STORAGE

Software-Defined Storage the Ceph Way, Part Two

24 Feb 2015 6:17am, by Pushpesh Sharma

This is part two of a three part series about software-defined storage. Part one covers how to understand SDS. In this post and in part three, we explore Ceph, a popular SDS solution.

Ceph is a full-featured, yet evolving, software-defined storage (SDS) solution. It's very popular because of its robust design and scaling capabilities, and it has a thriving open source community. Ceph provides all data access methods (file, object, block) and appeals to IT administrators with its unified storage approach.

In the true spirit of SDS solutions, Ceph can work with commodity hardware or, to put it other standard protocol and interfaces. Ceph storage clusters are based on Reliable Automatic Distributed Object Store (RADOS), which uses the CRUSH algorithm to stripe, distribute and replicate data. The CRUSH algorithm originated from a PhD thesis by Sage Weil at the University of California, Santa Cruz. Here's an overview of Ceph's different ways for accessing stored data:

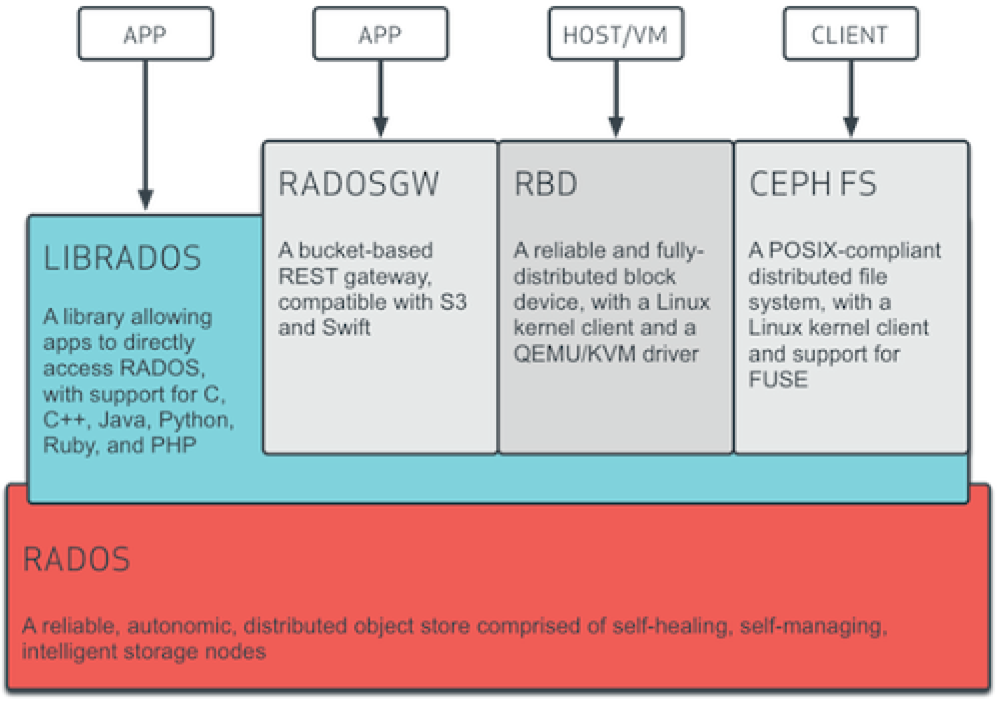

APP
APP
HOST/VM
CLIENT
LIBRADOS
A library allowing apps to directly access RADOS, with support for C, C++, Java, Python, Ruby, and PHP
RADOSGW
A bucket-based REST gateway, compatible with S3 and Swift
RBD
A reliable and fully-distributed block device, with a Linux kernel client and a QEMU/KVM driver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

Figure 1: Ceph Storage Cluster

The Ceph Storage Cluster

A Ceph storage cluster is a heterogeneous group of compute and storage resources (bare metal servers, virtual machines and even Docker instances) often called Ceph nodes, where each member of the cluster is either working as a monitor (MON) or object storage device (OSD). A Ceph storage cluster is used by Ceph clients to store their data directly as RADOS objects or by using virtualized resources like RDBs or other interfaces. There are many components of a Ceph storage cluster which can be summarized as follows:

Ceph Monitor

A Ceph monitor is a daemon running on a Ceph node which holds the latest copy of a cluster map. For now, let's say the cluster map is a global map, which knows the whereabouts of any RADOS object, and these objects are chunks of actual data and are master copy. As this information is very critical, it is recommended to have multiple Ceph monitors running for high availability and cluster robustness.

A Ceph monitor is also responsible for keeping the state of information about other monitors as well as various OSD daemons in a cluster. It is possible, in a highly distributed environment, that various monitors will have different copies of cluster maps, or different states of information about OSD daemons. To keep these monitors in sync and always provide the correct information to clients, the Paxos algorithm is used. The Paxos algorithm works as a consensus building method among various Ceph monitors, where the information available on the majority of running monitors is considered correct. A majority in 2N+1 monitors is formed by N+1 running monitors.Ex. In a cluster having five monitors in total, three running monitors are considered a majority, and a consensus about a cluster state can be built even if the other two monitors are not reachable.

For high availability, it is recommended to have Ceph monitor daemons running on dedicated Ceph Nodes. Ceph monitor daemons do not require very high computational resources, so even VMs can be used to host these daemons in a hyperscale environment; however, these are critical components of a Ceph storage cluster, so all failure scenarios should be kept in mind while choosing hosts or VMs.

Ceph Object Storage Device (Ceph OSD)

Ceph OSD is a daemon running on a Ceph node, responsible for storing RADOS objects. Rather than having its own mechanism of storing data in block devices, block address translation and other related tasks, OSD daemons outsource this task to a subsystem called OSD backend mechanisms. There are various available OSD backend mechanisms like FileStore, KeyValueStore and MemStore. FileStore OSD backend is the most stable one. A FileStore backend makes use of any general purpose file systems that support extended attributes (XATTRs). XFS, Ext4 and Btrfs are available options, however XFS is considered the most stable among them. Each OSD daemon has a corresponding mounted FS on the OSD host. Each RADOS object is stored as a file in the mounted file system. These RADOS objects are arranged in directories based on PGs numbers.

An OSD daemon also talks to other neighboring OSD daemons in clusters, and can then update the Ceph monitor about its state, as well as the state of the other daemons, as seen

Each OSD daemon should be backed by a physical disk or SSD. Additionally, each OSD may use a dedicated or shared journal. A journal is usually backed by a high-speed physical device, like SSD, to improve written performance. A journal is also a mounted file system. Objects written in a journal are then dumped to an appropriate data partition or disk belonging to the OSD.

Meta Data Server (Ceph MDS)

This daemon is used to store CephFS metadata. It is only used when file access to the storage cluster is required. CephFS is not production ready as of now; CephFS and MDS are not part of this write-up.

Placement Group (PG)

A placement group is a group of OSDs, where each OSD in the group has one replica of a RADOS object. The number of OSDs in the group is determined by the replica count. Each PG has a designated primary OSD. A RADOS object first lands in a primary OSD, and is replicated to other secondary OSDs, tertiary OSDs, etc... Each placement group can be depicted as follows:

```
&lt;pool_id&gt;.&lt;pg_id&gt;(primary_osd_num,secondary_osd.....)
```

Ex: 1.09(3,9,600) — here the replica count is 3.

Like monitor and OSD daemons, PGs also have states associated with them. These states are as follows:

Creating

Ceph is still creating the placement group.

Active

Ceph will process requests to the placement group.

Clean

Ceph replicated all objects in the placement group the correct number of times.

Down

A replica with necessary data is down, so the placement group is offline.

Splitting
Ceph is splitting the placement group into multiple placement groups.

Scrubbing
Ceph is checking the placement group for inconsistencies.

Degraded
Ceph has not replicated some objects in the placement group the correct number of times yet.

Inconsistent
Ceph detects inconsistencies in one or more replicas of an object in the placement group (e.g., objects are the wrong size, objects are missing from one replica after recovery is finished.)

Peering
The placement group is undergoing the peering process.

Repair
Ceph is checking the placement group and repairing any inconsistencies it finds (if possible).

Recovering
Ceph is migrating/synchronizing objects and their replicas.

Backfill
Ceph is scanning and synchronizing the entire contents of a placement group instead of inferring what contents need to be synchronized from the logs of recent operations. Backfill is a special case of recovery.

Wait-backfill
The placement group is waiting in line to start backfill.

Backfill-toofull
A backfill operation is waiting because the destination OSD is over its full ratio.

failed OSDs that may contain the needed information, or temporarily adjust min_size to allow recovery.

Stale

The placement group is in an unknown state — the monitors have not received an update for it since the placement group mapping changed.

Remapped

The placement group is temporarily mapped to a different set of OSDs from what CRUSH specified.

Undersized

The placement group has fewer copies than the configured pool replication level.

Peered

The placement group has peered, but cannot serve client IO due to not having enough copies to reach the pool's configured min_size parameter. Recovery may occur in this state, so the pg may heal up to min_size eventually.

Some of the above states are obvious, some of them not so obvious. It is advisable to revisit these states once the reader completes reading other sections. For now, it is sufficient to know the optimum state for placement groups is active+clean.

Pool

A pool is a set of PGs among which RADOS objects are distributed. A pool can also be seen as a virtualized storage partition that a ceph client may use to hold their data. A pool may hold many RBDs (Remote Block Devices), S3/Swift Objects (RADOS Gateway Interface) and files and metadata (CephFS). There are four fundamental attributes of a pool:

1. **PG Num:** The total number of placement groups in the pool.
2. **PGP Num:** The total number of placement groups for placement purposes, or distributing the pool data. This should be equal to the total number of placement groups, except for placement group splitting scenarios.
3. **Pool Size:** Pool size is the number of times an object will be replicated. This determines the number of OSDs in a PG set.

available/UP OSDs goes below this number, then the PG can only be used after recovery of the OSDs.

The following diagram depicts the relationship between OSDs, PGs and Pool:

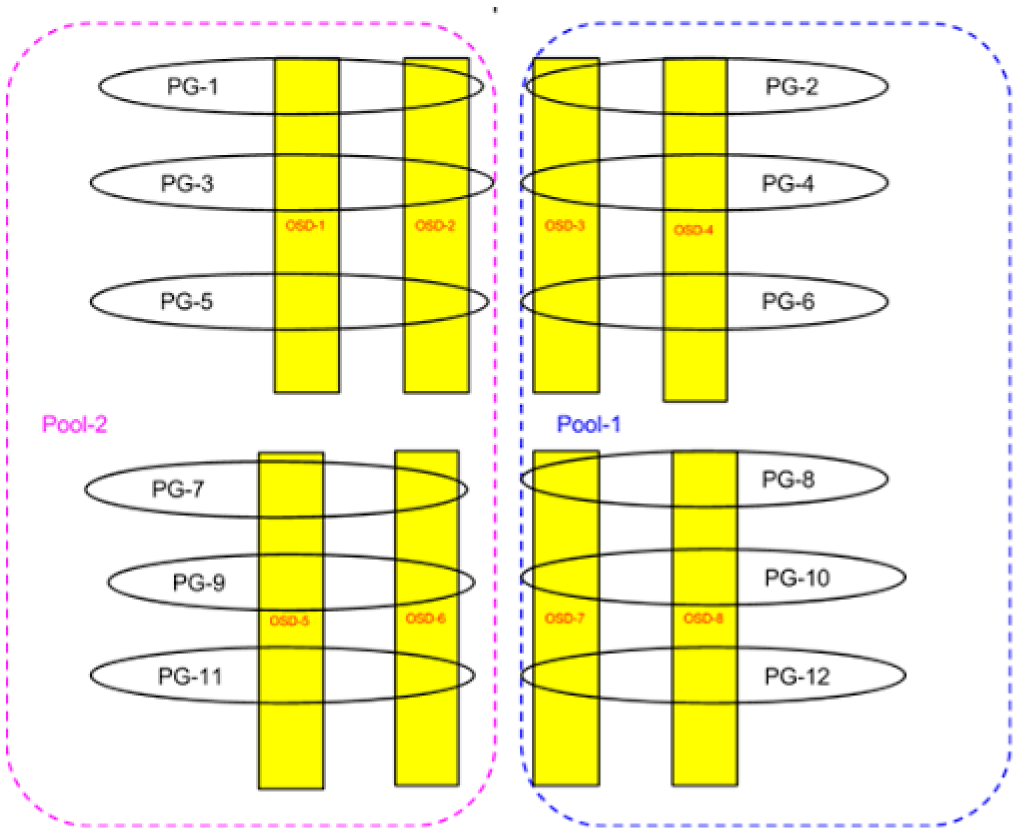


Figure 2: Ceph OSDs, PGs and Pools (Replica count=2)

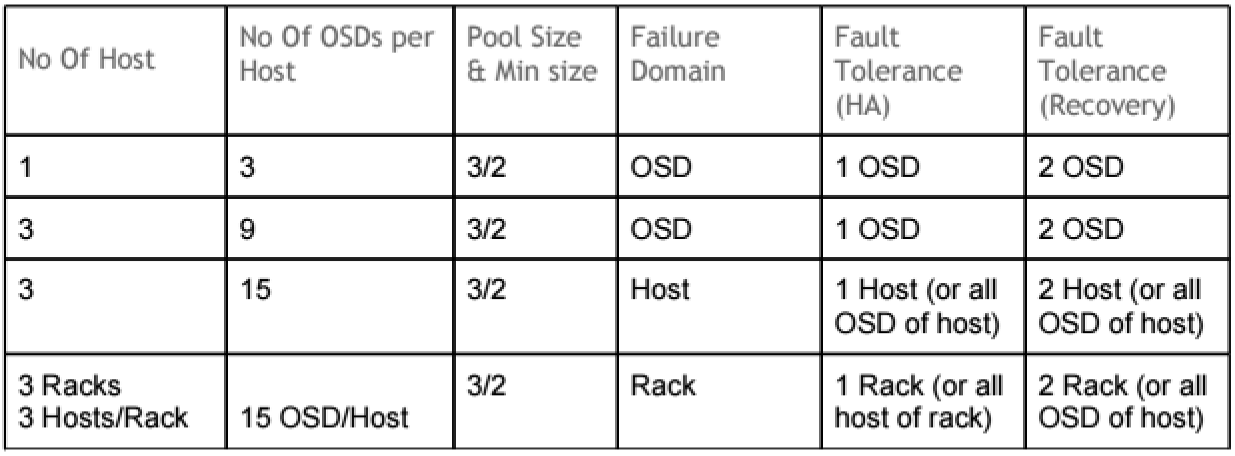
CRUSH Rule and Cluster Map

Any distributed storage system is based on distributing the data among many storage devices, thereby gaining from the parallel processing in this pool of resources. As data integrity and reliability are very essential requirements of any storage system, there should be some fault-tolerance mechanism to assure the availability and reliability of data in case of hardware failure. Data replication is one of the ways to provide fault-tolerance to storage systems. In Ceph storage systems, rules governing this replication and distribution are called CRUSH rules. CRUSH rules work as the backbone for providing enterprise-class reliability with commodity-class hardware, as it considers failure the norm in clusters, rather than the exception. CRUSH defines cluster reliability in terms of failure domains.

THENEWSTACK Ebooks Podcasts Events Newsletter

Architecture Development Operations

customization based on data-center design and reliability requirements. By default, the failure domain is set to host, and pool size and min-size is set to 3 and 2 respectively. A few examples of different crush maps and cluster configurations are explained below:



No Of Host	No Of OSDs per Host	Pool Size & Min size	Failure Domain	Fault Tolerance (HA)	Fault Tolerance (Recovery)
1	3	3/2	OSD	1 OSD	2 OSD
3	9	3/2	OSD	1 OSD	2 OSD
3	15	3/2	Host	1 Host (or all OSD of host)	2 Host (or all OSD of host)
3 Racks 3 Hosts/Rack	15 OSD/Host	3/2	Rack	1 Rack (or all host of rack)	2 Rack (or all OSD of host)

For participating in the task of data management and lookup, both the Ceph client and Ceph OSD should be aware of data distribution and replication scheme. This data distribution and replication scheme, or cluster topology, is called a cluster map. Earlier approaches of distributed storage systems were based on some centralized daemon to know the data whereabouts and act as a broker between clients and servers. This approach created bottlenecks and single points of failure. On the contrary, Ceph decentralizes the task of efficiently computing information about object location. It makes use of the computation powers of Ceph client nodes as well as Ceph OSD nodes, where both contain a copy of the cluster map and establish a direct connection. A cluster map is not a single entity but a set of the following different maps:

1. **The monitor map**: contains the cluster fsid, the position, name IP address and port of each monitor. It also indicates the current epoch, when the map was created, and the last time it changed.
2. **The OSD map**: contains the cluster fsid, when the map was created and last modified, a list of pools, replica sizes, PG numbers, a list of OSDs and their status (e.g., up, in.)
3. **The PG map**: contains the PG version; its timestamp; the last OSD map epoch; the full ratios and details on each placement group such as the PG ID, the Up Set, the Acting Set, the state of the PG (e.g., active+clean); and data usage statistics for each

THENEWSTACK Ebooks Podcasts Events Newsletter

Architecture Development Operations

4. **The CRUSH map**: contains a list of storage devices, the failure domain hierarchy (e.g., device, host, rack, row, room), and rules for traversing the hierarchy when storing data.
5. **The MDS map** (only used in CephFS): Contains the current MDS map epoch, when the map was created, and the last time it was changed. It also contains the pool for storing metadata, a list of metadata servers, and which metadata servers are up and in.

A Ceph client knows about all the monitors and OSDs in the cluster. It computes object location using a cluster map; it only requires Pool-ID and object-ID as input. Based on this computation, it knows which OSDs to talk to for getting or placing a particular object. It reads and writes only to the primary OSD. In the case of write, the primary OSD takes care of replicating it to other OSDs in the PG set.

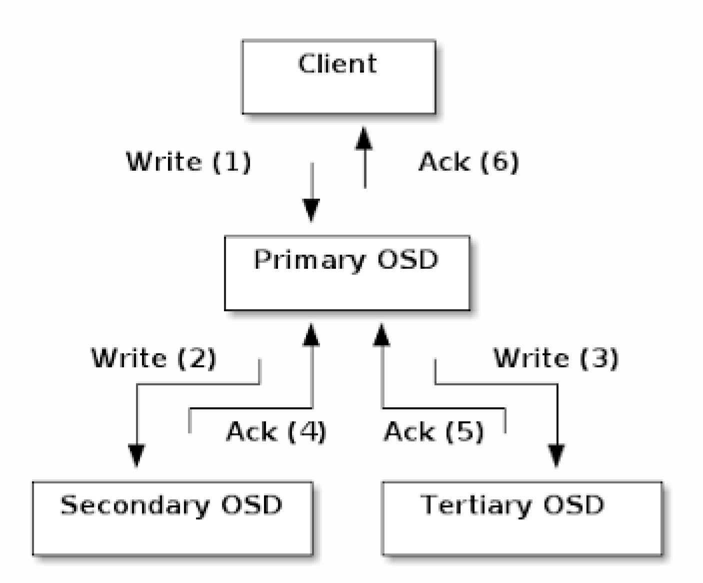


Figure 3: Ceph Write Operation

Ceph Authentication (CephX)

Ceph also provide a Kerberos-like authentication system called CephX. A Ceph client is required to register itself with any monitor in the cluster and obtain a shared secret. Each OSD daemon is also required to register itself with any monitor and obtain a shared secret. These shared secrets are copied across all monitor in the cluster. When a client wishes to communicate to any OSD, it first obtains an encrypted session key from the

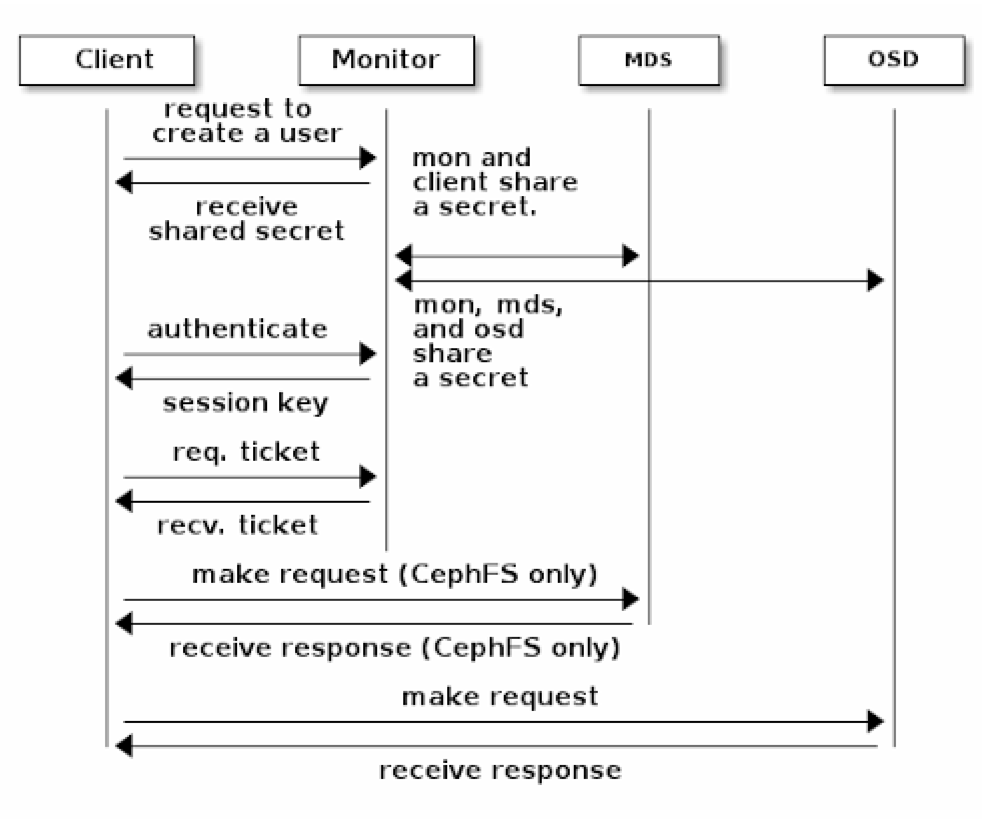


Figure 4: CephX Communication

Deploying a Simple Ceph Storage Cluster

It is recommended to go through the official documentation for deploying any production cluster. The purpose of putting these steps here is for illustrative purposes only:

Hardware available

- **Four VMs**: running Ubuntu 14.04.
- **Three bare metal machines**: each have Ubuntu 14.04.
- **Three physical disks:** 1 TB each.
- **One SSD:** 400 GB.

Preconditioning

Each node should have a hostname and each node should be able to resolve the hostnames of other nodes. Let us assume these nodes are named as follows:

- **Four VMs**: admin-node, mon1, mon2 and mon3.

The admin-node has password-less SSH access to all the other nodes. Fix the firewall setting and check the required ports to be opened in the official documentation. Make sure each Ceph node is in time-sync with the other. The popular method of deploying a Ceph cluster is by using Ceph-deploy. This package should be installed on the admin-node. All the following commands are executed from admin-node:

Install Ceph

Add the latest stable ceph repository on admin-node. Refer to the official documentation.

```
admin-node# ceph-deploy install mon-1 mon-2 mon-3 osd-node-1 osd-node-2 osd-node-3
```

Create Cluster

Create a cluster named 'ceph', having mon-1, mon-2 and mon-3 as initial members of the cluster monitor quorum.

```
admin-node# ceph-deploy new mon-1 mon-2 mon-3
```

Add Monitor

```
admin-node# ceph-deploy mon create mon-1 mon-2 mon-3
```

Gather Keys

Get all the keys created by the monitor from one of the monitors.

```
admin-node# ceph-deploy gather-keys mon-1
```

List Disks

Look out for all available disks on each OSD node.

```
admin-node# ceph-deploy disk list osd-node-1 osd-node-2 osd-node-3
```

This will be displaying four disks available with each host. The SSD listed in each host will act as the journal. Considering /dev/sde as the SSD, it can be partitioned into three parts, with each partition for a different OSD.

Format Disks

```
admin-node# for i in 1 2 3
do
```

```
done
done
```

Create OSDs

```
admin-node# for i in 1 2 3
do
for dev in sdb sdc sdd
do
ceph-deploy disk zap osd-node$i:$dev
ceph-deploy osd create osd-node$i:$dev:/dev/sde$i
done
done
```

Check Health

If all of the above steps are completed successfully, you should be having an up and running Ceph storage cluster. You can check the status of the cluster as follows:

From any Ceph Node (Mon-nodes or OSD-nodes)

```
#ceph -s
```

You should see something like below:

```
cluster b0354daa-11ae-466e-ba51-99f3fe656978
health HEALTH_OK
monmap e1: 3 mons at {mon-1=10.242.43.96:6789/0,mon-2=10.242.43.98:6789/0,mon-3=10.242.43.100:6
202, quorum 0,1,2 mon-1,mon-2,mon-3
osdmap e4210: 9 osds: 9 up, 9 in
pgmap v174144: 64 pgs, 1 pools, 0 data, 0 objects
100 MB used, 9000 GB / 9000 GB avail
64 active+clean
```

Create a Ceph Pool

From any Ceph node:

```
#ceph osd pool create mypool 512 512
```

It creates a Ceph pool named 'mypool' with a PG count of 512, followed by pgp_num, and a default replica count (pool size) of 3. Please refer to the official documentation for deciding upon the number of PGs. The general formula for deciding PG_num is as follows:

THENEWSTACK Ebooks Podcasts Events Newsletter

Architecture Development Operations

After successful creation of a pool, you should be able to see 512 PGs in the active+clean state.

RADOS Object Operations

```
#rados -p mypool put objectone infile1.txt
#rados -p mypool get objectone outfile2.txt
```

See the object created in a Ceph cluster as follows:

```
#rados -p mypool ls
```

Pushpesh Sharma is currently working as a senior test development engineer at SanDisk India Device Design Center in Bangalore. He has over six years experience in evaluating cloud, virtualization and storage technologies. He holds a Bachelors degree in Engineering (Information Technology) from Government Engineering College Kota (Raj.), India. He also holds a Certificate in Marketing and HRM from SJMSOM, IIT, Bombay. In his free time he likes to read (anything, mostly) and listen to good music, and he enjoys good food and wine.

Featured image via Flickr Creative Commons.

TUTORIAL

+

THE NEW STACK UPDATE

A newsletter digest of the week's most important stories & analyses.

Email Address

Subscribe

We don't sell or share your email. By continuing, you agree to our Terms of Use.

THENEWSTACK Ebooks Podcasts Events Newsletter

Architecture Development Operations

CLOUD NATIVE / KUBERNETES / STORAGE

Tutorial: Install and Configure Portworx on a Bare-Metal Kubernetes Cluster

3 Apr 2020 12:03pm, by Janakiram MSV

CONTAINERS / STORAGE

Tutorial: Create a Docker Swarm with Persistent Storage Using GlusterFS

2 Apr 2020 12:00pm, by Jack Wallen

View / Add Comments

Please stay on topic and be respectful of others. Review our Terms of Use.

SPONSORED FEED

THENEWSTACK Ebooks Podcasts Events Newsletter

Architecture Development Operations

PRISMA BY PALO ALTO NETWORKS

Best Practices for Video Conferencing Security

APRIL 04, 2020

SAP

UI5 i18n Plugin for SAP Business Application Studio (Open Source)

APRIL 04, 2020

redislabs HOME OF REDIS

Redis University Announces Self-Paced On-Demand Courses

APRIL 03, 2020

New Relic.

Connected: How New Relic Employees Are Working as One During Difficult Times

APRIL 03, 2020

dynatrace

Further improvements to OneAgent security for Windows and Linux

APRIL 03, 2020

CLOUD NATIVE COMPUTING FOUNDATION

We're all in this Together: A Wellness Guide from the CNCF Well-Being Working Group

APRIL 03, 2020

NS1.

4 Steps to making sure your VPN access remains seamless

APRIL 03, 2020

THENEWSTACK Ebooks Podcasts Events Newsletter

Architecture Development Operations

APRIL 03, 2020

Cloud Foundry Lends Itself to Volkswagen Financial's New CI/CD Program
APRIL 03, 2020

Comparing npm Audit Versus AuditJS
APRIL 03, 2020

Understand incident management with GitLab
APRIL 02, 2020

Remote Work Tips: Create Test Automation From Afar With ARA
APRIL 02, 2020

Simplified Time-Series Analysis with Amazon CloudWatch Contributor Insights
APRIL 02, 2020

Calling all Innovators! Nominations are Open for the 2020 MongoDB Innovation Awards
APRIL 02, 2020

Continuous integration and deployment for Android apps with fastlane
APRIL 02, 2020

Ebooks Podcasts Events Newsletter

Architecture Development Operations

New to working from home? Here's how to make remote work work.
APRIL 02, 2020

TRIGGERMESH CLOUD NATIVE INTEGRATION PLATFORM

Working at Home Tips from the TriggerMesh Team
APRIL 02, 2020

HAPROXY

HAProxy 1.8+ HTTP/2 HPACK Decoder Vulnerability Fixed
APRIL 02, 2020

HashiCorp

HashiCorp Nomad Remote Exec Web UI
APRIL 01, 2020

CAPSULE8

Goal Oriented in Soccer and for Customers: Capsulator Austin Britt
APRIL 01, 2020

CiTRIX

Citrix ADC for Kubernetes: Scaling DNS with CPX as node local cache
APRIL 01, 2020

SALTSTACK

How Can IT Automation Help During The Covid-19 Crisis?
MARCH 31, 2020

THE LINUX FOUNDATION

MERA, Mocana, and Osaka NDS Join Automotive Grade Linux

THENEWSTACK Ebooks Podcasts Events Newsletter

Architecture Development Operations

The art of shipping and monitoring software with speed and confidence
MARCH 31, 2020

DIAMANTI

Diamanti Spektra 2.4 is Purpose-Built for Today's Applications
MARCH 31, 2020

ASPEN MESH

Digital Transformation: How Service Mesh Can Help
MARCH 26, 2020

logdna

Announcing the General Availability of Extract and Aggregate fields
MARCH 26, 2020

Lightbend

Cloudstate - Towards Stateful Serverless
MARCH 26, 2020

portworx

Essential Capabilities for Kubernetes Backup and Recovery
MARCH 25, 2020

SENTRY

Customer Story: How Aumni Keeps Errors out of "Userland"
MARCH 23, 2020

cloudbees

Upskilling 2020: Adapting Humans at The Speed of DevOps

Managing Kubernetes at enterprise scale: A closer look at Tanzu Mission Control
MARCH 19, 2020

snyk

AngularJS Security Fundamentals
MARCH 19, 2020

mongoDB.

Getting storage engines ready for fast storage devices
MARCH 16, 2020

packet

Building a Career in Tech
JANUARY 16, 2020

ARCHITECTURE

Cloud Native

Containers

Edge/IoT

Microservices

Networking

Serverless

Storage

DEVELOPMENT

Cloud Services

Data

Development

Machine Learning

THENEWSTACK Ebooks Podcasts Events Newsletter

Architecture Development Operations

CI/CD

Culture

DevOps

Kubernetes

Monitoring

Service Mesh

Tools

THE NEW STACK

Ebooks

Podcasts

Events

Newsletter

About / Contact

Sponsors

Disclosures

Contributions

© 2020 The New Stack. All rights reserved.

Privacy Policy. Terms of Use.