:

```
$ ceph osd pool create ecpool 12 12 erasure
pool 'ecpool' created
$ echo ABCDEFGHI | rados --pool ecpool put NYAN -
$ rados --pool ecpool get NYAN -
ABCDEFGHI
```

Note: the 12 in *pool create* stands for the number of placement groups.

ERASURE CODE PROFILES

The default erasure code profile sustains the loss of a single OSD. It is equivalent to a replicated pool of size two but requires 1.5TB instead of 2TB to store 1TB of data. The default profile can be displayed with:

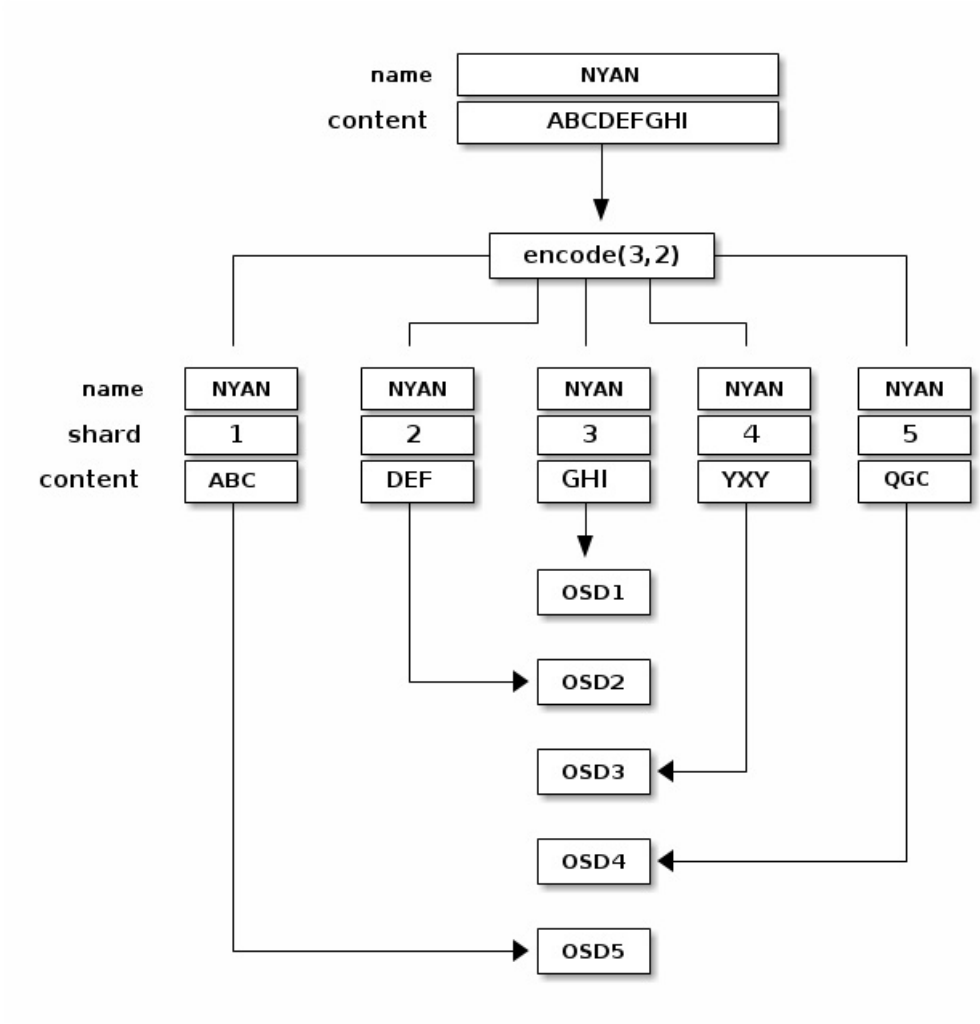
```
$ ceph osd erasure-code-profile get default
k=2
m=1
plugin=jerasure
crush-failure-domain=host
technique=reed_sol_van
```

Choosing the right profile is important because it cannot be modified after the pool is created: a new pool with a different profile needs to be created and all objects from the previous pool moved to the new.

The most important parameters of the profile are *K*, *M* and *crush-failure-domain* because they define the storage overhead and the data durability. For instance, if the desired architecture must sustain the loss of two racks with a storage overhead of 40% overhead, the following profile can be defined:

```
$ ceph osd erasure-code-profile set myprofile \
   k=3 \
   m=2 \
   crush-failure-domain=rack
$ ceph osd pool create ecpool 12 12 erasure myprofile
$ echo ABCDEFGHI | rados --pool ecpool put NYAN -
$ rados --pool ecpool get NYAN -
ABCDEFGHI
```

The *NYAN* object will be divided in three (*K=3*) and two additional *chunks* will be created (*M=2*). The value of *M* defines how many OSD can be lost simultaneously without losing any data. The *crush-failure-domain=rack* will create a CRUSH rule that ensures no two *chunks* are stored in the same rack.



More information can be found in the erasure code profiles documentation.

ERASURE CODING WITH OVERWRITES

By default, erasure coded pools only work with uses like RGW that perform full object writes and appends.

Since Luminous, partial writes for an erasure coded pool may be enabled with a per-pool setting. This lets RBD and CephFS store their data in an erasure coded pool:

```
ceph osd pool set ec_pool allow_ec_overwrites true
```

This can only be enabled on a pool residing on bluestore OSDs, since bluestore's checksumming is used to detect bitrot or other corruption during deep-scrub. In addition to being unsafe, using filestore with ec overwrites yields low performance compared to bluestore.

Erasure coded pools do not support omap, so to use them with RBD and CephFS you must instruct them to store their data in an ec pool, and their metadata in a replicated pool. For RBD, this means using the erasure coded pool as the `--data-pool` during image creation:

```
rbd create --size 1G --data-pool ec_pool replicated_pool/image_name
```

For CephFS, an erasure coded pool can be set as the default data pool during file system creation or via file layouts.

ERASURE CODED POOL AND CACHE TIERING

Erasure coded pools require more resources than replicated pools and lack some functionalities such as omap. To overcome these limitations, one can set up a cache tier before the erasure coded pool.

For instance, if the pool *hot-storage* is made of fast storage:

```
$ ceph osd tier add ecpool hot-storage
$ ceph osd tier cache-mode hot-storage writeback
$ ceph osd tier set-overlay ecpool hot-storage
```

will place the *hot-storage* pool as tier of *ecpool* in *writeback* mode so that every write and read to the *ecpool* are actually using the *hot-storage* and benefit from its flexibility and speed.

More information can be found in the cache tiering documentation.

GLOSSARY

chunk
: when the encoding function is called, it returns chunks of the same size. Data chunks which can be concatenated to reconstruct the original object and coding chunks which can be used to rebuild a lost chunk.

K
: the number of data *chunks*, i.e. the number of *chunks* in which the original object is divided. For instance if *K* = 2 a 10KB object will be divided into *K* objects of 5KB each.

M
: the number of coding *chunks*, i.e. the number of additional *chunks* computed by the encoding functions. If there are 2 coding *chunks*, it means 2 OSDs can be out without losing data.

TABLE OF CONTENT