Ceph Erasure Coding Introduction

By [Yuan Zhou (https://software.intel.com/en-us/user/496209)](https://software.intel.com/en-us/user/496209), published on April 6, 2015

Ceph introduction

Ceph, The Future of Storage™, is a massively scalable, open source, software-defined storage system that runs on commodity hardware. Ceph has been developed from the ground up to deliver object, block, and file system storage in a single software platform that is self-managing, self-healing and has no single point of failure. Because of its highly scalable, software defined storage architecture, Ceph is an ideal replacement for legacy storage systems and a powerful storage solution for object and block storage for cloud computing environments.

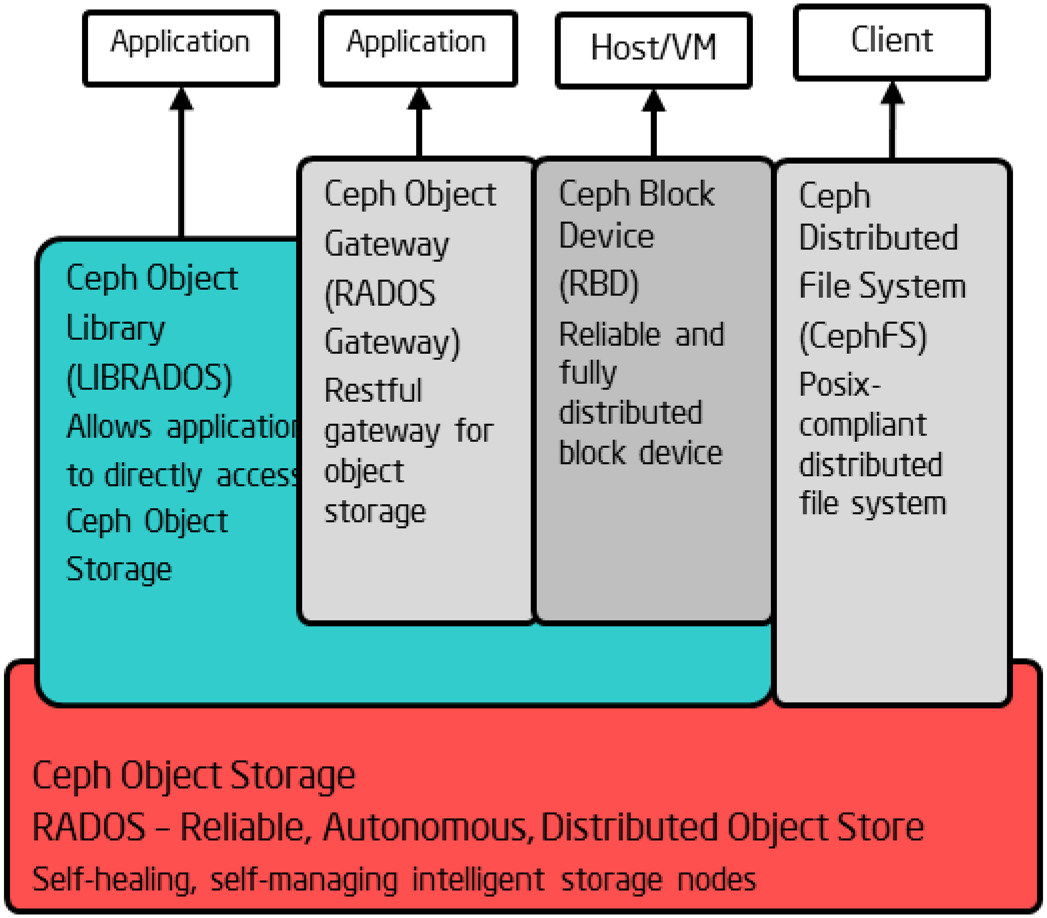
Ceph is started by Sage Weil's PHD paper: in June 2004. Currently Ceph belongs to RedHat. But it's still open source software. Ceph community is also very active ([https://github.com/ceph/ceph (https://github.com/ceph/ceph)](https://github.com/ceph/ceph)).

Below is the architecture of Ceph. The core is the RADOS (resilient automatic distributed object storage). Above RADOS, Ceph provides several interfaces:

- LibRADOS: it's the native API for Ceph, including read, write, append and truncate etc.
- RGW: it's the object storage API for Ceph, which is also RESTful and compatible with Swift and S3

- RBD: it’s the block storage API for Ceph. Currently its driver is already merge in linux kernel. It also provides the driver for QEMU.
- CephFS: it’s the filesystem API for Ceph. It’s POSIX compatible.

These interfaces are actually the ‘clients’ of the RADOS, since all of them are implemented with RADOS protocal.

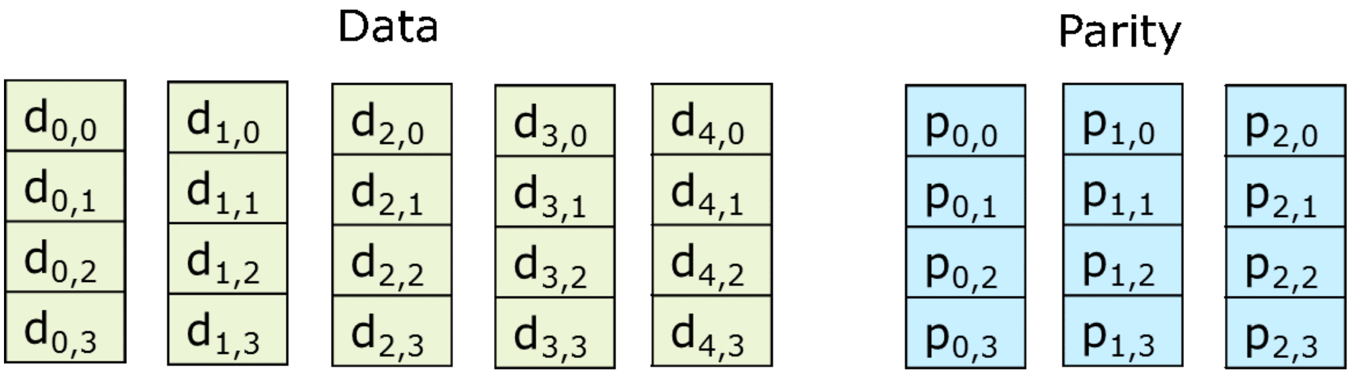

Application
Application
Host/VM
Client
Ceph Object Library (LIBRADOS)
Allows applicatio[n] to directly access Ceph Object Storage
Ceph Object Gateway (RADOS Gateway)
Restful gateway for object storage
Ceph Block Device (RBD)
Reliable and fully distributed block device
Ceph Distributed File System (CephFS)
Posix-compliant distributed file system
Ceph Object Storage
RADOS – Reliable, Autonomous, Distributed Object Store
Self-healing, self-managing intelligent storage nodes

Erasure code introduction

Erasure Code is a theory started at 1960s. The most famous algorithm is the Reed-Solomon. As time goes by, many variations came out, like the Fountain Codes, Pyramid Codes and Local Repairable Codes, etc.

Erasure Codes usually defines the number of total disks (N) and the number of data disks (K), and it can tolerate N – K failures with overhead of N/K

E,g, a typical Reed Solomon scheme: (8, 5), where 8 is the total disks, 5 is the data disks. In this case, the data in disks would be like:

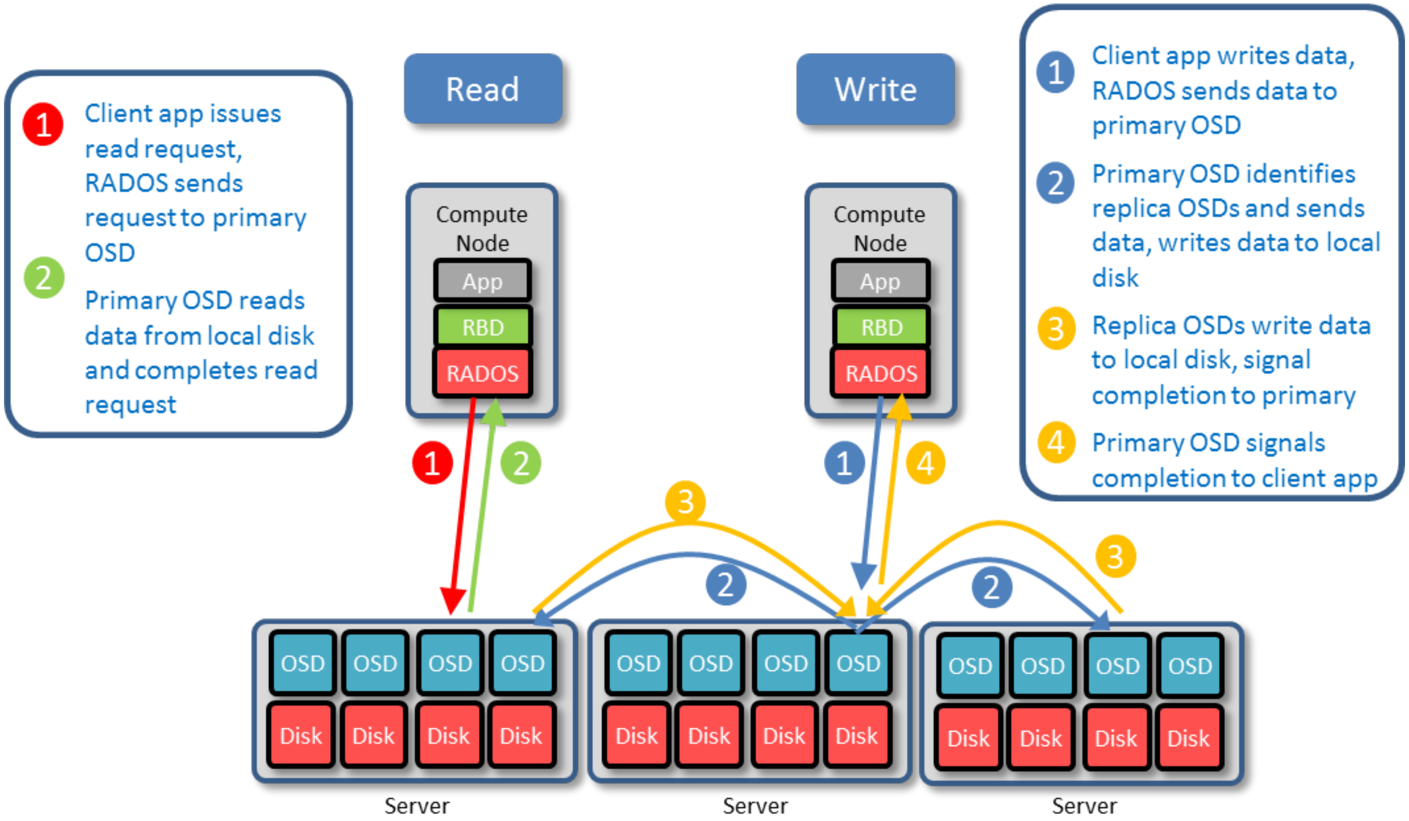

Data
Parity
$d_{0,0}$ $d_{0,1}$ $d_{0,2}$ $d_{0,3}$
$d_{1,0}$ $d_{1,1}$ $d_{1,2}$ $d_{1,3}$
$d_{2,0}$ $d_{2,1}$ $d_{2,2}$ $d_{2,3}$
$d_{3,0}$ $d_{3,1}$ $d_{3,2}$ $d_{3,3}$
$d_{4,0}$ $d_{4,1}$ $d_{4,2}$ $d_{4,3}$
$p_{0,0}$ $p_{0,1}$ $p_{0,2}$ $p_{0,3}$
$p_{1,0}$ $p_{1,1}$ $p_{1,2}$ $p_{1,3}$
$p_{2,0}$ $p_{2,1}$ $p_{2,2}$ $p_{2,3}$

RS (8, 5) can tolerate 3 arbitrary failures. If there's some data chunks missing, then one could use the rest available data to restore the original content.

In cloud storage, replication is commonly used to guarantee the availability. The issue is the storage requirement would be quite high if the storage goes to PB level. By using this technical much storage space can be saved while keep the same availability, which will dramatically save TCO. Ceph supports Erasure Code feature since Firefly version.

How does Ceph support Erasure Code

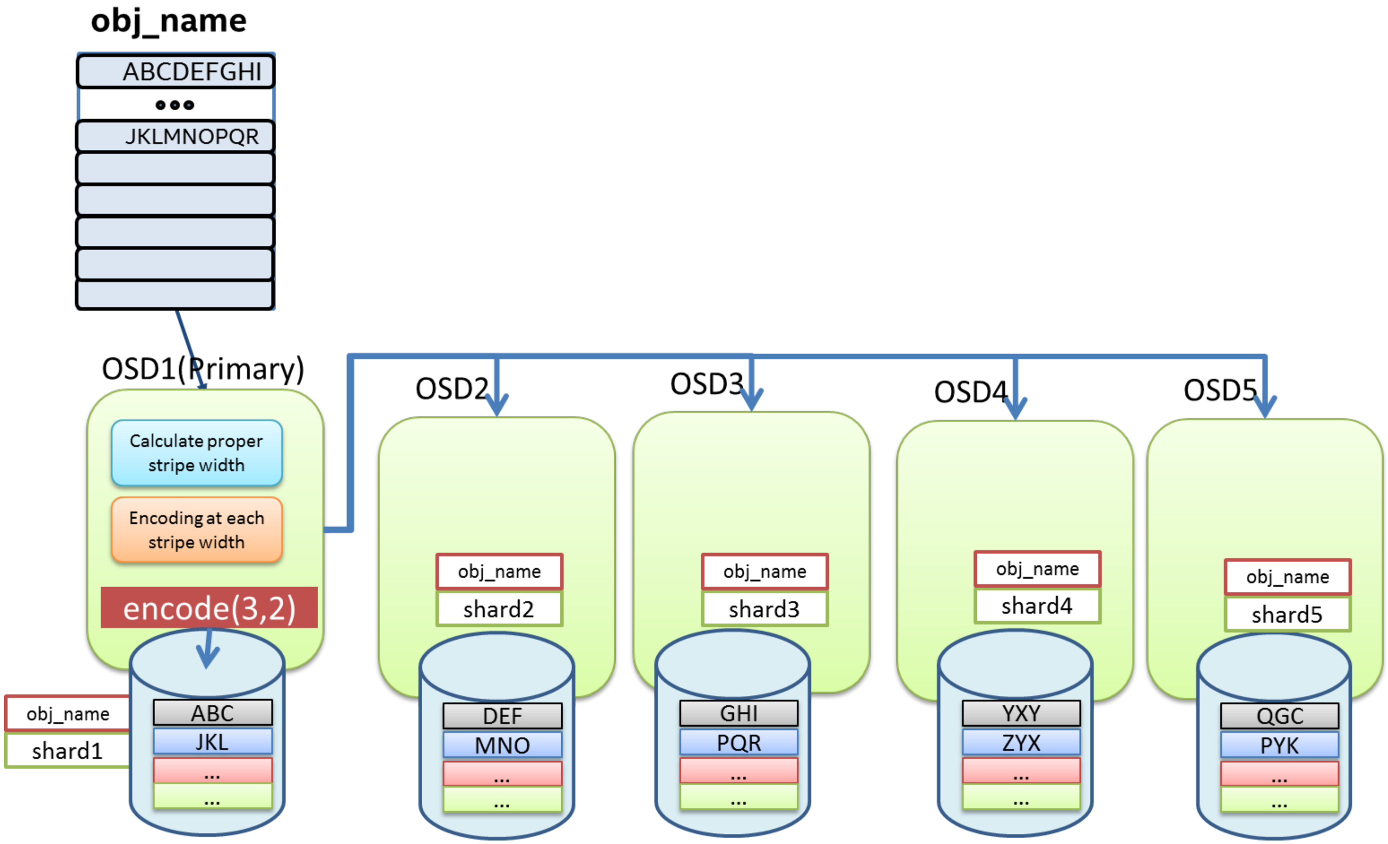
The general read/write flow Ceph is like:


1 Client app issues read request, RADOS sends request to primary OSD
2 Primary OSD reads data from local disk and completes read request
Read
Write
Compute Node
App
RBD
RADOS
1 Client app writes data, RADOS sends data to primary OSD
2 Primary OSD identifies replica OSDs and sends data, writes data to local disk
3 Replica OSDs write data to local disk, signal completion to primary
4 Primary OSD signals completion to client app
OSD
Disk
Server

With Erasure Code feature support, the read/write flow has changed to:

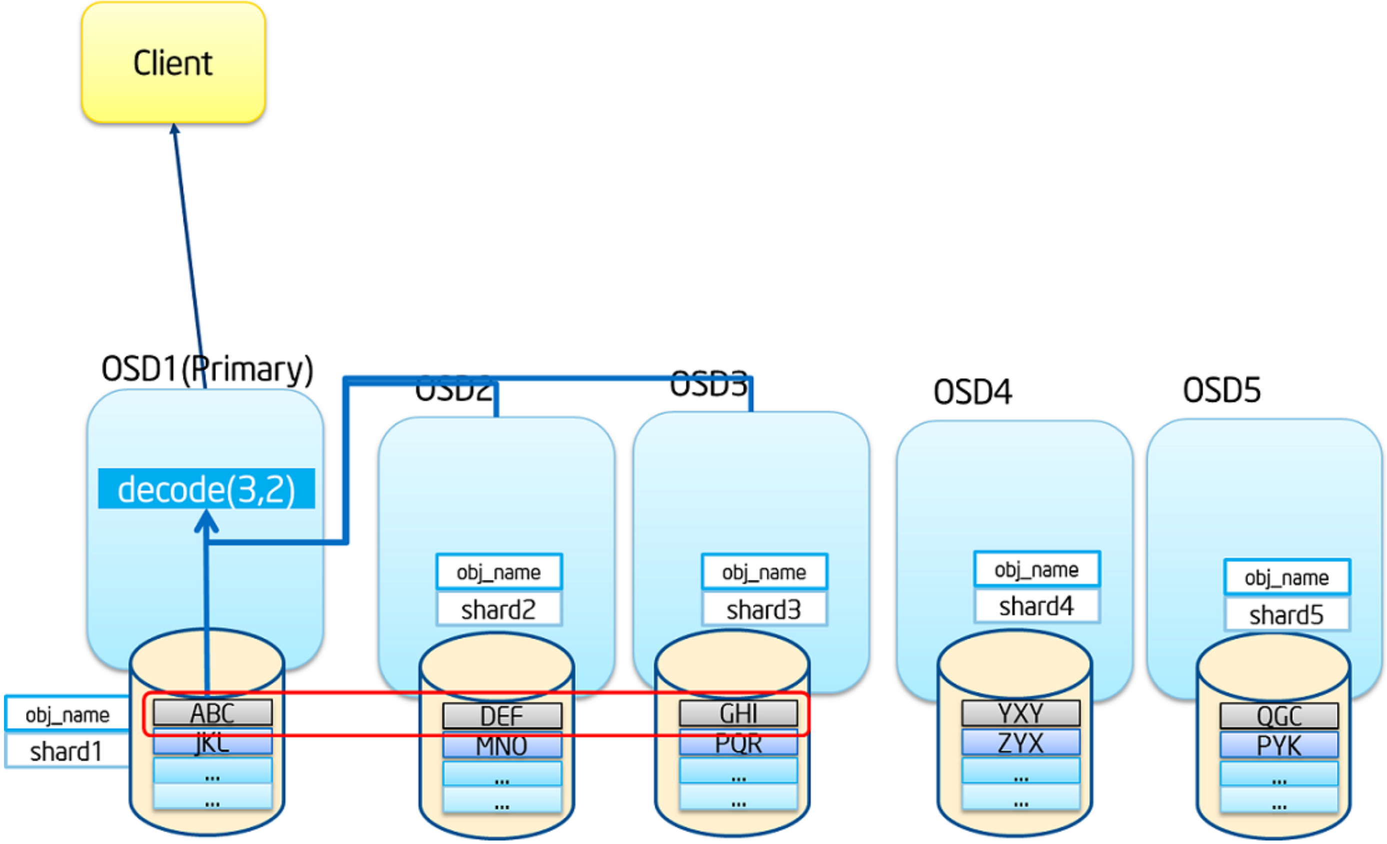
EC write:

Data will be encoded in the primary OSD and then spread to the corresponding OSDs

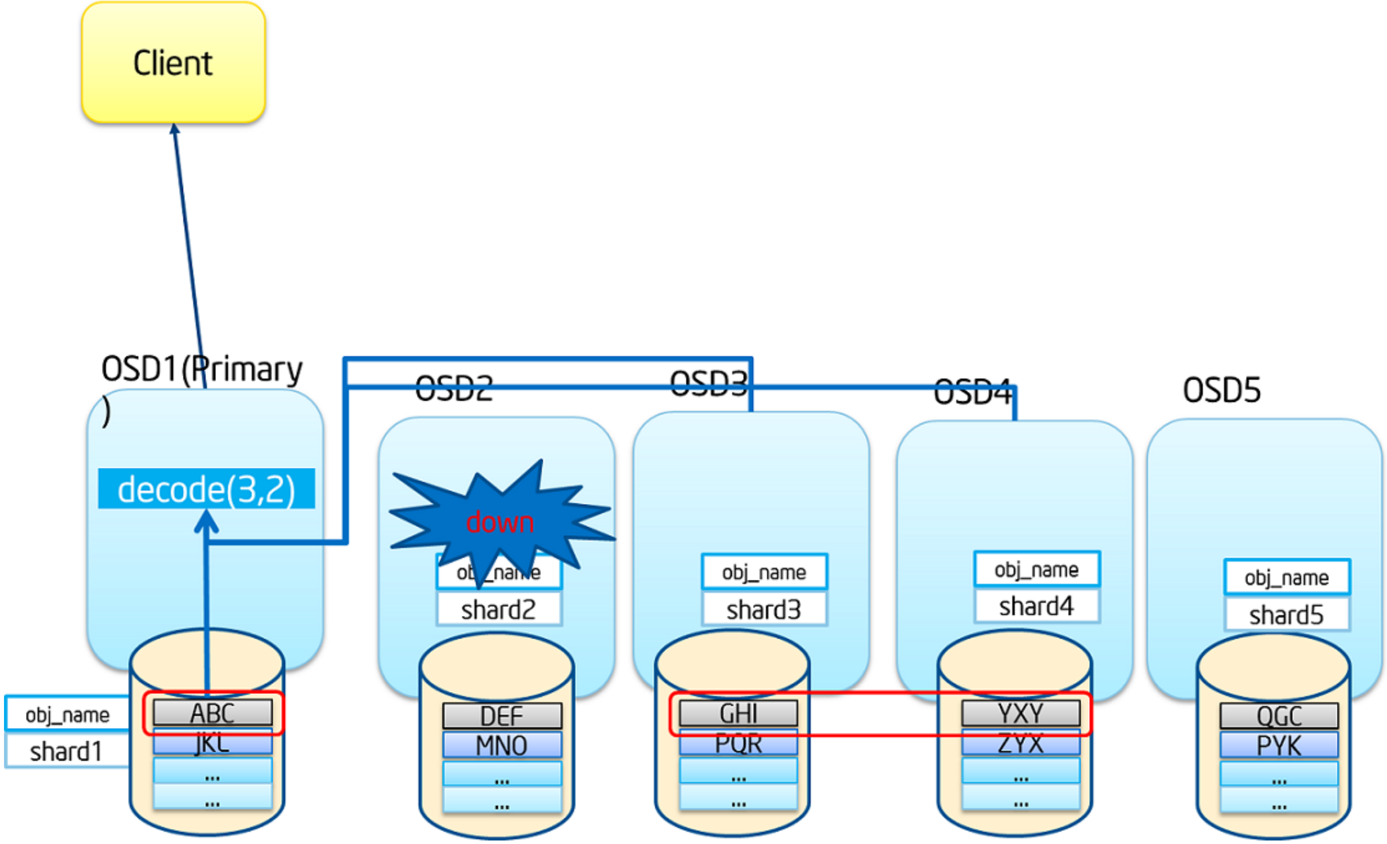


EC read:

Data will be gathered from the corresponding OSDs and then do the decode work.



If there's some data missing, Ceph will automatically read from the parity and then do the decode.



For now EC was recommended in object storage mode. For the filesystem and block storage mode, Ceph does not recommended to use EC since the performance would suffer a lot.

Currently there're several different EC plugins in Ceph: Jerasure, ISA-l and LRC.

Jerasure was an open source EC library developed by Prof. James Plank, it supports lots of EC technology now and the performance is good.

ISA-l was optimized for Intel platforms using some platform specific instructions. Currently it’s open source software.

LRC was mostly a different layer than Jerasure and ISA-l. Since it could use either Jerasure or ISA-l as the backend encoding/decoding library.

How to use Erasure Code feature in Ceph?

Ceph EC was set at pool level. All the EC parameters are defined when creating the pool. E.g.:

```
ceph osd pool create poolname test_pool \

   erasure-code-directory=<dir>         \ # mandatory

   erasure-code-plugin=jerasure         \ # mandatory


   erasure-code-m=1                    \ # optional and plugin dependant

   erasure-code-k=2                    \ # optional and plugin dependant

   erasure-code-technique=reed_sol_van  \ # optional and plugin dependan
```

All the objects stored in this test_pool will be ECed. However the clients are transparent to this.

Currently Ceph provides its own EC plugin management system, which makes adding more EC plugins quite easy in future. The interfaces are defined like:

```
set<int> minimum_to_decode(const set<int> &want_to_read, const set<int>

set<int> minimum_to_decode_with_cost(const set<int> &want_to_read, const

map<int, buffer> encode(const set<int> &want_to_encode, const buffer &in

map<int, buffer> decode(const set<int> &want_to_read, const map<int, buf
```

Ceph can load your own EC plguins cleanly once your EC plugin supports these interfaces.

To easily manage the Erasure Code parameters, Ceph provides an EC profile concept:

```
Ceph osd erasure-code-profile set {name} \

         [{k=data-chnks}] \

         [{m=coding-chunks}] \

         [{directory=directory}] \

         [{plugin=plugin}] \

         [{key=value}..] \

         [--force]
```

One could create one EC pool with the erasure-code-profile easily:

ceph osd pool create ecpool PG_NUM PGP_NUM erasure ecprofle

Reference:

Ceph project

- http://ceph.com/ (http://ceph.com/)

Inktank – Ceph professional support services

- http://www.inktank.com (http://www.inktank.com/)/ (http://www.inktank.com/)

Inktank – Ceph Reference Architecture

- http://www.inktank.com/resource/ceph-reference-architecture (http://www.inktank.com/resource/ceph-reference-architecture/)/ (http://www.inktank.com/resource/ceph-reference-architecture/)

Inktank Ceph Enterprise (ICE)

- http://www.inktank.com/enterprise/ (http://www.inktank.com/enterprise/)

Official Ceph Documentation

- http://ceph.com/docs/master/ (http://ceph.com/docs/master/)

Getting Started with Ceph

- http:// (http://www.slideshare.net/Inktank_Ceph/webinar-getting-started-with-ceph)www.slideshare.net/Inktank_Ceph/webinar-getting-started-with-ceph (http://www.slideshare.net/Inktank_Ceph/webinar-getting-started-with-ceph)

Ceph Advanced Features

- http://www.slideshare.net/Inktank_Ceph/webinar-advance-ceph-features (http://www.slideshare.net/Inktank_Ceph/webinar-advance-ceph-features)

Ceph Overview

- http://storageconference.org/2012/Presentations/M05.Weil.pdf (http://storageconference.org/2012/Presentations/M05.Weil.pdf)
- http://www.anchor.com.au/blog/2012/09/a-crash-course-in-ceph/ (http://www.anchor.com.au/blog/2012/09/a-crash-course-in-ceph/)

Ceph and OpenStack

- http://www.sebastien-han.fr/blog/2012/06/10/introducing-ceph-to-openstack (http://www.sebastien-han.fr/blog/2012/06/10/introducing-ceph-to-openstack/)/ (http://www.sebastien-han.fr/blog/2012/06/10/introducing-ceph-to-openstack/)

Attachment	Size
ceph_ec_05_1.png (https://software.intel.com/sites/default/files/managed/39/36/ceph_ec_05_1.png)	169.33 KB
ceph_ec_06.png (https://software.intel.com/sites/default/files/managed/39/36/ceph_ec_06.png)	188.88 KB

For more complete information about compiler optimizations, see our Optimization Notice (/en-us/articles/optimization-notice#opt-en).

- Hardware Developers
  - Firmware
  - Modeling & Simulation
  - Resource and Design Center
  - Shop Intel
- Open Source
  - 01.org
  - GitHub*
- Manage Your Tools
  - Download Center
  - Priority Support
  - Registration Center
- Connect
  - Forums
  - Meet the Experts
  - Newsletter
  - Recent Updates
  - YouTube* Channel