CEPH CLUSTER CREATION AND MANAGEMENT ON VAGRANT VIRTUAL MACHINES

A Degree Thesis
Submitted to the Faculty of the
Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona
Universitat Politècnica de Catalunya
by
Nabil el Alami Khalifi

In partial fulfilment
of the requirements for the degree in
TELEMATICS ENGINEERING

Advisor: Jose Luis Muñoz Tapia

Barcelona, January 2017

Abstract

This project aims to deploy a Ceph Cluster using Vagrant provisioned Virtual machines. The analysis of this Software Defined Storage system will help me out to understand the architecture and functionality of a cloud storage service as Microsft Azure.

Currently, I'm working as a Cloud and Infrastructure Consultant at TOKIOTA, SL, so this project would help get the background that I need to get the Microsoft Certifications for Azure and Cloud Storage.

Also, this project propose Ceph as a reliable storage system for an academical Cloud for our University. This cloud would managed by the faculty professors and would meant to be a project repository for projects.

Resum

El objectiu d'quest projecte és desenvolupar un Cluster basat en Ceph i utilitzant Vagrant com a provisionador de la màquines virtuals. L'anàlisis d'aquest sistema d'emmagatzematge basat en software em servirà d'ajuda per entendre l'arquitectura i les funcionalitats d'un servei d'emmagatzematge al núvol com és Micorsoft Azure.

Actualment, estic treballant com a consultor al departament de Cloud i infrastructura a TOKIOTA, SL, així que aquest projecte em pot ajudar a conseguir la teoria que necessito per poder conseguir les certificacions de Microsoft per a Azure i emmagatzematge als núvols.

Aquest projecte tmabé proposa Ceph com a un sistema d'emmagatzematge de confiança per a un Cloud acadèmic per a la nostra facultat. Aquest Cloud seria gestionat pels professors de la facultat i estaria pensat per fer-se servir com a repositori per als projectes dels alumnes.

Resumen

El objectivo de este proyecto es desenvolupar un Cluster basado en Ceph y utilizando Vagrant com provisionador de la máquines virtuales. El análisis de este sistema de almacenamiento basado en software me servirá de ayuda per entendre la arquitectura y las funcionalidades de un servicio de almacenamiento en la nube com es Micorsoft Azure.

Actualmente, estoy trabajando com consultor en el departamento de Cloud y infrastructura en TOKIOTA, SL, así que este proyecto me puede ayudar a conseguir la teoria que necessito para poder conseguir las certificaciones de Microsoft para Azure y almacenamiento en la nube.

este proyecto también Ceph como sistema de almacenamiento de confianza para un Cloud academico para nuestra facultad. Este Cloud sería gestionado por los professores de la facultad y estara pensado para hacerse servir com repositorio para los proyectos de los alumnos.

Acknowledgments

I want to thank my advisor, José Luis Muñoz, for introducing me to Software Defined Storage systems such Ceph is, and guiding me along this project.

Revision history and approval record

Revision	Date	Purpose
0	15/10/2016	Document creation
1	22/11/2016	Document revision
2	09/12/2016	Document revision
3	03/01/2017	Document revision

DOCUMENT DISTRIBUTION LIST

Name	e-mail
Nabil el Alami Khalifi	nabilnd92@gmail.com
José Luis Muñoz Tapia	jose.munoz@entel.upc.edu

Written by:		Reviewed by:	
Date	12/01/2017	Date	13/01/2017
Name	Nabil el Alami Khalifi	Name	José Luis Muñoz Tapia
Position	Project author	Position	Project Supervisor

Table of contents

List of Figures

List of Tables

1 Introduction

The main goal of this document is to provide a summary of the actual project. All the extended documentation of the researches and technologies used in this project are contained in the annex of this memory, including the cluster nodes configuration.

1.1 Statement of purpose (objectives)

The purpose of this project is to analyze and deploy a Ceph storage cluster in order to review the structure and behavior of a Software Defined Storage (SDS) system. Vagrant is used to create the machines and make the initial files and scripts provisioning.

During the realization of this project, we chose to focus our study on the Client node and how the communication with the cluster is managed when data is formatted and stored. Ceph has practically an endless fields of study, so we set the key bases to understand its capabilities and architecture.

Ceph technical terms, architecture, protocol and the scenario deployment are documented and analyzed in the annex, in order to provided an extended information and discussions.

This study discuses the suitability of Ceph as a distributed storage system capable of managing an academical cloud for our university due to its scalability and high performance. Ceph avoid a single point of failure using Cephx protocol (explaind below), so it makes it more reliable than traditional sotrage systems.

1.2 Requirements and specifications

The requirements of this project can be organized in two different categories: technical requirements and conceptual requirements. Conceptual requirements are focused on the basics of Storage Technology meanwhile technical requirements are focused on programming skills and implementations.

Conceptual requirements:

1. Storage Devices.
2. Filesystem.

3. Block Devices.
4. RAID
5. Disck Attached Storage.
6. NetWork Attached sotrage.
7. Storage Area Networks.
8. Software Defined Storage.

Technical requirements:

1. Use of Vagrant.
2. Use of Ceph deployment tool.
3. Bash scripting basics.
4. Ruby scripting basics.
5. Use of the software version control system SVN.
6. Use of the LaTeX editor to document.
7. Use of Linux kernel-based OS.

1.3 Third party resources

This project uses the *Storage Book* theory and background in order to get storage technologies and systems conceptual terms.

1.4 Work Plan

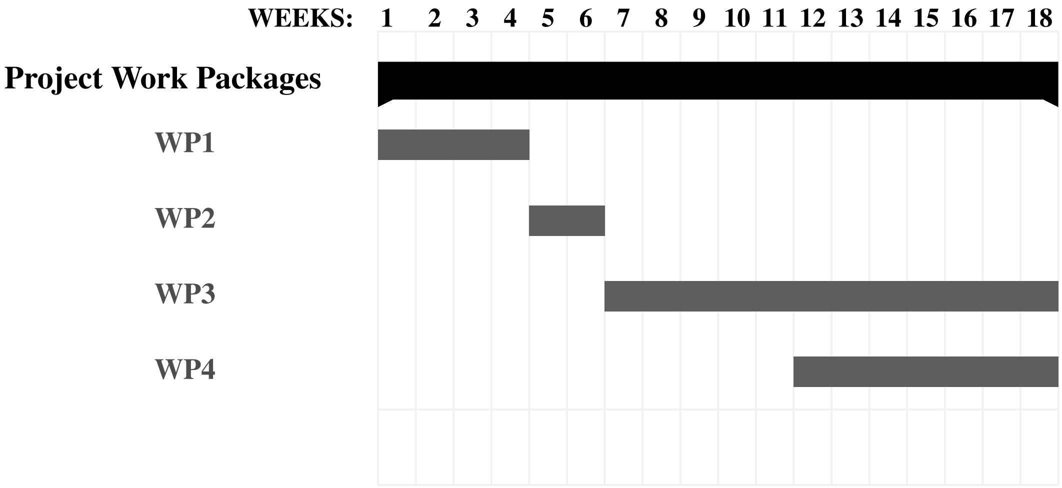
Work packages:

Project: storage book lecture and creation of project repository	WP ref: WP1
Major constituent: theoretical background	Sheet 1 of 4
Short description: read the storage book wrote by the advisorand creation of the project repository	Planned start date:15/09/2016 Planned end date:25/10/2016
	Start event:15/10/2016 End event:25/10/2016

Project: install new ubuntu distro and vagrant tool	WP ref: WP2
Major constituent: ubuntu upgrate to 16.04	Sheet 3 of 4
Short description: ubuntu upgrate to 16.04 and installation of the vagrant tool	Planned start date:26/10/2016 Planned end date:29/10/2016
	Start event:26/10/2016 End event:29/10/2016

Project: Ceph cluster deployment and testing	WP ref: WP3
Major constituent: Cluster deployment	Sheet 3 of 4
Short description: Create the cluster nodes and configure them to form a cluster. Test the storage capbilities.	Planned start date: 26/10/2016 Planned end date: 30/12/2016
	Start event: 26/10/2016 End event: 30/12/2016

Project: project documentation	WP ref: WP4
Major constituent: documentation	Sheet 4 of 4
Short description: Theory and cluster configuration documentatio	Planned start date: 26/10/2016 Planned end date: 30/12/2016
	Start event:26/10/2016 End event: 30/12/2016



1.5 Deviations from the initial plan

The original project was going to use LXC Containers for the cluster nodes.

2 State of the art of the technology used or applied in this thesis:

2.1 CEPH

Ceph is an open source distributed storage system with high-performance and scalability, built on top of commodity components, demanding reliability to the software layer. We can say that Ceph is a scale out Software Defined Object Storage built on commodity hardware. Ceph storage software it's one of the pillars of the OpenStack project, which involved developers around the world working on different components of open source cloud platforms.

Ceph was developed to make possible the standardized storage of Object, Block and Files with a distributed x86 cluster. Ceph has the ability to manage vast amounts of data and delivers extraordinary scalability–thousands of clients accessing petabytes to exabytes of data. A Ceph Node leverages commodity hardware and intelligent daemons, and a Ceph Storage Cluster accommodates large numbers of nodes, which communicate with each other to replicate and redistribute data dynamically.

2.2 Ceph technical terms

2.2.1 RADOS

Ceph Storage Cluster is based on an open source object storage service called Reliable Automatic Distributed Object Store (RADOS).

RADOS has the ability to scale to thousands of hardware devices by making use of management software that runs on each of the individual storage nodes. The software provides storage features such as thin provisioning, snapshots and replication. RADOS distributes objects across the storage cluster and replicates objects for fault tolerance.

2.2.2 CRUSH

The Controlled Replication Under Scalable Hashing algorithm determines how the data is replicated and mapped to the individual nodes. CRUSH ensures that data is evenly distributed across the cluster and that all cluster nodes are able to retrieve data quickly without any centralized bottlenecks.

The algorithm determines how to store and retrieve data by computing data storage locations. CRUSH empowers Ceph clients to communicate with OSDs (explained below) directly rather than through a centralized server or broker. With an algorithmically determined method of storing and retrieving data, Ceph avoids a single point of failure, a performance bottleneck, and a physical limit to its scalability.

CRUSH requires a map of our cluster (contain a list of OSDs) , and uses the CRUSH map to pseudo-randomly store and retrieve data in OSDs with a uniform distribution of data across the cluster.

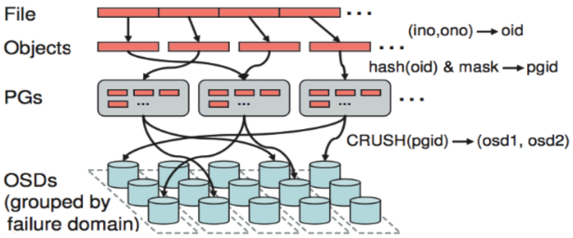


Figure 1: Crush Algorithm.

2.2.3 Ceph Nodes

Ceph OSDs

A Ceph OSD Daemon stores data as objects on a storage node, handles data replication, recovery, backfilling, rebalancing, and provides some monitoring information to Ceph Monitors by checking other Ceph OSD Daemons for a heartbeat. So, as Ceph OSDs run the RADOS service, calculate data placement with CRUSH and maintain their own copy of the cluster map they should have a reasonable amount of processing power. A Ceph Storage Cluster requires at least two Ceph OSD Daemons to achieve an active + clean state when the cluster makes two copies of our data (by default).

Ceph Monitor

Maintains a master copy of the cluster map (cluster state), including the monitor map, the OSD map, the PG map, and the CRUSH map. Ceph maintains a history (called an “epoch”) of each state change in the Ceph Monitors, Ceph OSD Daemons and PGs. They are not CPU intensive.

Ceph Metadata Server

A MDS stores metadata on behalf of the Ceph Filesystem (i.e., Ceph Block Devices and Ceph Object Storage do not use MDS). Ceph Metadata Servers make it feasible for Portable Operating System Interface (POSIX) file system users to execute basic commands like ls, find, etc. without placing an enormous burden on the Ceph Storage Cluster.

2.2.4 Placement Group (PG)

Aggregates objects within a pool (logical partitions for storing objects) because tracking object placement and object metadata on a per-object basis is computationally expensive.

2.2.5 Ceph Cluster Map

The set of maps comprising the monitor map, OSD map, PG map, MDS map and CRUSH map.

2.3 Ceph Architecture

Ceph is made for big company’s IT infrastructures as is able to manage vast amounts of data and deliver extraordinary scalability–thousands of clients accessing petabytes to exabytes of data. But not only these, Ceph also (and uniquely) delivers object, block and file storage in one unified System, being highly reliable, easy to manage and free.

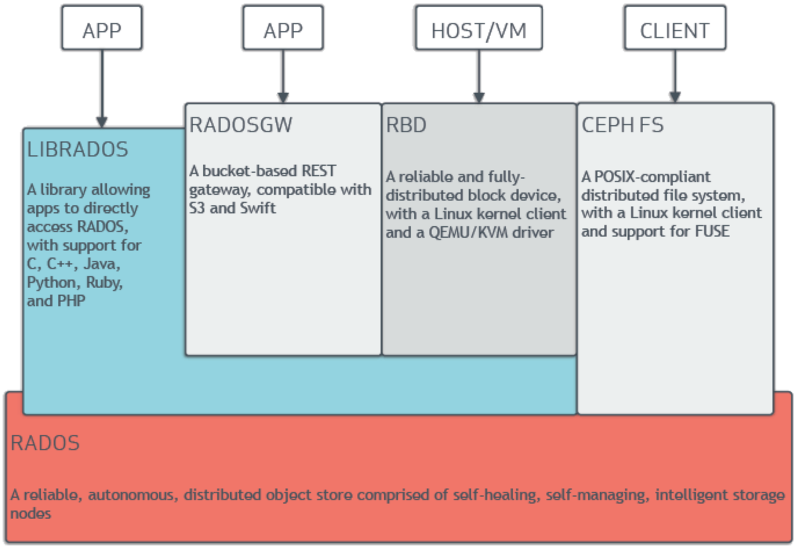

APP
APP
HOST/VM
CLIENT
LIBRADOS
A library allowing apps to directly access RADOS, with support for C, C++, Java, Python, Ruby, and PHP
RADOSGW
A bucket-based REST gateway, compatible with S3 and Swift
RBD
A reliable and fully-distributed block device, with a Linux kernel client and a QEMU/KVM driver
CEPH FS
A POSIX-compliant distributed file system, with a Linux kernel client and support for FUSE
RADOS
A reliable, autonomous, distributed object store comprised of self-healing, self-managing, intelligent storage nodes

Figure 2: Ceph Architecture.

But behind the flagship products we have many components that are important in the functioning of Ceph. Like the Ceph Nodes, who leverages commodity hardware and intelligent daemons, and the Ceph Storage Cluster, that accommodates large numbers of nodes, which communicate with each other to replicate and redistribute data dynamically.

In these section we will see some of these components and we will explain the role that they have throughout the Ceph system.

2.3.1 Ceph storage working mode

As we explained earlier, Ceph provides an infinitely scalable Ceph Storage Cluster based upon RADOS, and consists of two types of daemons: **MONs** and **OSDs**.

Storage cluster clients and each Ceph OSD Daemon use the **CRUSH** algorithm to efficiently compute information about data location, instead of having to depend on a central lookup table. Ceph's high-level features include providing a native interface to the Ceph Storage Cluster via **librados**, and a number of service interfaces built on top of librados.

Storing Data

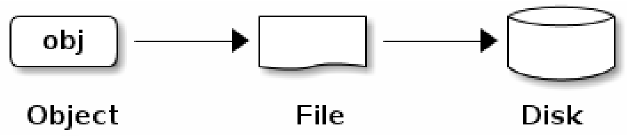


Figure 3: Ceph storing data.

The Ceph Storage Cluster receives data from **Ceph Clients**–whether it comes through a Ceph Block Device, Ceph Object Storage, the Ceph Filesystem or a custom implementation we create using librados–and it stores the data as objects. Each object corresponds to a file in a filesystem, which is stored on an Object Storage Device. Ceph OSD Daemons handle the read/write operations on the storage disks.

Ceph OSD Daemons store all data as objects in a flat namespace (e.g., no hierarchy of directories). An object has an identifier (ID), which is unique across the entire cluster, not just the local filesystem,a binary data, and metadata consisting of a set of name/value pairs. The semantics are completely up to Ceph Clients.

Scalability and High Availability

Ceph eliminates the centralized gateway used in traditional architectures (e.g., a gateway), and enable clients to interact with Ceph OSD Daemons directly, increasing both performance and scalability. Ceph OSD Daemons create object replicas on other Ceph Nodes to ensure data safety and high availability. Ceph also uses a cluster of monitors to ensure high availability. To eliminate centralization, Ceph uses an algorithm called **CRUSH**.

High Availability Monitors

Before Ceph Clients can read or write data, they must contact a **Ceph Monitor** to obtain the most recent copy of the cluster map. A Ceph Storage Cluster can operate with a single monitor; however, this introduces a single point of failure.For added reliability and fault tolerance, Ceph supports a cluster of monitors.

In a cluster of monitors, latency and other faults can cause one or more monitors to fall behind the current state of the cluster. For this reason, Ceph must have agreement among various monitor instances regarding the state of the cluster. Ceph always uses a majority of monitors (e.g., 1, 2:3, 3:5, 4:6, etc.) and the PAXOS algorithm to establish a consensus among the monitors about the current state of the cluster.

To identify users and protect against man-in-the-middle attacks, Ceph provides its **cephx** authentication system to authenticate users and daemons.

Cephx uses shared secret keys for authentication, meaning both the client and the monitor cluster have a copy of the client's secret key. The authentication protocol is such that both parties are able to prove to each other they have a copy of the key without actually revealing it. This provides mutual authentication, which means the cluster is sure the user possesses the secret key, and the user is sure that the cluster has a copy of the secret key.

To use **cephx**, an administrator must set up users first. In the following diagram, the **client.admin** user invokes **ceph auth get-or-create-key** from the command line to generate a username and secret key. Ceph's auth subsystem generates the username and key, stores a copy with the monitor(s) and transmits the user's secret back to the **client.admin** user. This means that the client and the monitor share a secret key.

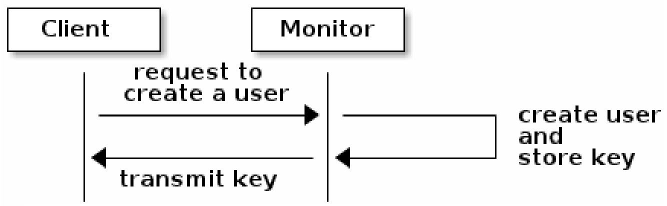


Figure 4: Cephx function.

To authenticate with the monitor, the client passes in the user name to the monitor, and the monitor generates a session key and encrypts it with the secret key associated to the user name. Then, the monitor transmits the encrypted ticket back to the client. The client then decrypts the payload with the shared secret key to retrieve the session key. The session key identifies the user for the current session. The client then requests a ticket on behalf of the user signed by the session key. The monitor generates a ticket, encrypts it with the user's secret key and transmits it back to the client. The client decrypts the ticket and uses it to sign requests to OSDs and metadata servers throughout the cluster.

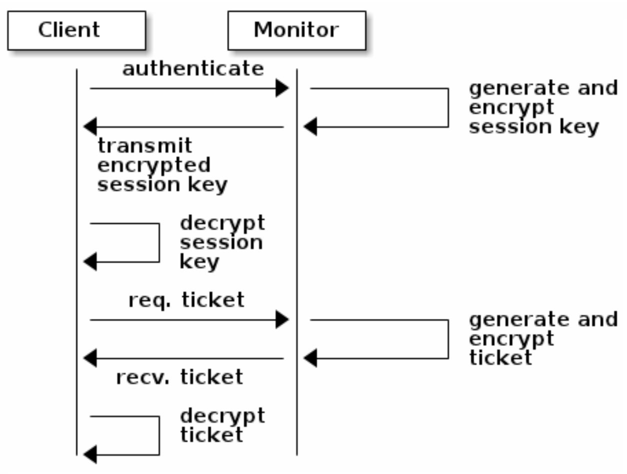

Client
Monitor
authenticate
generate and encrypt session key
transmit encrypted session key
decrypt session key
req. ticket
generate and encrypt ticket
recv. ticket
decrypt ticket

Figure 5: Cephx encrypted ticket.

The **cephx** protocol authenticates ongoing communications between the client machine and the Ceph servers. Each message sent between a client and server, subsequent to the initial authentication, is signed using a ticket that the monitors, OSDs and metadata servers can verify with their shared secret.

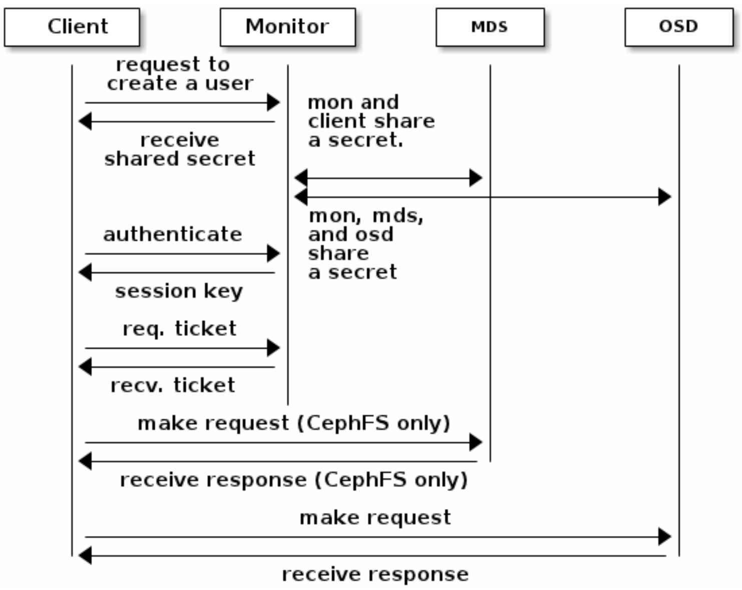


Figure 6: Cephx authentication.

The protection offered by this authentication is between the Ceph client and the Ceph server hosts. The authentication is not extended beyond the Ceph client. If the user accesses the Ceph client from a remote host, Ceph authentication is not applied to the connection between the user's host and the client host.

Smart Daemons Enable Hyperscale

In many clustered architectures, the primary purpose of cluster membership is so that a centralized interface knows which nodes it can access. Then the centralized interface provides services to the client through a double dispatch–which is a huge bottleneck at the petabyte-to-exabyte scale.

Ceph eliminates the bottleneck: Ceph's OSD Daemons AND Ceph Clients are cluster aware. Like Ceph clients, each Ceph OSD Daemon knows about other Ceph OSD Daemons in the cluster. This enables Ceph OSD Daemons to interact directly with other Ceph OSD Daemons and Ceph monitors. Additionally, it enables Ceph Clients to interact directly with Ceph OSD Daemons.

The ability of Ceph Clients, Ceph Monitors and Ceph OSD Daemons to interact with each other means that Ceph OSD Daemons can utilize the CPU and RAM of the Ceph nodes to easily perform tasks that would bog dawn a centralized server. The ability to leverage this computing power leads to several major benefits like that the OSDs can service Clients directly, Data Scrubbing or Replication.

So, as we can see in the following figure, the client writes the object to the identified placement group in the primary OSD. Then, the primary OSD with its own copy of the CRUSH map identifies the secondary and tertiary OSDs for replication purposes, and replicates the object to the appropriate placement groups in the secondary and tertiary OSDs (as many OSDs as additional replicas), and responds to the client once it has confirmed the object was stored successfully.

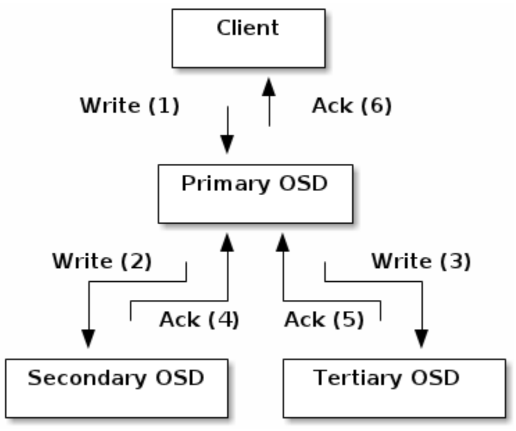


Figure 7: Client and daemons communication.

With the ability to perform data replication, Ceph OSD Daemons relieve Ceph clients from that duty, while ensuring high data availability and data safety.

Dynamic Cluster Management

In previous **Scalability and High Availability** section, we explained how Ceph uses CRUSH, cluster awareness and intelligent daemons to scale and maintain high availability. Key to Ceph's design is the autonomous, self-healing, and intelligent Ceph OSD Daemon. Next we will take a deeper look at how CRUSH works to enable modern cloud storage infrastructures to place data, rebalance the cluster and recover from faults dynamically.

Pools

The Ceph storage system supports the notion of 'Pools', which are logical partitions for storing objects.

Ceph Clients retrieve a Cluster Map from a Ceph Monitor, and write objects to pools. The pool's size or number of replicas, the CRUSH ruleset and the number of placement groups determine how Ceph will place the data.

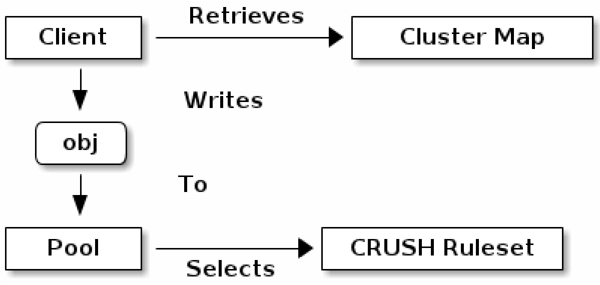


Figure 8: Client to pool.

Pools set at least the three parameters: Ownership/Access to Objects, Number of Placement Groups, and the CRUSH Ruleset to Use.

Mapping PGs to OSDs

Each pool has a number of placement groups. CRUSH maps PGs to OSDs dynamically. When a Ceph Client stores objects, CRUSH will map each object to a placement group.

Mapping objects to placement groups creates a layer of indirection between the Ceph OSD Daemon and the Ceph Client. The Ceph Storage Cluster must be able to grow (or shrink) and rebalance where it stores objects dynamically. If the Ceph Client “knew” which Ceph OSD Daemon had which object, that would create a tight coupling between the Ceph Client and the Ceph OSD Daemon. Instead, the CRUSH algorithm maps each object to a placement group and then maps each placement group to one or more Ceph OSD Daemons. This layer of indirection allows Ceph to rebalance dynamically when new Ceph OSD Daemons and the underlying OSD devices come online. The following diagram explains how CRUSH maps objects to placement groups, and placement groups to OSDs.

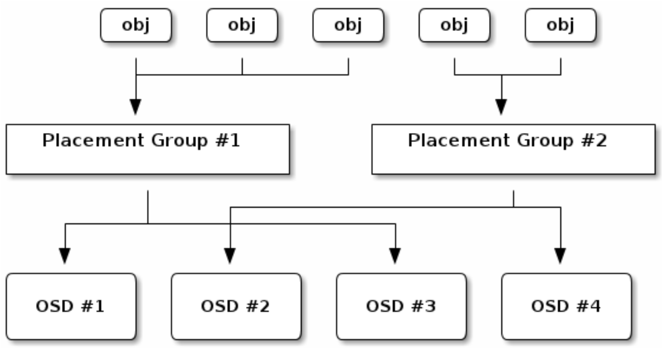


Figure 9: Crush to PG and PG to OSDs.

With a copy of the cluster map and the CRUSH algorithm, the client can compute exactly which OSD to use when reading or writing a particular object.

Rebalancing

When we add a Ceph OSD Daemon to a Ceph Storage Cluster, the cluster map gets updated with the new OSD. This changes the cluster map. Consequently, it changes object placement, because it changes an input for the calculations. The following diagram shows the rebalancing process (albeit rather crudely, since it is substantially less impactful with large clusters) where some, but not all of the PGs migrate from existing OSDs (OSD 1, and OSD 2) to the new OSD (OSD 3). Even when rebalancing, CRUSH is stable. Many of the placement groups remain in their original configuration, and each OSD gets some added capacity, so there are no load spikes on the new OSD after rebalancing is complete.

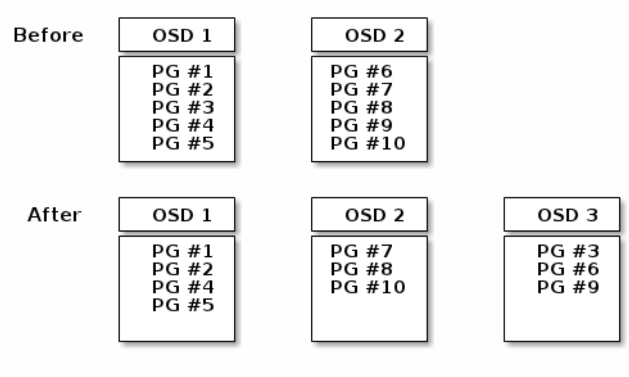


Figure 10: Rebalancing process.

Data Consistency

As part of maintaining data consistency and cleanliness, Ceph OSDs can also scrub objects within placement groups. That is, Ceph OSDs can compare object metadata in one placement group with its replicas in placement groups stored in other OSDs. Scrubbing (usually performed daily) catches OSD bugs or filesystem errors. OSDs can also perform deeper scrubbing by comparing data in objects bit-for-bit. Deep scrubbing (usually performed weekly) finds bad sectors on a disk that weren't apparent in a light scrub.

2.3.1.1 Ceph protocol

Ceph Clients use the native protocol for interacting with the Ceph Storage Cluster. Ceph packages this functionality into the librados library so that we can create our own custom Ceph Clients. The following diagram depicts the basic architecture.

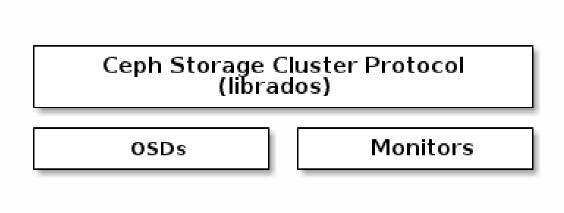


Figure 11: Ceph protocol.

Native Protocol and librados

Modern applications need a simple object storage interface with asynchronous communication capability. The Ceph Storage Cluster provides a simple object storage interface with asynchronous communication capability. The interface provides direct, parallel access to objects throughout the cluster.

2.3.1.2 Ceph Clients

Ceph Clients include a number of service interfaces. These include:

- **Block Devices**: The Ceph Block Device (RBD) service provides resizable, thin-provisioned block devices with snapshotting and cloning. Ceph stripes a block device across the cluster for high performance. Ceph supports both kernel objects (KO) and a QEMU hypervisor that uses **librbd** directly–avoiding the kernel object overhead for virtualized systems.

- **Object Storage**: The Ceph Object Storage (a.k.a., RGW) service provides RESTful APIs with interfaces that are compatible with Amazon S3 and OpenStack Swift.
- **Filesystem**: The Ceph Filesystem (CephFS) service provides a POSIX compliant filesystem usable with **mount** or as a filesytem in user space (FUSE).

Ceph can run additional instances of OSDs, MDSs, and monitors for scalability and high availability. The following diagram depicts the high-level architecture.

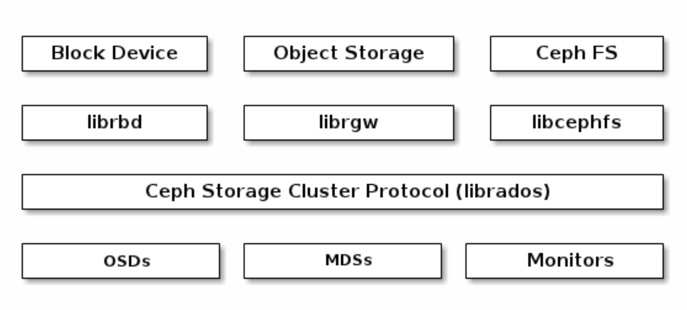


Figure 12: Ceph Clients.

CEPH OBJECT STORAGE

As we explained in **Ceph Storage components** section, the Ceph Object Storage daemon, **radosgw**, is a FastCGI service that provides a RESTful HTTP API to store objects and metadata. It layers on top of the Ceph Storage Cluster with its own data formats, and maintains its own user database, authentication, and access control. The RADOS Gateway uses a unified namespace, which means we can use either the OpenStack Swift-compatible API or the Amazon S3-compatible API. For example, we can write data using the S3-compatible API with one application and then read data using the Swift-compatible API with another application.

CEPH BLOCK DEVICE STORAGE

A Ceph Block Device stripes a block device image over multiple objects in the Ceph Storage Cluster, where each object gets mapped to a placement group and distributed, and the placement groups are spread across separate **ceph-osd** daemons throughout the cluster. It's important to say that striping allows RBD block devices to perform better than a single server could.

Thin-provisioned snapshottable Ceph Block Devices are an attractive option for virtualization and cloud computing. In virtual machine scenarios, people typically deploy a Ceph Block Device with the **rbd** network storage driver in Qemu/KVM, where the host machine uses **librbd** to provide a block device service to the guest. Many cloud computing stacks use **libvirt** to integrate with hypervisors. We can use thin-provisioned Ceph Block Devices with Qemu and **libvirt** to support OpenStack and CloudStack among other solutions.

While Ceph do not provide **librbd** support with other hypervisors at this time, we may also use Ceph Block Device kernel objects to provide a block device to a client. Other virtualization technologies such as Xen can access the Ceph Block Device kernel object(s). This is done with the command-line tool **rbd**.

CEPH FILESYSTEM

The Ceph Filesystem (Ceph FS) provides a POSIX-compliant filesystem as a service that is layered on top of the object-based Ceph Storage Cluster. Ceph FS files get mapped to objects that Ceph stores in the Ceph Storage Cluster. Ceph Clients mount a CephFS filesystem as a kernel object or as a Filesystem in User Space (FUSE).

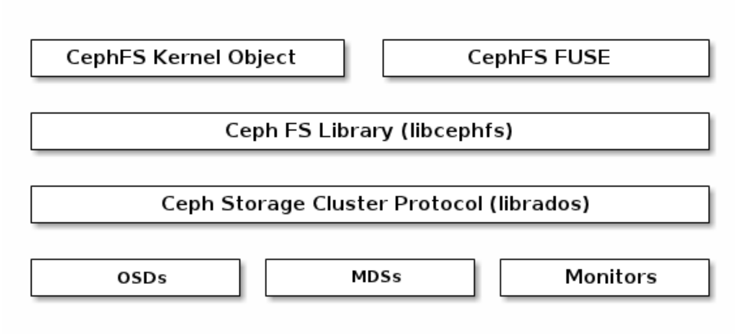

CephFS Kernel Object
CephFS FUSE
Ceph FS Library (libcephfs)
Ceph Storage Cluster Protocol (librados)
OSDs
MDSs
Monitors

Figure 13: Ceph Filesystem.

The Ceph Filesystem service includes the Ceph Metadata Server (MDS) deployed with the Ceph Storage cluster. The purpose of the MDS is to store all the filesystem metadata (directories, file ownership, access modes, etc) in high-availability Ceph Metadata Servers where the metadata resides in memory. The reason for the MDS (a daemon called **ceph-mds**) is that simple filesystem operations like listing a directory or changing a directory (**ls, cd**) would tax the Ceph OSD Daemons unnecessarily. So separating the metadata from the data means that the Ceph Filesystem can provide high performance services without taxing the Ceph Storage Cluster.

Ceph FS separates the metadata from the data, storing the metadata in the MDS, and storing the file data in one or more objects in the Ceph Storage Cluster. The Ceph filesystem aims for POSIX compatibility. **ceph-mds** can run as a single process, or it can be distributed out to multiple physical machines, either for high availability or for scalability:

- **High Availability**: The extra **ceph-mds** instances can be standby, ready to take over the duties of any failed **ceph-mds** that was active. This is easy because all the data, including the journal, is stored on RADOS. The transition is triggered automatically by **ceph-mon**.
- **Scalability**: Multiple **ceph-mds** instances can be active, and they will split the directory tree into subtrees (and shards of a single busy directory), effectively balancing the load amongst all active servers.

2.4 Paxos

Paxos is a family of protocols for solving consensus in a network of unreliable processors. Consensus is the process of agreeing on one result among a group of participants. This problem becomes difficult when the participants or their communication medium may experience failures.Leslie Lamport came in, with his Paxos protocol which was discovered and made famous in the 1990s.

2.4.1 Roles

One node acts as a proposer, and is responsible for initiating the protocol. Only one node can act as proposer at a time, but if two or more choose to then the protocol will (typically) fail to terminate until only one node continues to act as proposer. Again, this is sacrificing termination for correctness.

The other nodes which conspire to make a decision about the value being proposed are called, in Paxos terminology, ‘acceptors’. Acceptors respond to proposals from the proposer either by rejecting them for some reason, or agreeing to them in principle and making promises in return about the proposals they will accept in the future. These promises guarantee that proposals that may come from other proposers will not be erroneously accepted, and in particular they ensure that only the latest of the proposals sent by the proposer is accepted.

’Accept’ here means that an acceptor commits to a proposal as the one it considers definitive. Once a majority of acceptors have accepted the same proposal, the Paxos protocol can terminate and the proposed value may be disseminated to nodes which are interested in it (these are called ‘listeners’).

PROPOSERS:

1. Submit a proposal numbered n to a majority of acceptors. Wait for a majority of acceptors to reply.

2. If the majority reply ‘agree’, they will also send back the value of any proposals they have already accepted. Pick one of these values, and send a ‘commit’ message with the proposal number and the value. If no values have already been accepted, use your own. If instead a majority reply ‘reject’, or fail to reply, abandon the proposal and start again.

3. If a majority reply to your commit request with an ‘accepted’ message, consider the protocol terminated. Otherwise, abandon the proposal and start again.

ACCEPTORS:

1. Once a proposal is received, compare its number to the highest numbered proposal you have already agreed to. If the new proposal is higher, reply ‘agree’ with the value of any proposals you have already accepted. If it is lower, reply ‘reject’, along with the sequence number of the highest proposal.

2. When a ‘commit’ message is received, accept it if a) the value is the same as any previously accepted proposal and b) its sequence number is the highest proposal number you have agreed to. Otherwise, reject it.

3 Methodology / project development:

The analysis and development of the applications discussed in this thesis could not be possible without the following list of tools:

- **SVN**: apache Subversion is an open-source version control tool which we use for shared documents.
- **Vagrant**. an automation tool with a domain-specific language (DSL) that is used to automate the creation of VMs and VM environments. Vagrant allows as to create the cluster nodes and define provisioning plans for them, in order to configure an initial state.
- **Ceph-deploy tools**: with this tool we can get Ceph up and running quickly with sensible initial configuration settings relying only upon SSH access to the servers, sudo, and some Python. With ceph-deploy we can develop scripts to install Ceph packages on remote hosts, create a cluster, add monitors, gather (or forget) keys, add OSDs and metadata servers, configure admin hosts, and tear down the clusters. Ceph deploy runs in our workstation, and does not require servers, databases, or any other tools.

The methodology used for Ceph system analysis could be summarized in the following steps:

1. Vagrant file creation defining the cluster nodes parameters: Operative System, hostnames, network IPs, provisioning files and scripts.
2. Initial cluster configuration: admin node, monitor node, 2 OSDs.
3. More Monitors, OSDs and client addition to the cluster using ceph-deploy tool.
4. Storage pools and block devices creation using RADOS service.

total number of The annex contains the cluster configuration in detail with all the commands run during the deployment.

4 Results

As a result of this project, the storage book chapter had been added, including documentation about the main storage system architecture and protocols.

The most relevant result of this project are the following ones:

- The scenario offers the possibility to interact with a fully operable cluster and different client settings, that helps to understand how those storage techniques had evolve.

- The analysis of the CRUSH algorithm for OSD Object location from the client point of view, concludes that when a OSD is set out of the cluster, the re-balancing process takes immediately and implies that the amount of that moved among the cluster is $\frac{1}{N}$, where N is the total number of OSDs at the cluster. That because OSDs are constantly monitoring each other working at a peer to peer fashion, and when they notice a OSD is set out of the cluster, they get a new cluster map from the monitors. They use this new cluster map to calculate where the data belongs to and then, they do the re-balancing peer to peer, from one OSD to another.

- The analysis of the Paxos algorithm used for mange the monitors state, ensure the reliability and efficiency of the cluster monitoring. Paxos ensures that decisions made all the time must remain consensual, in order to report the right cluster map to all the cluster nodes. Also, the odd number of monitors and the Cephx protocol, avoids a single point of failure. Cephx protocol increases the overall data input and output, by letting the clients and the OSD interact with each other directly without an intermediator.

5 Budget

All the software used during this project is Open source, so did't add any cost to this project.

the operating system used for the scenario development is Ubuntu, which is a free GNU/Linux distribution, so it also does not add any cost to the thesis.

The estimated number of hours dedicated to this thesis is 380h. Taking in account the real cost of a recently graduated engineer, which is 12.5 €/h. The total cost of the thesis is 4750€.

6 Conclusions and future development:

6.1 Conclusions

It has been a very interesting project where a deep analysis of the Ceph distributed sotrage system helped my out to get a better understanding of the Software Defined Storage techniquesand architectures.

Ceph is a Open Source software and community based, so it's philosophy ensures a long-term support and the possibility to add more future specifications that now are under development, like CephFS and brtfs FileSystem improvementt.

It seems clear that the Ceph's high performance and scalability makes it suitable to run a Cloud service for educational environments, where can provide repositories for project developments and backup large files.

6.2 Future development

Ceph has a lot of features to study, like radosGW that provides a REST API for applications. Also, this project set the bases to investigate th CephFS when the development of it will be launch.

The scenario configuration of this project could be deployed using a different Virtual machines provider or LXC Containers and do a comparison of the performance of these.

Another proposed study field of study would be deploying a similar scenario using GlusterFs, which is a scale-out network-attached storage file system for cloud computing.

7 Bibliography:

1. Ceph docs: quick deploy
   http://docs.ceph.com/docs/master/start/quick-ceph-deploy/
2. Ceph docs: filesystem-recommendations
   http://docs.ceph.com/docs/jewel/rados/configuration/filesystem-recommendations/
3. Ceph docs: hardware-recommendations
   http://docs.ceph.com/docs/jewel/start/hardware-recommendations/
4. Ceph docs: quick-rbd
   http://docs.ceph.com/docs/jewel/start/quick-rbd/
5. Ceph docs: user-management
   http://docs.ceph.com/docs/giant/rados/operations/user-management/
6. Vagrant: file provisioning
   https://www.vagrantup.com/docs/provisioning/basic_usage.html
7. Vagrant: Tips
   https://www.vagrantup.com/docs/vagrantfile/tips.html
8. Vagrant: shell provisioning
   https://www.vagrantup.com/docs/provisioning/shell.html

8 Appendices:

8.1 Vagrant

Vagrant is an automation tool with a domain-specific language (DSL) that is used to automate the creation of VMs and VM environments. The idea is that a user can create a set of instructions, using Vagrant's DSL, that will set up one or more VMs and possibly configure those VMs. Every time the user uses the precreated set of instructions, the end result will look exactly the same. This can be beneficial for a number of use cases, including developers who want a consistent development environment or folks wanting to share a demo environment with other users.

Vagrant makes this work by using a number of different components:

- **Providers**: These are the "back end" of Vagrant. Vagrant itself doesn't provide any virtualization functionality; it relies on other products to do the heavy lifting. Providers are how Vagrant interacts with the products that will do the actual virtualization work. A provider could be VirtualBox (included by default with Vagrant), VMware Fusion, Hyper-V, vCloud Air, or AWS, just to name a few.

- **Boxes**: At the heart of Vagrant are boxes. Boxes are the predefined images that are used by Vagrant to build the environment according to the instructions provided by the user. A box may be a plain OS installation, or it may be an OS installation plus one or more applications installed. Boxes may support only a single provider or may support multiple providers (for example, a box might only work with VirtualBox, or it might support VirtualBox and VMware Fusion). It's important to note that multi-provider support by a box is really handled by multiple versions of a box (i.e, a version supporting VirtualBox, a version supporting AWS, or a version supporting VMware Fusion). A single box supports a single provider.

- **Vagrantfile**: The Vagrantfile contains the instructions from the user, expressed in Vagrant's DSL, on what the environment should look like—how many VMs, what type of VM, the provider, how they are connected, etc. Vagrantfiles are so named because the actual filename is Vagrantfile. The Vagrant DSL (and therefore Vagrantfiles) are based on Ruby.

COMMANDS:

```
Common commands:
box             manages boxes: installation, removal, etc.
connect         connect to a remotely shared Vagrant environment
destroy         stops and deletes all traces of the vagrant machine
global-status   outputs status Vagrant environments for this user
halt            stops the vagrant machine
help            shows the help for a subcommand
init            initializes a new Vagrant environment by creating a
   Vagrantfile
login           log in to HashiCorp's Atlas
package         packages a running vagrant environment into a box
          plugin          manages plugins: install, uninstall, update,
   etc.
port            displays information about guest port mappings
powershell      connects to machine via powershell remoting
provision       provisions the vagrant machine
push            deploys code in this environment to a configured
   destination
rdp             connects to machine via RDP
reload          restarts vagrant machine, loads new Vagrantfile
   configuration
resume          resume a suspended vagrant machine
share           share your Vagrant environment with anyone in the world
snapshot        manages snapshots: saving, restoring, etc.
ssh             connects to machine via SSH
ssh-config      outputs OpenSSH valid configuration to connect to the
   machine
status          outputs status of the vagrant machine
suspend         suspends the machine
up              starts and provisions the vagrant environment
version         prints current and latest Vagrant version
```

The most used ones are **vagrant up** that creates/starts or provisions the machines, **vagrant halt** that safely stops all machines and **vagrant destroy** which removes/deletes all machines and file related to them.

8.1.1 Vagrant File

In this project, the VagrantFile that is going to be used is the following one:

```
# -*- mode: ruby -*-
# vi: set ft=ruby :
```

```
Vagrant.configure("2") do |config|
# some shared setup
config.vm.box = "ubuntu/trusty64"
config.ssh.forward_agent = true
config.ssh.insert_key = false
#config.hostmanager.enabled = true
#config.cache.scope = :box

# The Ceph client will be our client machine to mount volumes and interact with the cluster
config.vm.define "ceph-client" do |client|
client.vm.hostname = "ceph-client"
client.vm.network :private_network, ip: "172.21.12.100"
client.vm.provision "file", source: "VMProvisioning/SSHKey/authorized_keys", destination: "~/.ssh/authorized_keys"
client.vm.provision :shell, path: "VMProvisioning/ClientProvision.sh"#, run: "never"
end

# Three nodes to be Ceph monitors
(0..2).each do |i|
config.vm.define "ceph-mon#{i}" do |mon|
mon.vm.hostname = "ceph-mon#{i}"
mon.vm.network :private_network, ip: "172.21.12.#{i+20}"
mon.vm.provision "file", source: "VMProvisioning/SSHKey/authorized_keys", destination: "~/.ssh/authorized_keys"
mon.vm.provision :shell, path: "VMProvisioning/MonProvision.sh", args: "mon#{i}" #, run: "never"
end
end

# Six nodes to be Ceph Object Storage Devices
(1..6).each do |i|
config.vm.define "ceph-osd#{i}" do |osd|
osd.vm.hostname = "ceph-osd#{i}"
osd.vm.network :private_network, ip: "172.21.12.#{i+30}"
osd.vm.provision "file", source: "VMProvisioning/SSHKey/authorized_keys", destination: "~/.ssh/authorized_keys"
osd.vm.provision :shell, path: "VMProvisioning/OSDProvision.sh", args: "osd#{i}"#, run: "never"
end
end

# Ceph admin machine to manage the cluster
config.vm.define "ceph-admin" do |admin|
admin.vm.hostname = "ceph-admin"
admin.vm.network :private_network, ip: "172.21.12.10"
admin.vm.provision "file", source: "VMProvisioning/SSHKey/id_rsa",
    destination: "~/.ssh/id_rsa" #, run: "always"
admin.vm.provision "file", source: "VMProvisioning/SSHKey/id_rsa.pub",
    destination: "~/.ssh/id_rsa.pub" #, run: "always"
admin.vm.provision :shell, path: "VMProvisioning/AdminProvision.sh" #,
    run: "always"
end
end
```

This file describes the scenario with all the machines defined by host names and private ip addresses. All share some common set up, like VM box, which is Ubuntu 14.04 (Trusty Tahr) 64 bits, and SSH configurations.

Ubuntu 16.04 is not sported, due a bug not resolved yet. When a VM is created using Ubuntu Xenial, we can't access to it, because vagrant user isn't set properly.

8.2 Scenario configurations

8.2.0.1 Cluster nodes initial provisioning Vagrant enables us to define the environment for our cluster scenario by allowing us to define an initial configuration to the cluster nodes. Setting the provisioning at the **VagrantFile** using files and scripts that run at the virtual machines when those are deploying, we can ensure that the nodes are ready to start operating at a known state.

8.2.0.2 SSH keys configuration Before we start installing the ceph software, we should ensure that the ssh connection between the admin node and all the cluster nodes is set properly. To do so, we use the files provision option. This option allows us to copy files when the machine is created specifying the source file at our local machine and the destination file at the virtual machines. Those files are going to be SSH key files, which we are going to configure as follows.

Create the RSA key at the host machine using the following command:

```
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/vagrant/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
```

```
Enter same passphrase again:
Your identification has been saved in /home/vagrant/.ssh/id_rsa.
Your public key has been saved in /home/vagrant/.ssh/id_rsa.pub.
The key fingerprint is:
2d:fb:f0:4e:83:33:c4:bc:67:8a:aa:a9:b1:75:5c:c7 vagrant@ceph-admin
The key's randomart image is:
+--[ RSA 2048]----+
|                 |
|                 |
|                 |
|       + .       |
|      . E .      |
|   . . o =       |
|. . o   B =      |
| + o   . @ .     |
|o.o.... ..+      |
+-----------------+
```

This command creates rsa key files (private and public). *id_rsa* should be copied to .ssh/config directory at the Admin machine, so this machine can connect to the rest of the nodes without prompting for credentials. Public RSA key is stored at *id_rsa.pub* and it has to be copied to the monitors and OSDs.

To provision the nodes with the RSA public key, the *id_rsa.pub* must be added append the authorized keys file:

```
$ cat id_rsa.pub >> authorized_keys
```

Then, this file is copied to the OSDs, monitors and clients that form the cluster using the following command (line 15 at the **VagrantFile** for the **ceph-client** provisioning) :

```
client.vm.provision "file", source: "VMProvisioning/SSHKey/
   authorized_keys", destination: "~/.ssh/authorized_keys"
```

8.2.0.3 Client provisioning We provision all the clients (in this scenario we only create one) using script provisioning. Once the machine is created, Vagrant runs the provision scripts that we pass indicating the path to these scripts and the programing language.

```
client.vm.provision :shell, path: "VMProvisioning/ClientProvision.sh" ,
    run: "always"
```

With the parameter *run* we can decide whether we want to apply the provision each time that the *vagrant up* command is called (with the "always" option) or never provisioning (with the "never" option).

By default, provisioning is called just once. Only when the machine is created for the first time or when it has been destroy previously and we call the *vagrant up* command.

For client machines, we invoke the following script:

```
#!/usr/bin/env bash
apt-get update
apt-get install ntp
apt-get install ceph-deploy
```

This script updates the machine repositories and installs ntp and ceph-deploy tools. The first one is the Network Time Protocol which synchronizes the virtual machines clocks over the network. The second one is the main ceph tool that we are going to use to connect our client to the cluster through the admin.

8.2.0.4 Monitors provisioning Monitors are the key parts of the cluster, so we have to be sure that all our monitors are synchronized. In this case, NTP protocol is critic when we are creating the machines and provisioning the packages.

```
#!/usr/bin/env bash
apt-get update
apt-get install ntp
apt-get install ceph-deploy
```

8.2.0.5 OSDs provisioning Object Storage Devices need to be provisioned with NTP protocol and the ceph-deploy software as well. Also, a storage directory must be created and chowned with a user with root privileges, in this vagrant user by default has those privileges.

```
#!/usr/bin/env bash
apt-get update
apt-get install ntp
apt-get install ceph-deploy
sudo mkdir /storage$1 && sudo chown vagrant:vagrant /storage$1
```

The provisioning script *args* parameter combined with the bash first input argument *$1*, allows us to differentiate all the storage directories or paths among all the OSDs.

```
osd.vm.provision :shell, path: "VMProvisioning/OSDProvision.sh", args:
    "osd#{i}"
```

8.2.0.6 Admin provisioning

The ceph cluster is managed by the **ceph-admin** node. The basic provisions for this node are going to be the same as we did for the previous nodes. The advanced configuration of this cluster is going to be performed by the `ceph-deploy` package.

Before we start the main ceph configuration, we should guarantee the connection between the Admin and the rest of the nodes. To do so, we configure the *.ssh/config* file with the RSA keys that we created previously adding the path to the private key. This private key is the Indentityfile paramater for the SSH connection. Also, we must map every host with its IP at he */etc/hosts* file.

Finally, we create the *ceph-cluster* directory, This directory contains all the cluster configuration files. We are going to make all the deploys and configurations from it. *Ceph-cluster* includes keyring files for the OSDs, monitors and the main admin keyring. Also, the *ceph.conf* file, which has the cluster configuration, and the log file.

```

#!/usr/bin/env bash
apt-get update
apt-get install ntp
apt-get install ceph-deploy

echo '
Host ceph-client

HostName 172.21.12.100
User vagrant
Identityfile ~/.ssh/id_rsa
Port 22
StrictHostKeyChecking no

Host ceph-mon0
HostName 172.21.12.20
User vagrant
```

```
Identityfile ~/.ssh/id_rsa
Port 22
StrictHostKeyChecking no

Host ceph-mon1
HostName 172.21.12.21
User vagrant
Identityfile ~/.ssh/id_rsa
Port 22
StrictHostKeyChecking no

Host ceph-mon2
HostName 172.21.12.22
User vagrant
Identityfile ~/.ssh/id_rsa
Port 22
StrictHostKeyChecking no

Host ceph-osd1
HostName 172.21.12.31
User vagrant
Identityfile ~/.ssh/id_rsa
Port 22
StrictHostKeyChecking no

Host ceph-osd2
HostName 172.21.12.32
User vagrant
Identityfile ~/.ssh/id_rsa
Port 22
StrictHostKeyChecking no

Host ceph-osd3
HostName 172.21.12.33
User vagrant
Identityfile ~/.ssh/id_rsa
Port 22
StrictHostKeyChecking no

Host ceph-osd4
HostName 172.21.12.34
User vagrant
Identityfile ~/.ssh/id_rsa
Port 22
StrictHostKeyChecking no
```

```
Host ceph-osd5
HostName 172.21.12.35
User vagrant
Identityfile ~/.ssh/id_rsa
Port 22
StrictHostKeyChecking no

Host ceph-osd6
HostName 172.21.12.36
User vagrant
Identityfile ~/.ssh/id_rsa
Port 22
StrictHostKeyChecking no' > .ssh/config

echo '
172.21.12.10 ceph-admin
172.21.12.100 ceph-client
172.21.12.20 ceph-mon0
172.21.12.21 ceph-mon1
172.21.12.22 ceph-mon2
172.21.12.31 ceph-osd1
172.21.12.32 ceph-osd2
172.21.12.33 ceph-osd3
172.21.12.34 ceph-osd4
172.21.12.35 ceph-osd5
172.21.12.36 ceph-osd6'> /etc/hosts
```

8.2.1 Cluster nodes configuration

Now that we have all the machines that are going to be part of the cluster, we are going to set a basic cluster with one monitor and three OSDs. When this initial configuration has end deploying, we are going to check the cluster status and then add two more monitors and the rest of the OSDs. Finally, we are going to add the client and store some basic files to see how the storage algorithm works.

Before we start deploying, it's important to notice that we can to clean settings and re-configure again, we must do like follows.

Remove packages:

```
vagrant@ceph-admin:~/ceph-cluster$ ceph-deploy purge ceph_node
```

Remove settings:

```
vagrant@ceph-admin:~/ceph-cluster$ ceph-deploy purgedata ceph_node
```

```
vagrant@ceph-admin:~/ceph-cluster$ ceph-deploy forgetkeys
```

8.2.1.1 Initial cluster configuration

In order to create our cluster, we are going yo use `ceph-deploy` command. As said previously, the main cluster configuration is going to be allocated at *ceph-cluster* directory at **ceph-admin** machine. Make sure that all deploys are made from this directory.

First, we create a new cluster indicating the monitor's hostname, fqdn or hostname:fqdn pair. We can set as many monitors as we want for the initial deploy, but they have to be configured at *ceph.conf*. In our scenario, we are going to set **ceph-mon0** as the initial monitor for the cluster, and then, add two more in order to avoid a single point of failure. To do so, we execute the following command:

```
vagrant@ceph-admin:~/ceph-cluster$ ceph-deploy new ceph-mon0
[ceph_deploy.cli][INFO  ] Invoked (1.4.0): /usr/bin/ceph-deploy new
    ceph-mon0
[ceph_deploy.new][DEBUG ] Creating new cluster named ceph
[ceph_deploy.new][DEBUG ] Resolving host ceph-mon0
[ceph_deploy.new][DEBUG ] Monitor ceph-mon0 at 172.21.12.20
[ceph_deploy.new][INFO  ] making sure passwordless SSH succeeds
[ceph-mon0][DEBUG ] connected to host: ceph-admin
[ceph-mon0][INFO  ] Running command: ssh -CT -o BatchMode=yes ceph-mon0
[ceph_deploy.new][DEBUG ] Monitor initial members are ['ceph-mon0']
[ceph_deploy.new][DEBUG ] Monitor addrs are ['172.21.12.20']
[ceph_deploy.new][DEBUG ] Creating a random mon key...
[ceph_deploy.new][DEBUG ] Writing initial config to ceph.conf...
[ceph_deploy.new][DEBUG ] Writing monitor keyring to ceph.mon.keyring
    ...
```

As we can see, `ceph-deploy new` creates a new cluster called **ceph** and writes its configuration at *ceph.conf* file. Also creates a secret key called *ceph.mon.keyring* which monitors use to communicate with each other.

Now, in order to add our first two OSDs, we must configure de cluster configuration file to set cluster's default OSDs pool size. We add append this file the following code line:

```
vagrant@ceph-admin:~/ceph-cluster$ echo 'osd pool default size = 2' >>
    ceph.conf && cat ceph.conf
[global]
fsid = 171ac133-fb55-4a8e-b1b7-c1140ca46f8e
```

```
mon_initial_members = ceph-mon0
mon_host = 172.21.12.20
auth_cluster_required = cephx
auth_service_required = cephx
auth_client_required = cephx
filestore_xattr_use_omap = true

osd pool default size = 2
```

ceph.conf contains the **fsid**, that stands for File System ID, which is the unique identifier for our cluster. Also includes monitor's parameters, such the hostname and the mapped ip. All the authentications with the cluster, are going to be managed by cephx protocol.

Also, we notice that Extended Atributes (XATTRs) are allowed to use object map for ext4 file systems. This may improve performance by using a method of storing XATTRs that is extrinsic to the underlying filesystem. This option can be removed, because we are going to use **XFS** as the Filesystem for the OSDs, since it's more efficient and stable. Finally, we added the OSDs pool size mentioned.

Before we start configuring the monitor and adding our OSDs, we must install Ceph packages on our cluster nodes typing this command line:

```
vagrant@ceph-admin:~/ceph-cluster$ ceph-deploy install ceph-admin ceph-
    mon0 ceph-osd1 ceph-osd2
```

Now that we have all the packages installed, we are going to add the monitor to the cluster. The following command deploys the monitors defined in mon initial members parameter set at the *ceph.conf* file and then gather keys. Gathering keys means that *Ceph CLI* contacts the mon and creates monitor keyring or retrieves the ones stored in their internal store.

```
vagrant@ceph-admin:~/ceph-cluster$ ceph-deploy mon create-initial
[ceph_deploy.cli][INFO  ] Invoked (1.4.0): /usr/bin/ceph-deploy mon
    create-initial
[ceph_deploy.mon][DEBUG ] Deploying mon, cluster ceph hosts ceph-mon0
[ceph_deploy.mon][DEBUG ] detecting platform for host ceph-mon0 ...
[ceph-mon0][DEBUG ] connected to host: ceph-mon0
[ceph-mon0][DEBUG ] detect platform information from remote host
[ceph-mon0][DEBUG ] detect machine type
[ceph_deploy.mon][INFO  ] distro info: Ubuntu 14.04 trusty
[ceph-mon0][DEBUG ] determining if provided host has same hostname in
    remote
```

```
[ceph-mon0][DEBUG ] get remote short hostname
[ceph-mon0][DEBUG ] deploying mon to ceph-mon0
[ceph-mon0][DEBUG ] get remote short hostname
[ceph-mon0][DEBUG ] remote hostname: ceph-mon0
[ceph-mon0][DEBUG ] write cluster configuration to /etc/ceph/{cluster}.
    conf
[ceph-mon0][DEBUG ] create the mon path if it does not exist
[ceph-mon0][DEBUG ] checking for done path: /var/lib/ceph/mon/ceph-ceph
    -mon0/done
[ceph-mon0][DEBUG ] done path does not exist: /var/lib/ceph/mon/ceph-
    ceph-mon0/done
[ceph-mon0][INFO  ] creating keyring file: /var/lib/ceph/tmp/ceph-ceph-
    mon0.mon.keyring
[ceph-mon0][DEBUG ] create the monitor keyring file
[ceph-mon0][INFO  ] Running command: sudo ceph-mon --cluster ceph --
    mkfs -i ceph-mon0 --keyring /var/lib/ceph/tmp/ceph-ceph-mon0.mon.
    keyring
[ceph-mon0][DEBUG ] ceph-mon: mon.noname-a 172.21.12.20:6789/0 is local
    , renaming to mon.ceph-mon0
[ceph-mon0][DEBUG ] ceph-mon: set fsid to 171ac133-fb55-4a8e-b1b7-
    c1140ca46f8e
[ceph-mon0][DEBUG ] ceph-mon: created monfs at /var/lib/ceph/mon/ceph-
    ceph-mon0 for mon.ceph-mon0
[ceph-mon0][INFO  ] unlinking keyring file /var/lib/ceph/tmp/ceph-ceph-
    mon0.mon.keyring
[ceph-mon0][DEBUG ] create a done file to avoid re-doing the mon
    deployment
[ceph-mon0][DEBUG ] create the init path if it does not exist
[ceph-mon0][DEBUG ] locating the `service` executable...
[ceph-mon0][INFO  ] Running command: sudo initctl emit ceph-mon cluster
    =ceph id=ceph-mon0
[ceph-mon0][INFO  ] Running command: sudo ceph --cluster=ceph --admin-
    daemon /var/run/ceph/ceph-mon.ceph-mon0.asok mon_status
[ceph-mon0][DEBUG ]
    ********************************************************************************

[ceph-mon0][DEBUG ] status for monitor: mon.ceph-mon0
[ceph-mon0][DEBUG ] {
[ceph-mon0][DEBUG ]   "election_epoch": 2,
[ceph-mon0][DEBUG ]   "extra_probe_peers": [],
[ceph-mon0][DEBUG ]   "monmap": {
[ceph-mon0][DEBUG ]     "created": "0.000000",
[ceph-mon0][DEBUG ]     "epoch": 1,
[ceph-mon0][DEBUG ]     "fsid": "171ac133-fb55-4a8e-b1b7-c1140ca46f8e",
[ceph-mon0][DEBUG ]     "modified": "0.000000",
[ceph-mon0][DEBUG ]     "mons": [
[ceph-mon0][DEBUG ]         {
[ceph-mon0][DEBUG ]           "addr": "172.21.12.20:6789/0",
[ceph-mon0][DEBUG ]           "name": "ceph-mon0",
[ceph-mon0][DEBUG ]           "rank": 0
[ceph-mon0][DEBUG ]         }
[ceph-mon0][DEBUG ]       ]
[ceph-mon0][DEBUG ]     },
[ceph-mon0][DEBUG ]     "name": "ceph-mon0",
[ceph-mon0][DEBUG ]     "outside_quorum": [],
[ceph-mon0][DEBUG ]     "quorum": [
[ceph-mon0][DEBUG ]       0
[ceph-mon0][DEBUG ]     ],
[ceph-mon0][DEBUG ]     "rank": 0,
[ceph-mon0][DEBUG ]     "state": "leader",
[ceph-mon0][DEBUG ]     "sync_provider": []
[ceph-mon0][DEBUG ]   }
[ceph-mon0][DEBUG ]
   ********************************************************************************

[ceph-mon0][INFO  ] monitor: mon.ceph-mon0 is running
[ceph-mon0][INFO  ] Running command: sudo ceph --cluster=ceph --admin-
   daemon /var/run/ceph/ceph-mon.ceph-mon0.asok mon_status
[ceph_deploy.mon][INFO  ] processing monitor mon.ceph-mon0
[ceph-mon0][DEBUG ] connected to host: ceph-mon0
[ceph-mon0][INFO  ] Running command: sudo ceph --cluster=ceph --admin-
   daemon /var/run/ceph/ceph-mon.ceph-mon0.asok mon_status
[ceph_deploy.mon][INFO  ] mon.ceph-mon0 monitor has reached quorum!
[ceph_deploy.mon][INFO  ] all initial monitors are running and have
   formed quorum
[ceph_deploy.mon][INFO  ] Running gatherkeys...
[ceph_deploy.gatherkeys][DEBUG ] Checking ceph-mon0 for /etc/ceph/ceph.
   client.admin.keyring
[ceph-mon0][DEBUG ] connected to host: ceph-mon0
[ceph-mon0][DEBUG ] detect platform information from remote host
[ceph-mon0][DEBUG ] detect machine type
[ceph-mon0][DEBUG ] fetch remote file
[ceph_deploy.gatherkeys][DEBUG ] Got ceph.client.admin.keyring key from
    ceph-mon0.
[ceph_deploy.gatherkeys][DEBUG ] Have ceph.mon.keyring
[ceph_deploy.gatherkeys][DEBUG ] Checking ceph-mon0 for /var/lib/ceph/
   bootstrap-osd/ceph.keyring
[ceph-mon0][DEBUG ] connected to host: ceph-mon0
[ceph-mon0][DEBUG ] detect platform information from remote host
[ceph-mon0][DEBUG ] detect machine type
[ceph-mon0][DEBUG ] fetch remote file
[ceph_deploy.gatherkeys][DEBUG ] Got ceph.bootstrap-osd.keyring key
```

```
   from ceph-mon0.
[ceph_deploy.gatherkeys][DEBUG ] Checking ceph-mon0 for /var/lib/ceph/
   bootstrap-mds/ceph.keyring
[ceph-mon0][DEBUG ] connected to host: ceph-mon0
[ceph-mon0][DEBUG ] detect platform information from remote host
[ceph-mon0][DEBUG ] detect machine type
[ceph-mon0][DEBUG ] fetch remote file
```

This command's output shows us that the cluster configuration is stored to /etc/ceph/-ceph.conf at he **ceph-mon0** machine. Then manages the keyring and creates quorum state and defines it as a leader. `ceph-deploy mon` checks if there are more monitors and if they form quorum.

The next step is to prepare the OSDs. For that, we are going to use *ceph-deploy osd prepare* indicating the osd machine and the directory where de objects are going to be stored. In our case, for ceph-osd1 and ceph-osd2 at the directories created at machine provisioning. So, the command that we should execute is this one:

```
vagrant@ceph-admin:~/ceph-cluster$ ceph-deploy osd prepare ceph-osd1:/
   storageosd1 ceph-osd2:/storageosd2
[ceph_deploy.cli][INFO  ] Invoked (1.4.0): /usr/bin/ceph-deploy osd
   prepare ceph-osd1:/storageosd1 ceph-osd2:/storageosd2
[ceph_deploy.osd][DEBUG ] Preparing cluster ceph disks ceph-osd1:/
   storageosd1: ceph-osd2:/storageosd2:
[ceph-osd1][DEBUG ] connected to host: ceph-osd1
[ceph-osd1][DEBUG ] detect platform information from remote host
[ceph-osd1][DEBUG ] detect machine type
[ceph_deploy.osd][INFO  ] Distro info: Ubuntu 14.04 trusty
[ceph_deploy.osd][DEBUG ] Deploying osd to ceph-osd1
[ceph-osd1][DEBUG ] write cluster configuration to /etc/ceph/{cluster}.
   conf
[ceph-osd1][WARNIN] osd keyring does not exist yet, creating one
[ceph-osd1][DEBUG ] create a keyring file
[ceph-osd1][INFO  ] Running command: sudo udevadm trigger --subsystem-
   match=block --action=add
[ceph_deploy.osd][DEBUG ] Preparing host ceph-osd1 disk /storageosd1
   journal None activate False
[ceph-osd1][INFO  ] Running command: sudo ceph-disk-prepare --fs-type
   xfs --cluster ceph -- /storageosd1
[ceph_deploy.osd][DEBUG ] Host ceph-osd1 is now ready for osd use.
[ceph-osd2][DEBUG ] connected to host: ceph-osd2
[ceph-osd2][DEBUG ] detect platform information from remote host
[ceph-osd2][DEBUG ] detect machine type
[ceph_deploy.osd][INFO  ] Distro info: Ubuntu 14.04 trusty
```

```
[ceph_deploy.osd][DEBUG ] Deploying osd to ceph-osd2
[ceph-osd2][DEBUG ] write cluster configuration to /etc/ceph/{cluster}.
    conf
[ceph-osd2][WARNIN] osd keyring does not exist yet, creating one
[ceph-osd2][DEBUG ] create a keyring file
[ceph-osd2][INFO  ] Running command: sudo udevadm trigger --subsystem-
    match=block --action=add
[ceph_deploy.osd][DEBUG ] Preparing host ceph-osd2 disk /storageosd2
    journal None activate False
[ceph-osd2][INFO  ] Running command: sudo ceph-disk-prepare --fs-type
    xfs --cluster ceph -- /storageosd2
[ceph_deploy.osd][DEBUG ] Host ceph-osd2 is now ready for osd use.
Unhandled exception in thread started by
sys.excepthook is missing
lost sys.stderr
```

Now, we have the OSDs prepared. The executed command copies the ceph.conf file to /etc/ceph directory. Then, checks if there is an existing OSD keyring, and if not, it creates one. Also, *ceph_deploy osd* prepares the storage directory (e.g., /storageosd1 at ceph-osd1) defining the file system type, which can be xfs,btrfs or ext4. By default, the file system is set to **xfs**, but we can change it with the parameter *–fs-type FS_TYPE*, where the FS_TYPE can be one of the file systems mentioned previously.

In order to join the OSDs to the cluster, we must execute the following command:

```
vagrant@ceph-admin:~/ceph-cluster$ ceph-deploy osd activate ceph-osd1:/
    storageosd1 ceph-osd2:/storageosd2
[ceph_deploy.cli][INFO  ] Invoked (1.4.0): /usr/bin/ceph-deploy osd
    activate ceph-osd1:/storageosd1 ceph-osd2:/storageosd2
[ceph_deploy.osd][DEBUG ] Activating cluster ceph disks ceph-osd1:/
    storageosd1: ceph-osd2:/storageosd2:
[ceph-osd1][DEBUG ] connected to host: ceph-osd1
[ceph-osd1][DEBUG ] detect platform information from remote host
[ceph-osd1][DEBUG ] detect machine type
[ceph_deploy.osd][INFO  ] Distro info: Ubuntu 14.04 trusty
[ceph_deploy.osd][DEBUG ] activating host ceph-osd1 disk /storageosd1
[ceph_deploy.osd][DEBUG ] will use init type: upstart
[ceph-osd1][INFO  ] Running command: sudo ceph-disk-activate --mark-
    init upstart --mount /storageosd1
[ceph-osd1][WARNIN] got monmap epoch 1
[ceph-osd1][WARNIN] 2017-01-06 17:56:28.814090 7f48d16a9800 -1 journal
    FileJournal::_open: disabling aio for non-block journal.  Use
    journal_force_aio to force use of aio anyway
[ceph-osd1][WARNIN] 2017-01-06 17:56:28.864092 7f48d16a9800 -1 journal
```

```
   FileJournal::_open: disabling aio for non-block journal.  Use
   journal_force_aio to force use of aio anyway
[ceph-osd1][WARNIN] 2017-01-06 17:56:28.865478 7f48d16a9800 -1
   filestore(/storageosd1) could not find 23c2fcde/osd_superblock/0//-1
    in index: (2) No such file or directory
[ceph-osd1][WARNIN] 2017-01-06 17:56:28.873565 7f48d16a9800 -1 created
   object store /storageosd1 journal /storageosd1/journal for osd.0
   fsid 171ac133-fb55-4a8e-b1b7-c1140ca46f8e
[ceph-osd1][WARNIN] 2017-01-06 17:56:28.873633 7f48d16a9800 -1 auth:
   error reading file: /storageosd1/keyring: can't open /storageosd1/
   keyring: (2) No such file or directory
[ceph-osd1][WARNIN] 2017-01-06 17:56:28.873725 7f48d16a9800 -1 created
   new key in keyring /storageosd1/keyring
[ceph-osd1][WARNIN] added key for osd.0
[ceph-osd2][DEBUG ] connected to host: ceph-osd2
[ceph-osd2][DEBUG ] detect platform information from remote host
[ceph-osd2][DEBUG ] detect machine type
[ceph_deploy.osd][INFO  ] Distro info: Ubuntu 14.04 trusty
[ceph_deploy.osd][DEBUG ] activating host ceph-osd2 disk /storageosd2
[ceph_deploy.osd][DEBUG ] will use init type: upstart
[ceph-osd2][INFO  ] Running command: sudo ceph-disk-activate --mark-
   init upstart --mount /storageosd2
[ceph-osd2][WARNIN] got monmap epoch 1
[ceph-osd2][WARNIN] 2017-01-06 17:56:31.394333 7fc519319800 -1 journal
   FileJournal::_open: disabling aio for non-block journal.  Use
   journal_force_aio to force use of aio anyway
[ceph-osd2][WARNIN] 2017-01-06 17:56:31.444406 7fc519319800 -1 journal
   FileJournal::_open: disabling aio for non-block journal.  Use
   journal_force_aio to force use of aio anyway
[ceph-osd2][WARNIN] 2017-01-06 17:56:31.445995 7fc519319800 -1
   filestore(/storageosd2) could not find 23c2fcde/osd_superblock/0//-1
    in index: (2) No such file or directory
[ceph-osd2][WARNIN] 2017-01-06 17:56:31.459293 7fc519319800 -1 created
   object store /storageosd2 journal /storageosd2/journal for osd.1
   fsid 171ac133-fb55-4a8e-b1b7-c1140ca46f8e
[ceph-osd2][WARNIN] 2017-01-06 17:56:31.459390 7fc519319800 -1 auth:
   error reading file: /storageosd2/keyring: can't open /storageosd2/
   keyring: (2) No such file or directory
[ceph-osd2][WARNIN] 2017-01-06 17:56:31.459625 7fc519319800 -1 created
   new key in keyring /storageosd2/keyring
[ceph-osd2][WARNIN] added key for osd.1
```

When this command execution ends, we have the OSDs joined to the cluster with their storage directories mounted. Now. we must copy the configuration file and admin key to our admin node and the cluster nodes, in order to be able to use Ceph CLI tools without having to specify the monitor address and keyring each time we execute a command. To do so, we execute the `ceph-deploy admin` for all the nodes:

```
vagrant@ceph-admin:~/ceph-cluster$ ceph-deploy admin ceph-mon0 ceph-
    osd1 ceph-osd2
```

Then, as a final step of the initial configuration, we must ensure that we have the correct permissions for the **ceph.client.admin.keyring** on every node, that allows us to communicate with the cluster client:

```
vagrant@ceph-admin:~/ceph-cluster$ sudo chmod +r  /etc/ceph/ceph.client
    .admin.keyring
vagrant@ceph-admin:~/ceph-cluster$ ssh ceph-mon0 sudo chmod +r  /etc/
    ceph/ceph.client.admin.keyring
vagrant@ceph-admin:~/ceph-cluster$ ssh ceph-osd1 sudo chmod +r  /etc/
    ceph/ceph.client.admin.keyring
vagrant@ceph-admin:~/ceph-cluster$ ssh ceph-osd2 sudo chmod +r  /etc/
    ceph/ceph.client.admin.keyring
```

Finally, we check the cluster status and configuration :

```
vagrant@ceph-admin:~/ceph-cluster$ ceph -s
cluster a8dac0ff-01d4-4aa4-91e1-f8d6ae930184
health HEALTH_OK
monmap e1: 1 mons at {ceph-mon0=172.21.12.20:6789/0}, election epoch 2,
     quorum 0 ceph-mon0
osdmap e6: 2 osds: 2 up, 2 in
pgmap v9: 192 pgs, 3 pools, 0 bytes data, 0 objects
13383 MB used, 63795 MB / 80568 MB avail
192 active+clean
```

As we expected, the cluster health is OK. We can see the monitors map, in this case, one monitor (ceph-mon0) at quorum state. Also, is important to notice the OSDmap, both OSDs are in and up. Is important to differentiate between in and up states; The first one indicates that the OSD forms part of the cluster, and the second one means that is active and running. Pgmap stands for Placement Groups Map and on its initial state we don't have data stored yet.

8.2.1.2 Adding new nodes to the cluster

8.2.1.2.1 Monitors Ceph clusters demands an odd number of monitors in order to form quorum and make decisions. In our case, as we want a small cluster we are going to add two more monitors to our initial configuration. As we configured all the machines previously using Vagrant, we only need to execute three commands.

First, install packages to the monitors:

```
vagrant@ceph-admin:~/ceph-cluster$ ceph-deploy install ceph-mon1 ceph-
    mon2
```

We add the first monitor using `ceph-deploy mon add`:

```
vagrant@ceph-admin:~/ceph-cluster$ ceph-deploy mon add ceph-mon1
```

Then, the second one:

```
vagrant@ceph-admin:~/ceph-cluster$ ceph-deploy mon add ceph-mon2
```

These new monitors are added to the cluster and set as peons of the leader monitor (**ceph-mon0**). To check the cluster quorum status we need to execute the following command:

```
vagrant@ceph-admin:~/ceph-cluster$ ceph quorum_status --format json-
    pretty
{ "election_epoch": 8,
"quorum": [
0,
1,
2],
"quorum_names": [
"ceph-mon0",
"ceph-mon1",
"ceph-mon2"],
"quorum_leader_name": "ceph-mon0",
"monmap": { "epoch": 3,
"fsid": "a8dac0ff-01d4-4aa4-91e1-f8d6ae930184",
"modified": "2017-01-07 10:04:50.745416",
"created": "0.000000",
"mons": [
{ "rank": 0,
"name": "ceph-mon0",
"addr": "172.21.12.20:6789\/0"},
{ "rank": 1,
"name": "ceph-mon1",
"addr": "172.21.12.21:6789\/0"},
{ "rank": 2,
"name": "ceph-mon2",
"addr": "172.21.12.22:6789\/0"}]}}
```

As we did with the other nodes, we must copy the file configuration and keyrings to these new nodes. Then, we check the *client.admin.keyring* file permissions:

```
vagrant@ceph-admin:~/ceph-cluster$ ceph-deploy admin ceph-mon1 ceph-
   mon2
vagrant@ceph-admin:~/ceph-cluster$ ssh ceph-mon1 sudo chmod +r  /etc/
   ceph/ceph.client.admin.keyring
vagrant@ceph-admin:~/ceph-cluster$ ssh ceph-mon2 sudo chmod +r  /etc/
   ceph/ceph.client.admin.keyring
```

Finally, we check the cluster status with `ceph -s`:

```
vagrant@ceph-admin:~/ceph-cluster$ ceph -s
cluster a8dac0ff-01d4-4aa4-91e1-f8d6ae930184
health HEALTH_OK
monmap e3: 3 mons at {ceph-mon0=172.21.12.20:6789/0,ceph-mon1
   =172.21.12.21:6789/0,ceph-mon2=172.21.12.22:6789/0}, election epoch
   8, quorum 0,1,2 ceph-mon0,ceph-mon1,ceph-mon2
osdmap e15: 2 osds: 2 up, 2 in
pgmap v32: 192 pgs, 3 pools, 0 bytes data, 0 objects
13388 MB used, 63790 MB / 80568 MB avail
192 active+clean
```

8.2.1.2.2 OSDs OSD are not limited, so we can add as many OSDs as we want. Ceph developers recommend one OSDs per path, rather than one per machine. In our case, we have one OSD per storage directory, so we add four more that make a total of six OSDs forming part of our cluster.

First step is installing the ceph files and configuration as we did with the other nodes:

```
vagrant@ceph-admin:~/ceph-cluster$ ceph-deploy install ceph-osd3 ceph-
   osd4 ceph-osd5 ceph-osd6
```

Then, we prepare and activate them:

```
vagrant@ceph-admin:~/ceph-cluster$ ceph-deploy osd prepare ceph-osd3:/
    storageosd3 ceph-osd4:/storageosd4 ceph-osd5:/storageosd5 ceph-osd6
    :/storageosd6
vagrant@ceph-admin:~/ceph-cluster$ ceph-deploy osd activate ceph-osd3:/
    storageosd3 ceph-osd4:/storageosd4 ceph-osd5:/storageosd5 ceph-osd6
    :/storageosd6
```

Finally, we execute the `ceph-deploy admin` to them and check the keyring permissions:

```
vagrant@ceph-admin:~/ceph-cluster$ ceph-deploy admin ceph-osd3 ceph-
    osd4 ceph-osd5 ceph-osd6
vagrant@ceph-admin:~/ceph-cluster$ ssh ceph-osd3 sudo chmod +r  /etc/
    ceph/ceph.client.admin.keyring
vagrant@ceph-admin:~/ceph-cluster$ ssh ceph-osd4 sudo chmod +r  /etc/
    ceph/ceph.client.admin.keyring
vagrant@ceph-admin:~/ceph-cluster$ ssh ceph-osd5 sudo chmod +r  /etc/
    ceph/ceph.client.admin.keyring
vagrant@ceph-admin:~/ceph-cluster$ ssh ceph-osd6 sudo chmod +r  /etc/
    ceph/ceph.client.admin.keyring
```

Then, we check the cluster status:

```
vagrant@ceph-admin:~/ceph-cluster$ ceph -s
cluster a8dac0ff-01d4-4aa4-91e1-f8d6ae930184
health HEALTH_WARN clock skew detected on mon.ceph-mon1, mon.ceph-mon2
    Monitor clock skew detected
monmap e3: 3 mons at {ceph-mon0=172.21.12.20:6789/0,ceph-mon1
    =172.21.12.21:6789/0,ceph-mon2=172.21.12.22:6789/0}, election epoch
    10, quorum 0,1,2 ceph-mon0,ceph-mon1,ceph-mon2
osdmap e28: 6 osds: 6 up, 6 in
pgmap v63: 192 pgs, 3 pools, 0 bytes data, 0 objects
40149 MB used, 186 GB / 236 GB avail
192 active+clean
```

Now, we can see that we have all our OSD in and up ready to store objects. As we did with **ceph-osd1** and **ceph-osd2**, we are going to leave XFS as the default File System.

8.2.1.2.3 Client Clients interact with the cluster using cephx protocol. This protocol uses keyrings to perform a secure connection between the client and the cluster to store or read data. In our case, we are going to set our client as a Block Device, but it can be any type of Ceph Client (e.g., Object Storage, Filesystem, API,etc).

Ceph Monitors, OSDs and Metadata Servers use cephx protocol too. In order to manage and identify the nodes and connections, Ceph uses the notation **TYPE.ID**; where **TYPE** refers to Ceph's nodes type, such as Monitor, OSD or client, and **ID** is an unique identifier for each type. In order to clarify this notation, we can see how our OSDs and clients are identified usign `ceph auth list`:

```
vagrant@ceph-admin:~/ceph-cluster$ ceph auth list
installed auth entries:

osd.0
key: AQCX6G9YcHoQBRAAoppb15cquF/r1A81b6lBtw==
caps: [mon] allow profile osd
caps: [osd] allow *
osd.1
key: AQCY6G9Y2PuKIxAAJfZ/HAapXtfT0HYSAF0LoA==
caps: [mon] allow profile osd
caps: [osd] allow *
osd.2
key: AQAQJ3FY0KoFDBAA5QxkmOJZpoX3SpHYEJKm7g==
caps: [mon] allow profile osd
caps: [osd] allow *
osd.3
key: AQAUJ3FYMKXeLRAAX9NDS6l/4hqgL5xRzJVTUw==
caps: [mon] allow profile osd
caps: [osd] allow *
osd.4
key: AQAYJ3FYQPMEMRAARLrvfMObeWmc62jubJJnCA==
caps: [mon] allow profile osd
caps: [osd] allow *
osd.5
key: AQAmJ3FY2JJeDhAAgOTH1R6loTKrPo1YjXgXjw==
caps: [mon] allow profile osd
caps: [osd] allow *
client.admin
key: AQB76G9Y8CG6GhAAUHN9XPa6r41a0DM9ifw1Zw==
caps: [mds] allow
caps: [mon] allow *
caps: [osd] allow *
client.bootstrap-mds
key: AQB76G9YmKZdLBAAm9pYstcYE5maHQjIGXPgqQ==
caps: [mon] allow profile bootstrap-mds
```

```
client.bootstrap-osd
key: AQB76G9YiL+yIxAAt115oY5mo9+GXIzl/68Mnw==
caps: [mon] allow profile bootstrap-osd
```

Every OSD has his ID from 0 to 5, with an unique key and capabilities that can be set using `ceph auth` command. In order to add our client to the cluster, we can create different users defined by *client.userID*. Admin is the default user, so when we interact with the cluster without specifying the user, cephx uses the default keyring (ceph.client.admin.keyring), with full auth capabilities.

To add our client to the cluster, we just need to install ceph packages, copy keyring file nad make sure we have the right permissions on this file. Same steps as we did with the Monitors and OSDs, but without any specific roll in the cluster:

```
vagrant@ceph-admin:~/ceph-cluster$ ceph-deploy install ceph-client
vagrant@ceph-admin:~/ceph-cluster$ ceph-deploy admin ceph-client
vagrant@ceph-admin:~/ceph-cluster$ ssh ceph-client sudo chmod +r  /etc/
    ceph/ceph.client.admin.keyring
```

8.2.1.2.4 RADOS pools

Now that we have all the packages and files installed on our client, we are going to create a storage pool to store and retrieve files from it.

By default, there are three types of pools:

```
vagrant@ceph-client:~$ rados lspools
data
metadata
rbd
```

We can create OSD pools and store objects in them with the following commands:

```
vagrant@ceph-client:~$ rados mkpool datatest
successfully created pool datatest
vagrant@ceph-client:~$ rados put test-object-1 test1 --pool=datatest
```

```
vagrant@ceph-client:~$ rados put test-object-2 testvideo.mkv --pool=
    datatest
```

With *rados put* we can decide what will be the identifier of the stored object. In this case, the files that we store are a **4KB TXT** file (*test1*) and a **1.4GB MKV** video file (*testvideo.mkv*).

```
vagrant@ceph-client:~$ ls -sh
total 1.4G
4.0K test1  1.4G testvideo.mkv
```

Later, we are going analyze how these different size files are treated in the cluster.

Now, we can check if the objects were stored and then restore them with a different name:

```
vagrant@ceph-client:~$ rados -p datatest ls
test-object-2
test-object-1
vagrant@ceph-client:~$ rados get test-object-1 testfile1.txt --pool=
    datatest
vagrant@ceph-client:~$ ls -sh
total 1.4G
4.0K test1  4.0K testfile1.txt  1.4G testvideo.mkv
```

To remove a stored object, we use `rados rm`:

```
vagrant@ceph-client:~$ rados rm test-object-1 --pool=datatest
```

If we want to remove an entire storage pool, we have to use the `ceph osd pool delete` passing the pool twice with the parameter `-yes-i-really-really-mean-it`, because it's unrecoverable and all the data stored is going to be lost. For exemple, if we want to delete **datatest** pool, we must execute the following command:

```
vagrant@ceph-client:~$ ceph osd pool delete datatest datatest --yes-i-
    really-really-mean-it
pool 'datatest' removed
```

8.2.1.2.5 RBD Block Devices

We are going to create a 4GB block device image using `rbd create` command. RBD is thin provisioned, this does not mean that this image size is 4GB when it is created, it means that is the maximum capacity of this image.

```
vagrant@ceph-client:~$ rbd create BlockDevRBDpool --size 4096
```

By default, this command creates a Block Devices image at the *ceph.admin* user using its keyring and placing it to the **RBD pool**. To check if the image was created successfully, we list the RBD images in our cluster:

```
vagrant@ceph-client:~$ rbd list
BlockDevRBDpool
```

We can check the images available at the RBD pool using `rados ls` passing the rbd pool through parameter `-p`:

```
vagrant@ceph-client:~$ rados -p rbd ls
BlockDevRBDpool.rbd
rbd_directory
```

We need to map this image to our block device. As said previously, if no name is passed as parameter, this command takes *client.admin* as default. In case that we want to map to another client user, we must add `-name client.userID` and its appropriate keyring (`-k /path_to_keyring`) to the following command:

```
vagrant@ceph-client:~$ sudo rbd map BlockDevRBDpool
```

To check if the mapping is made correctly, we must execute the following commands:

```
vagrant@ceph-client:~$ rbd showmapped
id pool image           snap device
1  rbd  BlockDevRBDpool -    /dev/rbd1

vagrant@ceph-client:~$ ls -l /dev/rbd/rbd/
total 0
lrwxrwxrwx 1 root root 10 Jan 15 09:10 BlockDevRBDpool -> ../../rbd1
```

To use the created block device, we must create a XFS file system on our ceph-client node:

```
vagrant@ceph-client:~$ sudo mkfs.xfs /dev/rbd/rbd/BlockDevRBDpool
log stripe unit (4194304 bytes) is too large (maximum is 256KiB)
log stripe unit adjusted to 32KiB
meta-data=/dev/rbd/rbd/BlockDevRBDpool isize=256    agcount=9, agsize
   =130048 blks
=                       sectsz=512   attr=2, projid32bit=0
data     =                       bsize=4096   blocks=1048576, imaxpct
   =25
=                       sunit=1024   swidth=1024 blks
naming   =version 2              bsize=4096   ascii-ci=0
log      =internal log           bsize=4096   blocks=2560, version=2
=                       sectsz=512   sunit=8 blks, lazy-count=1
realtime =none                   extsz=4096   blocks=0, rtextents=0
```

In order to use our file system, we have to mount it:

```
vagrant@ceph-client:~$ sudo mkdir /mnt/AdminBlockDev

vagrant@ceph-client:~$ sudo mount /dev/rbd/rbd/BlockDevRBDpool /mnt/
   AdminBlockDev
```

8.2.1.2.6 Adding users

Now, we can create more user and define different capabilities. We can create *client.user1* with read and write capabilities on *datatest* pool and another used with the ID *client.user2* with just read capabilities.

To create these users, we are going to execute the `ceph auth get-or-create` command. This command creates the user adds the specified capabilities on the monitors and OSDs pools, then creates the secret key which we store using `-o` *`path_to_keyfile`*.

```
vagrant@ceph-admin:~/ceph-cluster$ ceph auth get-or-create client.user1
    mon 'allow r' osd 'allow rw pool=datatest' -o ceph.client.user1.
   keyring

vagrant@ceph-admin:~/ceph-cluster$ ceph auth get-or-create client.user2
    mon 'allow r' osd 'allow r pool=datatest' -o ceph.client.user2.
   keyring
```

In order to push the keyrings to the cluster nodes, we are going to import the created keys to the *ceph.client.admin.keyring* file using `ceph-authtool` that adds the keyrings append the *client.admin* key generated before.

```
vagrant@ceph-admin:~/ceph-cluster$ sudo ceph-authtool ceph.client.admin
    .keyring --import-keyring ceph.client.user1.keyring
importing contents of ceph.client.user1.keyring into ceph.client.admin.
    keyring

vagrant@ceph-admin:~/ceph-cluster$ sudo ceph-authtool ceph.client.admin
    .keyring --import-keyring ceph.client.user2.keyring
importing contents of ceph.client.user2.keyring into ceph.client.admin.
    keyring
```

Now, the *client.admin* keyring file is set to:

```

vagrant@ceph-admin:~/ceph-cluster$ cat ceph.client.admin.keyring
[client.admin]
key = AQBUUHpYGLgVGhAAT+R7X58YvrnrRy8TsVR86Q==
[client.user1]
key = AQD8hXtYmCIJFBAADQTW9jnI40s+5Uv8BADOKg==
[client.user2]
key = AQBxhXtYMJ48IhAA2shvhhsJAvFi43PdMqNkzw==
```

Using `ceph-deploy admin` to push the keyring file to he cluster nodes including the **ceph-admin** itself and storing this key at */etc/ceph* directory, we ensure that all nodes are able to interact with the created users.

```
vagrant@ceph-admin:~/ceph-cluster$ ceph-deploy admin ceph-admin ceph-
    mon0 ceph-mon1 ceph-mon2 ceph-osd1 ceph-osd2 ceph-osd3 ceph-osd4
    ceph-osd5 ceph-osd6 ceph-client
```

We can test the capabilities of the new users executing the `rados put` command passing the user name, the destination pool and the keyring file file. In this case, as it isn't the default user, we must always provide the keyring file path, otherwise it won't work.

```

rados put test-object-user1-1 test1 -n client.user1  --pool=datatest -k
     /etc/ceph/ceph.client.admin.keyring
```

However, if we try to store any object using *client.user2* which has only red capabilities, *rados* returns and error. The same happens if we try to store and object in another pool that isn't the one specified at the users creation, in this case, *datatest* pool.

```
vagrant@ceph-client:~$ rados put test-object-user2-1 test1 -n client.
    user2  --pool=datatest -k /etc/ceph/ceph.client.admin.keyring
error putting datatest/test-object-user2-1: (1) Operation not permitted
```

On the other hand, *user2* van retrieve any object from the *datatest* pool.

```
rados  get test-object-user1-1 get-user2.txt -n client.user1 -k /etc/
    ceph/ceph.client.admin.keyring --pool=datatest
```

we can remove a user executing the following command:

```
vagrant@ceph-admin:~/ceph-cluster$ ceph auth del client.user1
```

8.2.1.2.7 RBD with different user capabilities

In this case, we are going to create an RBD image using user1:

```
vagrant@ceph-client:~$ rbd create BlockDevUser1 --size 4096 -n client.
    user1 -k /etc/ceph/ceph.client.admin.keyring --pool=datatest
```

We map the image to *datatest* pool:

```
vagrant@ceph-client:~$ sudo rbd map BlockDevUser1 --pool=datatest
```

Running the `rbd shiwmapped` command, we can see that we have the imgae mapped correctly:

```

vagrant@ceph-client:~$ rbd showmapped
id pool     image          snap device
1  rbd      BlockDevRBDpool -    /dev/rbd1
2  datatest BlockDevUser1   -    /dev/rbd2
```

We can list the directory where this image is created. And then, we can format the Block Device using `sudo mkfs.ext4` command:

```
vagrant@ceph-client:~$ ls -l /dev/rbd/datatest/
total 0
lrwxrwxrwx 1 root root 10 Jan 15 16:34 BlockDevUser1 -> ../../rbd2

vagrant@ceph-client:~$ sudo mkfs.ext4 /dev/rbd/datatest/BlockDevUser1
mke2fs 1.42.9 (4-Feb-2014)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
Stride=1024 blocks, Stripe width=1024 blocks
262144 inodes, 1048576 blocks
52428 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=1073741824
32 block groups
32768 blocks per group, 32768 fragments per group
8192 inodes per group
Superblock backups stored on blocks:
32768, 98304, 163840, 229376, 294912, 819200, 884736

Allocating group tables: done
Writing inode tables: done
Creating journal (32768 blocks): done
Writing superblocks and filesystem accounting information: done
```

In order to use it, we have to mount it:

```
vagrant@ceph-client:~$ sudo mkdir /mnt/User1BlockDev
vagrant@ceph-client:~$ sudo mount /dev/rbd/datatest/BlockDevUser1 /mnt/
    User1BlockDev/
```

8.2.2 Proving Ceph cluster reliability

We saw that it's easy to interact with the cluster creating new users or just with the default one. We can create block devices, storage pools or use the default ones in order to store our files as object into the cluster. But, can we know where this objects are stored? What happens if a certain OSD or monitor goes down for a while or it's totally removed from the cluster?

CHRUSH algorithm helps us to located the OSDs and Placement groups where a certain object is stored. For Example, using `osd map` we can map **test-object-1** at the *datatest* pool:

```
vagrant@ceph-client:~$ ceph osd map datatest test-object-1
osdmap e41 pool 'datatest' (3) object 'test-object-1' -> pg 3.74dc35e2
    (3.2) -> up ([0,5], p0) acting ([0,5], p0)
```

In this example, this object is stored at *osd.0* (**ceph.osd1**) and *osd.5* (**ceph-oasd6**). This is determinated by its placement group *pg 3.74dc35e* that it's calculated using the object name hashed to the number of placement groups which is *0x74dc35e2*. Then, it adds the pool number, in this case pool datatest is equal 3, as we can see executing the following command:

```
vagrant@ceph-admin:~/ceph-cluster$ ceph osd lspools
0 data,1 metadata,2 rbd,3 datatest,
```

Then, the CRUSH algorithm using this PG, allocates the object OSDs. Now, we can turn down one of the OSDs and see what happens to this object.

Now, we are going to suspend temporally the **ceph-osd1** using `vagrant suspend` passing the VM ID at the host machine:

```
nabil@neak:~$ vagrant suspend 1f04ef3
```

Then, we check the OSDs status to see that osd.0 is down:

```
vagrant@ceph-client:~$ ceph osd tree
# id    weight  type name       up/down reweight
-1      0.24    root default
-2      0.03999         host ceph-osd1
0       0.03999                 osd.0   down    1
-3      0.03999         host ceph-osd2
1       0.03999                 osd.1   up      1
-4      0.03999         host ceph-osd3
2       0.03999                 osd.2   up      1
-5      0.03999         host ceph-osd4
3       0.03999                 osd.3   up      1
-6      0.03999         host ceph-osd5
4       0.03999                 osd.4   up      1
```

```
-7      0.03999         host ceph-osd6
5       0.03999                 osd.5   up      1
```

If we check the osd mapping now, we can see that it was a data re-balancing had occur.

```
vagrant@ceph-client:~$ ceph osd map datatest test-object-1
osdmap e45 pool 'datatest' (3) object 'test-object-1' -> pg 3.74dc35e2
    (3.2) -> up ([5,3], p5) acting ([5,3], p5)
```

Now, the PG is mapped to **OSD.5** acting as the principal one and **osd.3** is acting as the storage replication.

Now, if we resume the machine:

```
nabil@neak:~$ vagrant resume 1f04ef3
```

We will see that the object is re-balanced to its original locations:

```
vagrant@ceph-admin:~/ceph-cluster$ ceph osd map datatest test-object-1
osdmap e47 pool 'datatest' (3) object 'test-object-1' -> pg 3.74dc35e2
    (3.2) -> up ([0,5], p0) acting ([0,5], p0)
```

To ovoid a single point of failure, we set 3 monitors; **ceph-mon0**, **ceph-mon1** and **ceph-mon2**, with the first one acting as the leader. If we suspend the leader, we can analyze what happens to the other ones.

We halt the VM machines using the following command:

```
nabil@neak:~$ vagrant halt 67f0617
```

And then, bring it up again:

```
nabil@neak:~$ vagrant up 67f0617
```

To follow the dialog between the monitors, we are going to watch the live cluster changes using `ceph-w` wich reports the cluster status and configuration:
vagrant halt 67f0617

```
vagrant@ceph-admin:~/ceph-cluster$ ceph -w
cluster 351afd90-5a11-4265-ba6b-139df77ad8f3
health HEALTH_WARN clock skew detected on mon.ceph-mon2 Monitor clock
    skew detected
monmap e3: 3 mons at {ceph-mon0=172.21.12.20:6789/0,ceph-mon1
    =172.21.12.21:6789/0,ceph-mon2=172.21.12.22:6789/0}, election epoch
    30, quorum 0,1,2 ceph-mon0,ceph-mon1,ceph-mon2
osdmap e47: 6 osds: 6 up, 6 in
pgmap v496: 200 pgs, 4 pools, 3065 MB data, 436 objects
46289 MB used, 180 GB / 236 GB avail
200 active+clean

2017-01-05 20:32:55.803134 mon.0 [INF] pgmap v496: 200 pgs: 200 active+
    clean; 3065 MB data, 46289 MB used, 180 GB / 236 GB avail
2017-01-05 20:33:55.827751 7f6b6f7fe700  0 monclient: hunting for new
    mon
2017-01-05 20:33:57.817805 mon.1 [INF] mon.ceph-mon1 calling new
    monitor election
2017-01-05 20:33:57.896890 mon.2 [INF] mon.ceph-mon2 calling new
    monitor election
2017-01-05 20:34:02.821187 mon.1 [INF] mon.ceph-mon1@1 won leader
    election with quorum 1,2
2017-01-05 20:34:02.827384 mon.1 [INF] monmap e3: 3 mons at {ceph-mon0
    =172.21.12.20:6789/0,ceph-mon1=172.21.12.21:6789/0,ceph-mon2
    =172.21.12.22:6789/0}
2017-01-05 20:34:02.827488 mon.1 [INF] pgmap v496: 200 pgs: 200 active+
    clean; 3065 MB data, 46289 MB used, 180 GB / 236 GB avail
2017-01-05 20:34:02.827549 mon.1 [INF] mdsmap e1: 0/0/1 up
2017-01-05 20:34:02.827623 mon.1 [INF] osdmap e47: 6 osds: 6 up, 6 in
2017-01-05 20:41:17.512140 mon.0 [INF] mon.ceph-mon0 calling new
    monitor election
2017-01-05 20:41:17.530765 mon.0 [INF] mon.ceph-mon0 calling new
    monitor election
2017-01-05 20:41:17.558363 mon.0 [INF] mon.ceph-mon0@0 won leader
    election with quorum 0,1,2
2017-01-05 20:41:17.645065 mon.0 [INF] monmap e3: 3 mons at {ceph-mon0
    =172.21.12.20:6789/0,ceph-mon1=172.21.12.21:6789/0,ceph-mon2
    =172.21.12.22:6789/0}
2017-01-05 20:41:17.645150 mon.0 [INF] pgmap v496: 200 pgs: 200 active+
    clean; 3065 MB data, 46289 MB used, 180 GB / 236 GB avail
2017-01-05 20:41:17.645224 mon.0 [INF] mdsmap e1: 0/0/1 up
2017-01-05 20:41:17.645340 mon.0 [INF] osdmap e47: 6 osds: 6 up, 6 in
```

As we can see at the log, when mon.0 goes down, the other monitors call for monitor election, as we set that hierarchy goes from mon.0 to mon.2, the suitable option for the monitor leader is mon1.

Then, when we bring mon.0 up, it calls for another leader election and ends up beening the leader again.

8.2.3 Glossary

RADOS	Reliable Automatic Distirbuted Object Storage
OSD	Object Storage Daemon
RBD	Rados Block Device
RGW	Rados Gateway
CRUSH	Controlled Replication Under Scalable Hashing
PG	Placement Group
FS	File System
VM	virtual Machine