目錄

Ceph实践

主要内容

- Ceph运维实践,监控及日志管理
- Ceph的原理
- Ceph常见异常与解决
- Ceph压力测试,性能分析和优化

About me

本人运维开发一枚,专注于IaaS及Paas相关技术,欢迎大家拍砖,
frank6866@vip.sina.com

Ceph常见概念

存储相关概念

分布式存储

分布式存储按照用途，可以分为三类:

- 块存储: 提供块设备，常见的块存储是Ceph RBD，可以给虚拟机提供硬盘
- 对象存储: 提供文件的存储功能，常见的对象存储有S3、Ceph RGW和Swift
- 分布式文件系统: 提供文件系统，比如cephfs、glusterfs

存储基础知识

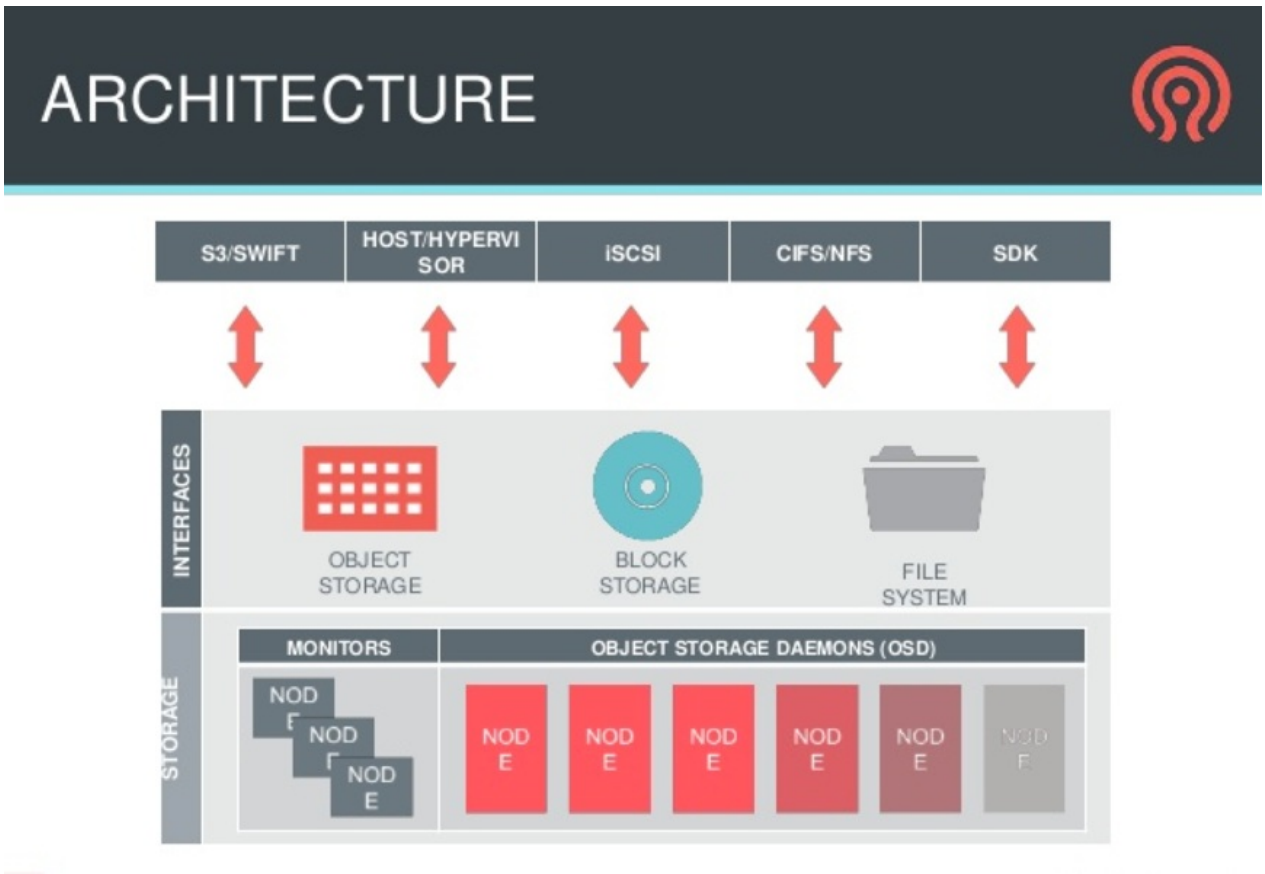
- SAN的全称是Storage Area Network，SAN是一种架构，SAN把外部存储设备和服务器连接起来。
- S3的全称是Simple Storage Service，是Amazon公司的对象存储类型的产品。
- Swift是OpenStack项目的对象存储类型的模块，Swift可以单独使用。
- iSCSI的全称是Internet Small Computer System Interface，用于将SCSI数据块映射成以太网数据包。它提供了在TCP/IP网络上的主机和存储设备传输数据的通道。
- SCSI的全称是Small Computer System Interface，是块数据传输传输协议，在存储行业广泛应用。

Ceph简介

Ceph是一个PB级的存储解决方案,提供如下存储服务:

- 对象存储(RGW)
- 块存储(RBD)
- 文件系统(cephfs)

1.Ceph两层架构

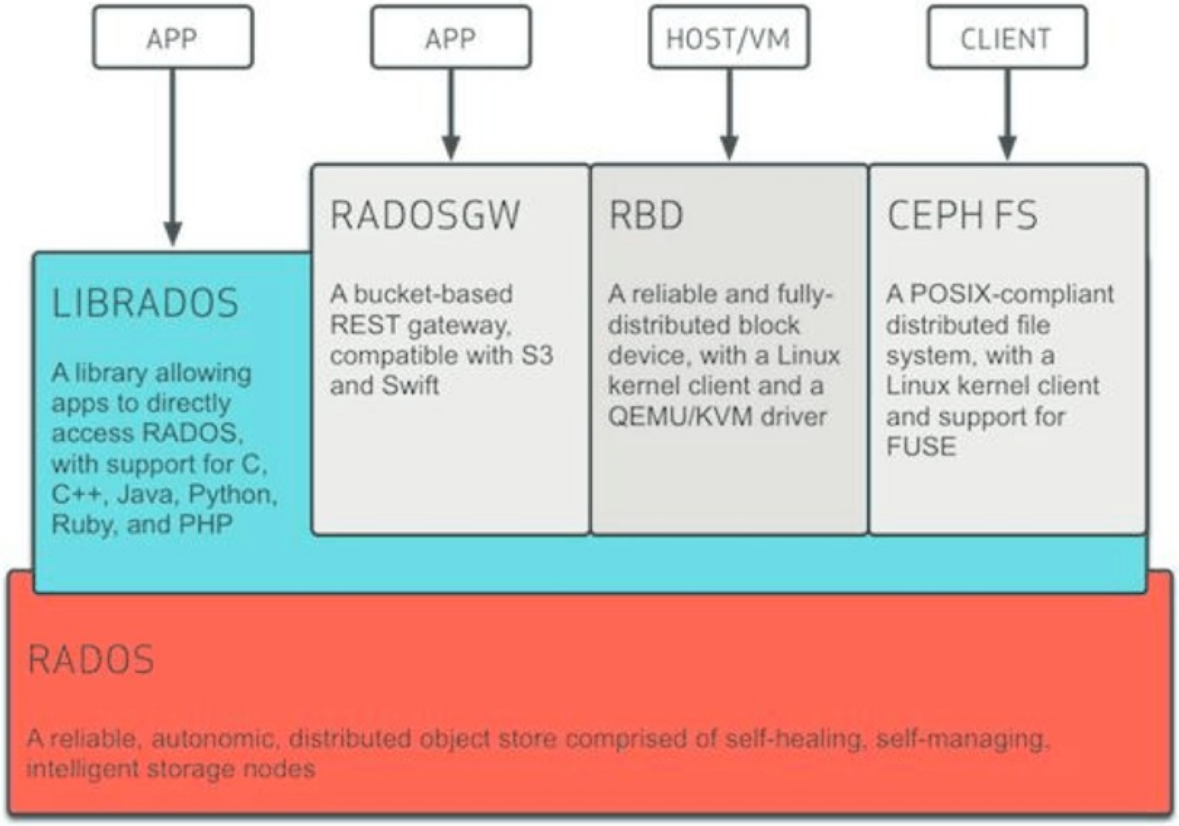


Ceph包含两层：

- 存储层： 存储层的主要功能是管理物理存储，包括Monitors和OSD。
- 接口层： 接口层主要功能是对外提供服务，包括对象存储、块存储和文件系统三种类型的接口。

2.接口层架构

Ceph的逻辑架构图如下:

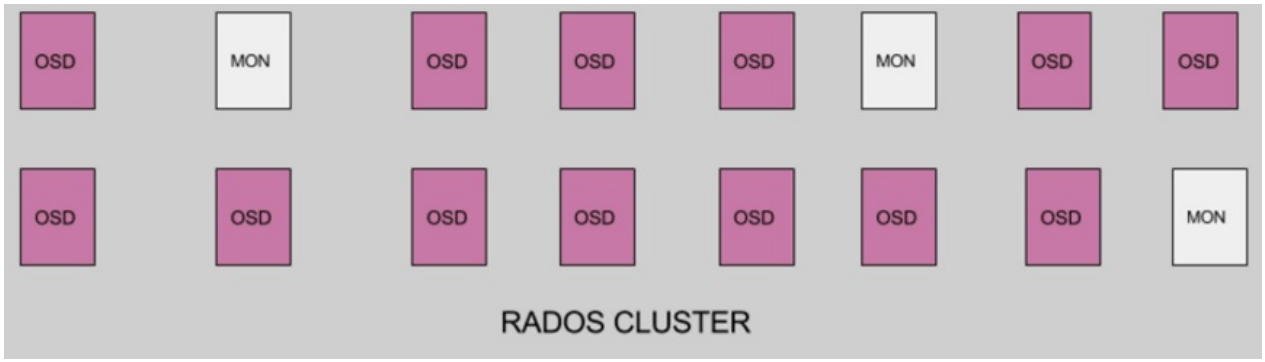


Ceph包含以下组件:

- RADOS: RADOS的全称是Reliable Autonomic Distributed Object Storage(可靠的自主分布式对象存储)，RADOS是一个完整的对象存储服务(这里的对象存储和S3或者Swift不是一个概念)，RADOS由大量的存储设备节点组成。
- LibRADOS: LibRADOS是一个Library，提供了访问RADOS的功能；RADOSGW、RBD、CephFS都会调用LibRADOS来访问RADOS。
- RBD: RBD的全称是RADOS Block Device，提供块存储。一般提供给机器作为磁盘。
- RGW(RadosGW): RGW(RadosGW)的全称是Radow Gateway，提供对象存储服务(类似于S3和Swift)。
- CephFS: CephFS是一个文件系统。

3.存储层(RADOS)架构

RADOS负责Ceph所有的功能，包含MON和OSD,其架构如下:



- MON：Monitor节点
- OSD：OSD的全称是Object Storage Daemon，一般对应一块物理硬盘(建议对应到硬盘)，也可以对应一个分区。

osd/pool/pg

pool是存储在ceph中对象的逻辑组，pool由pg(placement group)组成。

在创建pool的时候，我们需要提供pool中pg的数量和对象的副本数。

pool会将某个对象按照副本数分布在多个osd上，保证数据的可靠性。

- ceph集群可以有多个pool
- 每个pool有多个pg，pg数量在pool创建的时候指定好
- 一个pg包含多个对象
- pg分布在多个osd上面。第一个映射的osd是primary osd，后面的是secondary osd
- 很多pg映射到一个osd

pg

pg(placement group)对客户单来说是不可见的，但在ceph集群中扮演着一个非常重要的角色；pg是Ceph执行re-balance时的基本单位。

为什么要引入pg这个概念？
为了达到EB级别的存储容量，Ceph集群可能需要数千个OSD。Ceph客户端将对象存储在pool中，pool是集群中对象逻辑的集合，OSD是ceph中物理的集合，一般一个OSD对应一块物理磁盘；pool并不属于某个特定的OSD。存储在某个pool中的对象的个数很容易达到百万甚至千万的级别。对于一个系统来说，如果一个pool就有百万甚至千万的对象，这个系统很难追踪或维护所有对象的状态并且系统的性能还很好。在Ceph中，将对象归类到PG中，然后将PG映射到OSD上，来保证高效平衡每个OSD的容量。

在对象数量比较多时，跟踪**pool**中每一个对象的位置开销很大。为了高效地跟踪对象的位置，Ceph将一个pool分为多个PG；先将对象分到PG中，然后再将PG放到primary OSD上。如果一个OSD挂了，Ceph会将存放在这个OSD上的所有PG移动或复制到别的地方

Ceph故障域

Ceph中故障域表示任何一个或者多个不能访问OSD的故障范围,包括如下故障域:

- OSD级别
- 主机级别
- 机架级别
- 电源级别
- 机房级别
- 数据中心级别

在规划的时候,需要考虑ceph的故障域,在成本和性能之间取得平衡。

Ceph常用端口

- tcp/6789: monitor之间进6789端口行通信
- tcp/6800-7300: osd进程会在这个范围内使用可用的端口号
- tcp/7480: rgw的endpoint端口

weight和reweight

在ceph osd df命令中，有weight和reweight两个字段:

```
# ceph osd df
ID WEIGHT  REWEIGHT SIZE  USE    AVAIL %USE  VAR  PGS
 3 0.86909  1.00000  889G 78631M  813G  8.63 0.99 329
 4 0.86909  0.98000  889G 81207M  810G  8.91 1.02 333
 5 0.86909  1.00000  889G 78747M  813G  8.64 0.99 346
......
```

weight和磁盘容量有关系，比如1T磁盘的weight值是1，只和磁盘容量有关系，不会因为磁盘可用空间减少而减少。

reweight的目的是，当集群规模较小时，CRUSH算法的均衡性可能不会很好，可以通过设置osd的reweight来达到数据均衡的目的。

reweight是一个0-1之间的值，命令格式如下:

```
# osd reweight osd-id <float[0.0-1.0]>
```

比如，将id为1的osd的reweight设置为0.9

```
# ceph osd reweight 1 0.9
```

ceph存储对象的过程

往ceph中上传一个对象

```
# echo 'I am going to be stored in ceph' > data.txt
# rados -p pool-frank6866 put object-data data.txt
```

列出pool中的对象

```
# rados -p pool-frank6866 ls
object-data
```

查看pool中的对象所在的osd

```
# ceph osd map pool-frank6866 object-data
osdmap e42 pool 'pool-frank6866' (1) object 'object-data' -> pg
1.c9cf1b74 (1.74) -> up ([2,0,1], p2) acting ([2,0,1], p2)
```

- osdmap e42: 表示osdmap的版本是42
- pool 'pool-frank6866': 表示pool的名称是pool-frank6866
- (1): 表示pool的id是1
- object 'object-data': 表示对象名是object-data
- pg 1.c9cf1b74 (1.74): 表示对象所属pg的id是1.74,c9cf1b74表示的是对象的id
- up ([2,0,1], p2): 这里副本数设置的是3,up表示该对象所在的osd的id

查找id为2的osd所在的主机

```
# ceph osd find 2
{
    "osd": 2,
    "ip": "10.10.10.77:6800\/20546",
    "crush_location": {
        "host": "ceph-3",
        "root": "default"
    }
}
```

登录osd所在主机上查看挂载的目录信息:

```
# df -lTh /var/lib/ceph/osd/ceph-2
Filesystem     Type  Size  Used Avail Use% Mounted on
/dev/sdb1      xfs    45G   68M   45G   1% /var/lib/ceph/osd/cep
h-2
```

根据pg id查看该pg存放数据的地方:

```
# ls -al /var/lib/ceph/osd/ceph-2/current | grep 1.74
drwxr-xr-x   2 ceph ceph    67 Jun  3 17:20 1.74_head
drwxr-xr-x   2 ceph ceph     6 Jun  3 16:53 1.74_TEMP
```

查看pg所在目录的结构:

```
# tree /var/lib/ceph/osd/ceph-2/current/1.74_head/
/var/lib/ceph/osd/ceph-2/current/1.74_head/
├── __head_00000074__1
└── object-data__head_C9CF1B74__1
```

查看文件的内容:

```
# cat object-data__head_C9CF1B74__1
I am going to be stored in ceph
```

可以发现,和data.txt文件的内容是一样的。

CRUSH

CRUSH(Controlled Replication Under Scalable Hashing)算法通过计算数据存储的位置决定如何存储和获取数据。

CRUSH算法可以让客户端直接和OSD进行通信，而不用通过一个中心节点或者代理，从架构上避免了单点故障，避免了性能瓶颈，避免了容量限制。

1. Ceph首先计算数据data的hash值，用hash值对pg数进行取余运算，得到数据data的pg编号。
2. 通过CRUSH算法将pg映射到一组OSD中
3. 把数据data存放到pg对应的OSD中

这个过程包含了两次映射：将数据映射到PG，将PG映射到OSD。

CRUSH算法的目的是给PG分配一组存储数据的OSD。在选择OSD的过程中，需要考虑：

- PG在OSD上均衡地分布。根据OSD的权重，处理相应的PG；权重越大的OSD，存储更过的PG
- PG所在的OSD在不同的故障域上。

pg到osd的映射存储在Monitor节点中。

```
# ceph osd crush dump
{
    "devices": [
        {
            "id": 0,
            "name": "device0"
        },
        {
            "id": 1,
            "name": "osd.1"
        },
        {
            "id": 2,
            "name": "device2"
        },
```

```
        {
            "id": 3,
            "name": "osd.3"
        },
......
    ],
    "types": [
        {
            "type_id": 0,
            "name": "osd"
        },
        {
            "type_id": 1,
            "name": "host"
        },
        {
            "type_id": 2,
            "name": "chassis"
        },
        {
            "type_id": 3,
            "name": "rack"
        },
        {
            "type_id": 4,
            "name": "row"
        },
        {
            "type_id": 5,
            "name": "pdu"
        },
        {
            "type_id": 6,
            "name": "pod"
        },
        {
            "type_id": 7,
            "name": "room"
        },
        {
```

```
            "type_id": 8,
            "name": "datacenter"
        },
        {
            "type_id": 9,
            "name": "region"
        },
        {
            "type_id": 10,
            "name": "root"
        }
    ],
    "buckets": [
        {
            "id": -1,
            "name": "default",
            "type_id": 10,
            "type_name": "root",
            "weight": 572558,
            "alg": "straw",
            "hash": "rjenkins1",
            "items": [
                {
                    "id": -2,
                    "weight": 230384,
                    "pos": 0
                },
                {
                    "id": -3,
                    "weight": 171087,
                    "pos": 1
                },
                {
                    "id": -4,
                    "weight": 171087,
                    "pos": 2
                }
            ]
        },
        {
```

```
            "id": -2,
            "name": "ceph-1",
            "type_id": 1,
            "type_name": "host",
            "weight": 230384,
            "alg": "straw",
            "hash": "rjenkins1",
            "items": [
                {
                    "id": 3,
                    "weight": 56957,
                    "pos": 0
                },
                {
                    "id": 4,
                    "weight": 56957,
                    "pos": 1
                },
                {
                    "id": 5,
                    "weight": 56957,
                    "pos": 2
                },
                {
                    "id": 1,
                    "weight": 59513,
                    "pos": 3
                }
            ]
        },
......
    ],
    "rules": [
        {
            "rule_id": 0,
            "rule_name": "replicated_ruleset",
            "ruleset": 0,
            "type": 1,
            "min_size": 1,
            "max_size": 10,
```

```
            "steps": [
                {
                    "op": "take",
                    "item": -1,
                    "item_name": "default"
                },
                {
                    "op": "chooseleaf_firstn",
                    "num": 0,
                    "type": "host"
                },
                {
                    "op": "emit"
                }
            ]
        }
    ],
    "tunables": {
        "choose_local_tries": 0,
        "choose_local_fallback_tries": 0,
        "choose_total_tries": 50,
        "chooseleaf_descend_once": 1,
        "chooseleaf_vary_r": 1,
        "chooseleaf_stable": 0,
        "straw_calc_version": 1,
        "allowed_bucket_algs": 22,
        "profile": "firefly",
        "optimal_tunables": 0,
        "legacy_tunables": 0,
        "minimum_required_version": "firefly",
        "require_feature_tunables": 1,
        "require_feature_tunables2": 1,
        "has_v2_rules": 0,
        "require_feature_tunables3": 1,
        "has_v3_rules": 0,
        "has_v4_buckets": 0,
        "require_feature_tunables5": 0,
        "has_v5_rules": 0
    }
}
```

CRUSH map主要包括四个部分:

- Devices: 表示集群中当前或者历史的OSD,如果id对应的OSD在集群中还存在,那么device的名称就是osd的名称,比如osd.1;如果device id对应的osd在集群中不存在了,那么device name的名称是device+id,比如device0
- Bucket Types: Bucket定义了CRUSH层级结构中Bucket的类型。比如rows, racks, chassis, hosts等
- Bucket instances: 对集群中的资源,定义它属于某个类型的bucket。比如物理机的bucket type为host
- Rules: 定义了选择bucket的规则

常见的type如下:

```
# types
type 0 osd
type 1 host
type 2 chassis
type 3 rack
type 4 row
type 5 pdu
type 6 pod
type 7 room
type 8 datacenter
type 9 region
type 10 root
```

CRUSH map包括

- OSD的列表
- bucket的列表,bucket将设备聚合到物理位置
- rule的列表,rule概述CRUSH算法如何将pool中的数据副本分布在集群中

通过映射底层的物理结构,CRUSH算法可以解决可能的关联设备失效,比如共享电源或网络导致的失效。将这些可能导致失效的信息存储到cluster map中,CRUSH放置副本的策略可以将对象的副本存储在不同的故障域中。

CRUSH可以帮助我们快速定位问题,比如一个OSD宕掉了,我们可以检查这个OSD所在的机房、机柜、机架等信息。

OSD在CRUSH中的层级结构位置被称为"cursh location", 比如

```
root=default row=a rack=a2 chassis=a2a host=a2a1
```

key表示的是CRUSH type, 并不是所有的key都需要设置。

ceph-crush-location工具可以找出某个OSD在CRUSH map中的位置

```
# ceph-crush-location --id 0 --type osd
host=ceph-1 root=default
```

map

Ceph Monitor通过一系列的map来跟踪整个集群的健康状态:

- mon map
- osd map
- pg map
- crush map

所有集群节点都向monitor节点报告状态，并分享每一个状态变化的信息。monitor为每个组件维护一个独立的map，monitor不存储实际数据，实际数据存储在osd中。

对于任何读或者写操作，客户端首先向monitor请求集群的map，之后就可以无需monitor干预直接与osd进行i/o操作。

monitor map

monitor map维护着monitor结点间端到端的信息，包括:

- 集群id
- monitor主机名
- monitor ip地址
- monitor端口号
- mon map的创建和最后一次修改时间

```
$ ceph mon dump
dumped monmap epoch 1
epoch 1
fsid b64cae0f-99fd-4ade-b07d-cb0b07145f3b
last_changed xxx
created xxx
0: 10.10.10.11:6789/0 mon.ceph-1
1: 10.10.10.12:6789/0 mon.ceph-2
2: 10.10.10.13:6789/0 mon.ceph-3
```

osd map

osd map包含如下信息:

- 集群id
- osd map的创建和最后一次修改时间
- 集群中pool相关的信息，比如副本数、pg数量等
- osd的相关信息，比如osd的id、状态、权重、osd所在主机的ip与端口信息

```
$ ceph osd dump
epoch 44560
fsid b64cae0f-99fd-4ade-b07d-cb0b07145f3b
created xxx
modified xxx
flags sortbitwise
pool 0 'rbd' replicated size 3 min_size 2 crush_ruleset 0 object
_hash rjenkins pg_num 512 pgp_num 512 last_change 2538 flags has
hpspool stripe_width 0
        removed_snaps [1~3]
pool 3 'cephfs_data' replicated size 3 min_size 2 crush_ruleset
0 object_hash rjenkins pg_num 16384 pgp_num 16384 last_change 20
71 flags hashpspool crash_replay_interval 45 stripe_width 0
pool 4 'cephfs_metadata' replicated size 3 min_size 2 crush_rule
set 0 object_hash rjenkins pg_num 1024 pgp_num 1024 last_change
2069 flags hashpspool stripe_width 0
max_osd 546
osd.0 up   in  weight 1 up_from 18561 up_thru 44007 down_at 1855
3 last_clean_interval [4,18556) 10.10.10.11:6800/18728 10.10.10.
11:6837/1018728 10.10.10.11:6838/1018728 10.10.10.11:6839/101872
8 exists,up b84555fe-af15-4959-b247-3fc578f4b39b
osd.1 up   in  weight 1 up_from 24508 up_thru 44402 down_at 2450
4 last_clean_interval [8,24506) 10.10.10.11:6804/19394 10.10.10.
11:6848/1019394 10.10.10.11:6849/1019394 10.10.10.11:6850/101939
4 exists,up fa5976b1-6504-4b87-8f50-911839fa93f3
......
......
```

其中max_osd-1表示集群中最大的osd编号

ceph-deploy简介

ceph-deploy是一个自动化部署ceph集群的工具，使用ceph-deploy我们可以很容易就部署一套ceph集群。

ceph-deploy工具使用ssh连接ceph节点，执行相应的python脚本来完成任务，ceph-deploy是使用python开发的。

官网文档地址: http://docs.ceph.com/ceph-deploy/docs/#

安装

CentOS7

```
# pip install ceph-deploy
```

也可以使用yum安装，但先要配置ceph仓库。

macOS

```
➜  ~ pip install ceph-deploy

➜  ~ pip list | grep ceph-deploy
ceph-deploy (1.5.38)
```

安装前准备工作

在使用ceph-deploy部署ceph集群前，有如下checklist:

1. 确保ntp服务器已开启，并且各个节点上时间是同步的(参考ntp配置和ntp配置注意事项)
2. 每个节点之间，支持通过主机名互相访问(通过注册DNS或者编辑/etc/hosts文件，建议放到**DNS**中)
3. 每个节点上都需要创建一个具有sudo权限的用户，配置ceph-deploy节点可以通过这个用户免密码登录到各个结点上
4. 确保各个节点的防火墙服务已关闭
5. 确保各个节点的selinux服务已关闭

在上面的准备工作中，都是需要在么个节点上一个个去做的，可以通过ansible脚本来完成。可以使用我写的role https://galaxy.ansible.com/frank6866/ceph-prepare/

ssh端口配置

ceph-deploy没有配置ssh端口的配置,可以在ceph-deploy结点上通过修改.ssh/config文件配置ssh端口

~/.ssh/config

```
Host ceph-1
        Port 2345
```

ceph-deploy用法

创建一个新的集群

使用ceph-deploy new命令创建一个新的集群,参数为monitor结点的主机名列表

```
# ceph-deploy new ceph-1 ceph-2 ceph-3
```

该命令会在当前目录下创建如下文件:

```
# ls
ceph.conf  ceph-deploy-ceph.log  ceph.mon.keyring
```

ceph.conf文件如下:

```
# cat ceph.conf
[global]
fsid = 21b1a0c0-6e0d-4d02-b7fb-e5209283abfb
mon_initial_members = ceph-1, ceph-2, ceph-3
mon_host = 10.10.10.5,10.10.10.6,10.10.10.7
auth_cluster_required = cephx
auth_service_required = cephx
auth_client_required = cephx
```

生成的kering文件如下:

```
# cat ceph.mon.keyring
[mon.]
key = abcdefg
caps mon = allow *
```

安装ceph相关的package

```
# ceph-deploy install ceph-1 ceph-2 ceph-3
```

如果目标结点上/etc/yum.repos.d/ceph.repo文件不存在,ceph-deploy install会在目标结点上创建/etc/yum.repos.d/ceph.repo文件。也可以自定义repo的地址:

```
# ceph-deploy install ceph-1 ceph-2 ceph-3  --repo-url http://mi
rrors.aliyun.com/ceph/rpm-jewel/el7/
```

初始化Monitor节点

```
# ceph-deploy mon create-initial
```

推送配置文件

```
# ceph-deploy admin ceph-client
```

该命令会将ceph-deploy结点上的ceph.conf文件和ceph.client.admin.keyring文件拷贝到ceph-client结点的/etc/ceph目录中。

查看磁盘列表

```
# ceph-deploy disk list ceph-1
......
[ceph_deploy.osd][DEBUG ] Listing disks on ceph-1...
[ceph-1][DEBUG ] find the location of an executable
[ceph-1][INFO  ] Running command: /usr/sbin/ceph-disk list
[ceph-1][DEBUG ] /dev/dm-0 other, xfs, mounted on /
[ceph-1][DEBUG ] /dev/dm-1 swap, swap
[ceph-1][DEBUG ] /dev/sda :
[ceph-1][DEBUG ]  /dev/sda2 other, LVM2_member
[ceph-1][DEBUG ]  /dev/sda1 other, xfs, mounted on /boot
[ceph-1][DEBUG ] /dev/sdb :
[ceph-1][DEBUG ]  /dev/sdb2 other
[ceph-1][DEBUG ]  /dev/sdb1 ceph data, active, cluster ceph, osd
.0
[ceph-1][DEBUG ] /dev/sr0 other, unknown
[root@ceph-1 deploy]#
```

添加osd结点

```
# ceph-deploy osd create ceph-1:/dev/sdb
```

初始化集群

首先进行安装前准备工作

1.创建一个新的集群

使用ceph-deploy new命令创建一个新的集群,参数为monitor结点的主机名列表

```
# ceph-deploy new ceph-1 ceph-2 ceph-3
```

2.安装ceph相关的package

```
# ceph-deploy install ceph-1 ceph-2 ceph-3
```

如果目标结点上/etc/yum.repos.d/ceph.repo文件不存在,ceph-deploy install会在目标结点上创建/etc/yum.repos.d/ceph.repo文件。也可以自定义repo的地址:

```
# ceph-deploy install ceph-1 ceph-2 ceph-3  --repo-url http://mi
rrors.aliyun.com/ceph/rpm-jewel/el7/
```

3.初始化Monitor节点

```
# ceph-deploy mon create-initial
```

添加osd

首先进行安装前准备工作

1.安装ceph相关的package

```
# ceph-deploy install ceph-1 ceph-2 ceph-3
```

如果目标结点上/etc/yum.repos.d/ceph.repo文件不存在,ceph-deploy install会在目标结点上创建/etc/yum.repos.d/ceph.repo文件。也可以自定义repo的地址:

```
# ceph-deploy install ceph-1 ceph-2 ceph-3  --repo-url http://mi
rrors.aliyun.com/ceph/rpm-jewel/el7/
```

2. 查看磁盘列表

```
# ceph-deploy disk list ceph-1
......
[ceph_deploy.osd][DEBUG ] Listing disks on ceph-1...
[ceph-1][DEBUG ] find the location of an executable
[ceph-1][INFO  ] Running command: /usr/sbin/ceph-disk list
[ceph-1][DEBUG ] /dev/dm-0 other, xfs, mounted on /
[ceph-1][DEBUG ] /dev/dm-1 swap, swap
[ceph-1][DEBUG ] /dev/sda :
[ceph-1][DEBUG ]  /dev/sda2 other, LVM2_member
[ceph-1][DEBUG ]  /dev/sda1 other, xfs, mounted on /boot
[ceph-1][DEBUG ] /dev/sdb :
[ceph-1][DEBUG ]  /dev/sdb2 other
[ceph-1][DEBUG ]  /dev/sdb1 ceph data, active, cluster ceph, osd
.0
[ceph-1][DEBUG ] /dev/sr0 other, unknown
[root@ceph-1 deploy]#
```

3.清除磁盘数据

如果磁盘上有数据,可以先清除

```
# ceph-deploy disk zap ceph-1:/dev/sdb
```

4.添加osd结点

```
# ceph-deploy osd create ceph-1:/dev/sdb
```

添加rgw

首先进行安装前准备工作

1.安装ceph相关的package

```
# ceph-deploy install ceph-1 ceph-2 ceph-3
```

如果目标结点上/etc/yum.repos.d/ceph.repo文件不存在,ceph-deploy install会在目标结点上创建/etc/yum.repos.d/ceph.repo文件。也可以自定义repo的地址:

```
# ceph-deploy install ceph-1 ceph-2 ceph-3  --repo-url http://mi
rrors.aliyun.com/ceph/rpm-jewel/el7/
```

2.安装rgw

```
# ceph-deploy rgw create ceph-1 ceph-2 ceph-3
```

移除OSD

本文参考同事的文档

1.在crush中设置OSD weight为0，等待迁移完成。

```
# ceph osd crush reweight osd.{osd-num} 0
```

2.从集群中设置OSD为out（如果OSD还处于in状态）

```
# ceph osd out {osd-num}
```

3.停止OSD进程（如果进程还在运行）：

```
# systemctl stop ceph-osd@{osd-num}
```

4.从CRUSH map中移除osd

```
# ceph osd crush remove osd.{osd-num}
```

5.删除osd认证key

```
# ceph auth del osd.{osd-num}
```

6.从集群中移除集群

```
# ceph osd rm {osd-num}
```

移除RGW

本文参考同事的文档

1.停止radosgw服务

```
# systemctl stop ceph-radosgw@rgw.{hostname}
```

2.删除radosgw数据目录

```
# rm -rf /var/lib/ceph/radosgw/ceph-rgw.{hostname}
```

3.删除radosgw认证文件

```
# ceph auth del client.rgw.{hostname}
```

更换journal device

本文参考同事的文档

该步骤主要针对集群中OSD使用了独立的journal device（一般为SSD）。

更换journal disk的步骤如下:

1.检查当前OSD节点所有OSD jorunal使用设备，确定更换journal影响的OSD。例如，查看某个OSD journal使用设备的过程如下：

```
# ls -al /var/lib/ceph/osd/ceph-0/journal
lrwxrwxrwx 1 ceph ceph 58 Dec 27 15:07 /var/lib/ceph/osd/ceph-0/
journal -> /dev/disk/by-partuuid/28e46cec-d906-4edd-a04b-aeef1b4
abce3

# ls -al /dev/disk/by-partuuid/28e46cec-d906-4edd-a04b-aeef1b4ab
ce3
lrwxrwxrwx 1 root root 10 Dec 27 15:26 /dev/disk/by-partuuid/28e
46cec-d906-4edd-a04b-aeef1b4abce3 -> ../../sdb1
```

2.设置集群为noout，停止受到影响的OSD进程：

```
# ceph osd set noout
# ceph osd down osd.{osd-num};systemctl stop ceph-osd@{osd-num}
```

3.刷新journal数据到数据盘：

```
# ceph-osd -i {osd-num} --flush-journal
```

4.更换journal disk

5.创建新journal 在新磁盘创建分区：

```
# sgdisk --new=0:0:+{journal_size} --change-name=0:'ceph journal
' --partition-guid=0:{journal_uuid} --typecode=0:{journal_uuid}
--mbrtogpt -- {journal_disk}

# partprobe
查找分区对应partuuid：
# ls -al /dev/disk/by-partuuid/
新分区使用partuuid软链接到OSD数据目录：
# ln -s /dev/disk/by-partuuid/{part-uuid} /var/lib/ceph/ceph-{os
d-num}/journal
chown ceph: /var/lib/ceph/ceph-{osd-num}/journal
创建journal:
# ceph-osd -i {osd-num} --mkjournal
注：{journal_size}为journal大小，定义在ceph.conf中。{journal_uuid}为
OSD数据目录下journal_uuid文件内容，例如/var/lib/ceph/osd/ceph-{osd-n
um}/journal_uuid
```

1. 启动OSD：

```
# systemctl start ceph-osd@{osd-num}
```

pg状态

以一个正在扩容的集群为例:

```
# ceph -s
    cluster xxxxxx
     health HEALTH_WARN
            9 pgs backfill_toofull
            854 pgs backfill_wait
            12 pgs backfilling
            569 pgs degraded
            441 pgs recovery_wait
            1420 pgs stuck unclean
            128 pgs undersized
            recovery 756251/330647041 objects degraded (0.229%)
            recovery 11888297/330647041 objects misplaced (3.595
%)
            18 near full osd(s)
            mds0: Client 32724142 failing to respond to cache pr
essure
            1/472 in osds are down
            noout,sortbitwise flag(s) set
     monmap e1: 3 mons at {ceph-1=10.10.10.11:6789/0,ceph-20=10.
10.10.12:6789/0,ceph-40=10.10.10.13:6789/0}
            election epoch 758, quorum 0,1,2 ceph-1,ceph-40,ceph
-20
      fsmap e2910: 1/1/1 up {0=ceph-1=up:active}, 2 up:standby
     osdmap e31996: 476 osds: 471 up, 472 in; 1006 remapped pgs
            flags noout,sortbitwise
      pgmap v28969688: 17920 pgs, 3 pools, 406 TB data, 103 Mobj
ects
            1222 TB used, 491 TB / 1713 TB avail
            756251/330647041 objects degraded (0.229%)
            11888297/330647041 objects misplaced (3.595%)
               16466 active+clean
                 828 active+remapped+wait_backfill
                 414 active+recovery_wait+degraded
                 109 active+undersized+degraded
```

```
                  27 active+recovery_wait+degraded+remapped
                  22 active+clean+scrubbing+deep
                  18 active+undersized+degraded+remapped+wait_ba
ckfill
                  12 active+clean+scrubbing
                  11 active+remapped+backfilling
                   8 active+remapped+wait_backfill+backfill_toof
ull
                   3 active+remapped
                   1 active+remapped+backfill_toofull
                   1 active+undersized+degraded+remapped+backfil
ling
recovery io 1197 MB/s, 308 objects/s
  client io 8117 kB/s rd, 204 MB/s wr, 63 op/s rd, 116 op/s wr
```

- creating: ceph在创建pg
- active: 正常状态，ceph将请求发送到pg上
- clean: ceph对pg中所有对象都创建了配置的副本数
- down: pg中的对象不能满足法定个数，pg离线
- Scrubbing: ceph在check pg的不一致性
- Degraded: pg中的某些对象没有达到指定的副本数
- Inconsistent: ceph检测到pg中某个对象的副本不一致
- peering: pg在进行peering操作
- repairing: ceph在检查pg并尽可能修复不一致的数据
- recovering: ceph在迁移或者同步对象和它的副本
- Backfill: backfill是恢复的一种特例，ceph在扫描和同步pg所有的内容，而不是从日志中查看最近的操作来决定哪些内容需要同步
- Wait-backfill: pg在排队等待开始backfill
- Backfill-toofull: backfill操作在等待，因为目标OSD快满了
- Incomplete: ceph检测到丢失了写操作的信息，或者没有一个正常的副本。遇到这种情况，尝试启动down的osd，可能在这些down的osd里面有需要的信息
- Stale: pg处于未知的状态，自从pg map变化后，monitor没有收到来自pg的消息
- Remapped: pg被临时map到CRUSH指定的osd集合中
- Undersized: pg副本的数量小于pool设置的副本数
- Peered: peering操作已经完成

osd down排查

查看down了的osd

```
# ceph osd tree | grep down
431    3.56000         osd.431             down        0
 1.00000
178    3.63129         osd.178             down  1.00000
 1.00000
461    3.63129         osd.461             down        0
 1.00000
379    3.63129         osd.379             down        0
 1.00000
411    3.63129         osd.411             down        0
 1.00000
```

以431为例，查看osd所在的主机

```
# ceph osd find 431
{
    "osd": 431,
    "ip": "10.0.36.32:6812\/21453",
    "crush_location": {
        "host": "xg-ops-ceph-3",
        "root": "default"
    }
}
```

登录osd所在主机，

pg数量

创建一个pool时，必须指定该pool中pg的数量，

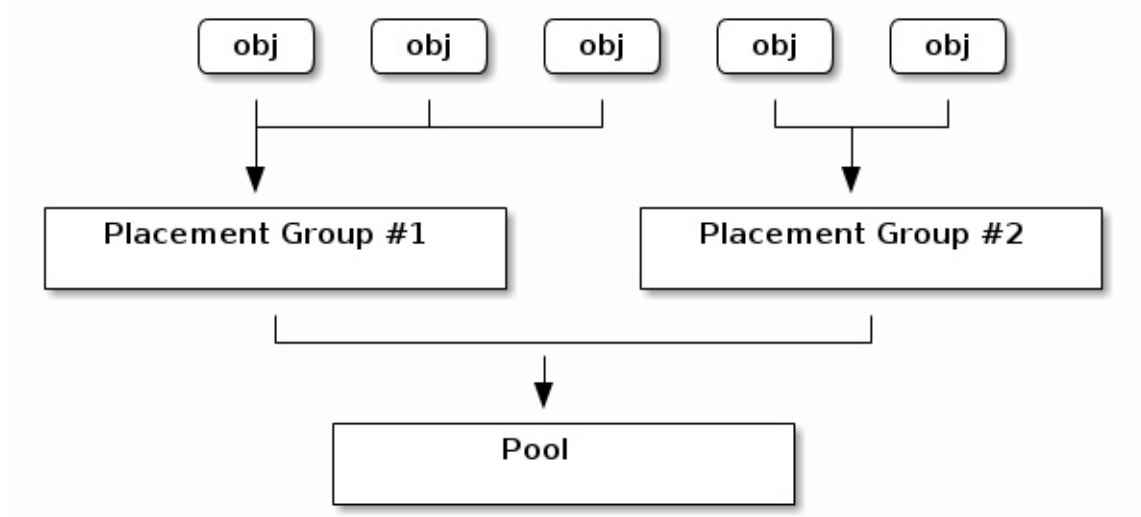
```
# ceph osd pool create {pool-name} pg_num
```

pg的数量建议设置为2的n次方，在osd少于50个时，可以参考下面的规则设置pg数量:

- osd数量少于5个时，pg数量设置为128
- 5-10个osd时，pg数量设置为512
- 10-50个osd时，pg数量设置为1024

当osd数量超过50个时，需要考虑多个因素确定pg的数量。

在pool中，pg将对象聚合在一起，因为追踪每个对象的成本太高了。如下图:



Ceph客户端会计算将对象放到哪个pg中，计算方法是： 对object id做hash，然后根据pool id和pool中pg个数计算出将这个对象放到哪个pg中。

对象所在的pg会被存储在osd集合中，osd集合的个数是pool的副本数。

数据持久性

当一个osd宕掉时，在这个osd包含的数据恢复前，数据丢失的风险会增加。下面的一些场景中，某个pg中所有的数据会永久丢失:

- osd宕掉，osd所包含的所有的数据都丢失了。在这个osd中所有的对象，副本数会减少1
- ceph开始恢复宕掉osd中的pg，选择一个新的osd创建该pg所有对象的副本

在一个ceph集群中，如果有10个osd，某个pool中有512个pg，副本数是3；CRUSH算法会给每个pg分配3个osd。因此，每个osd上pg的个数是(512*3)/10=~150。当一个osd宕掉时，会开始恢复这个osd上的约150个pg。剩余的osd会向其他osd发送对象，也会从其他osd接受对象。

osd宕掉后，恢复的时间取决于ceph的架构。假设每个OSD都是在1TB的SSD上，并且是万兆网络，那么恢复一个osd的时间在分钟级别。

选择pg数量

一般情况下，让osd上pg上的数量在100个左右，根据副本数，pg数量计算公式为:

pgs = (osd * 100)/副本数

pg的数量最好设置为2的n次方，将上面计算的数量，向上取值到最接近的2的n次方，就是pg的数量。

比如，有500个osd，副本数是3，

psg = (500 * 100) / 3 ~= 16666

大于16666最小的2的n次方的数是，32768

集群状态

使用如下命令可以查看集群状态:

> ceph -s

示例输出

```
[root@frank-ceph-1 ~]# ceph -s
    cluster c712f08b-c001-4b1c-969d-abec240138f7
     health HEALTH_WARN
            clock skew detected on mon.frank-ceph-2, mon.frank-c
eph-3
            Monitor clock skew detected
     monmap e1: 3 mons at {frank-ceph-1=10.10.10.5:6789/0,frank-
ceph-2=10.10.10.6:6789/0,frank-ceph-3=10.10.10.7:6789/0}
            election epoch 4, quorum 0,1,2 frank-ceph-1,frank-ce
ph-2,frank-ceph-3
     osdmap e15: 3 osds: 3 up, 3 in
            flags sortbitwise,require_jewel_osds
      pgmap v31: 64 pgs, 1 pools, 0 bytes data, 0 objects
            101 MB used, 134 GB / 134 GB avail
                  64 active+clean
```

cluster

> cluster c712f08b-c001-4b1c-969d-abec240138f7

cluster表示集群的id。

health

health表示集群的健康状态:

- **HEALTH_OK**表示集群状态正常
- **HEALTH_WARN**表示有告警

集群节点信息

使用ceph node命令可以查看集群的中所有结点的信息

```
# ceph node ls
{
    "mon": {
        "ceph-1": [
            0
        ],
        "ceph-2": [
            1
        ],
        "ceph-3": [
            2
        ]
    },
    "osd": {
        "ceph-1": [
            1,
            3,
            4,
            5
        ],
        "ceph-2": [
            6,
            7,
            8
        ],
        "ceph-3": [
            9,
            10,
            11
        ]
    },
    "mds": {}
}
```

mon表示monitor的结点信息，按照主机名输出，每个主机名后面的数组表示mon结点的编号；osd和mds也是。

也可以只查看部分信息，比如，只查看mon结点的信息:

```
# ceph node ls mon
{
    "ceph-1": [
        0
    ],
    "ceph-2": [
        1
    ],
    "ceph-3": [
        2
    ]
}
```

集群空间使用

ceph df命令可以查询集群空间使用情况

```
# ceph df
GLOBAL:
    SIZE      AVAIL     RAW USED     %RAW USED
    8016G     7317G         698G          8.72
POOLS:
    NAME                            ID      USED       %USED
  MAX AVAIL     OBJECTS
    rbd                             0        3917M      0.14
      2668G        1870
    volumes                         369       124G      4.46
      2668G       33127
    images                          370     82023M      2.91
      2668G       10307
    vms                             371          0         0
      2668G           2
    backups                         372      2048M      0.07
      2668G         516
......
```

GLOBAL区域表示整体的空间使用情况:

- SIZE: 表示集群中所有OSD总空间大小
- AVAIL: 表示可以使用的空间大小
- RAW USED: 表示已用空间大小
- %RAW USED: 表示已用空间百分比

POOLS区域表示某个pool的空间使用情况

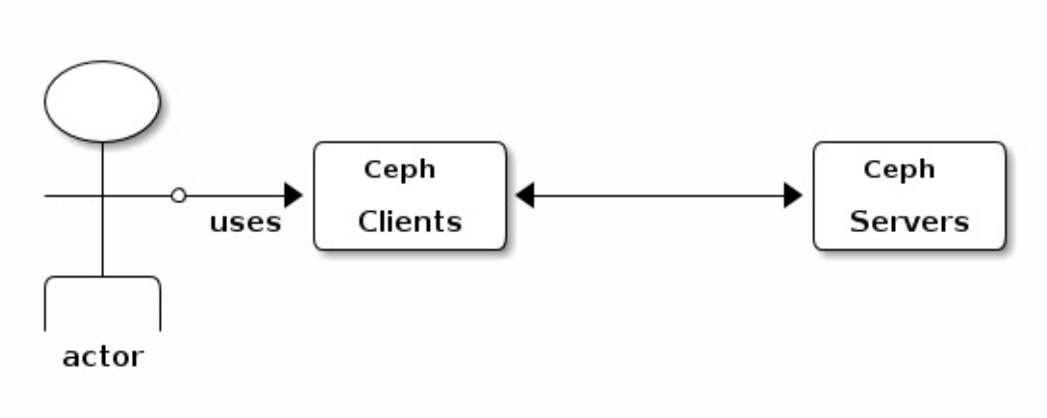
- NAME: pool名称
- ID: pool id
- USED: 已用空间大小
- %USED: 已用空间百分比
- MAX AVAIL: 最大可用空间大小

- OBJECTS: 这个pool中对象的个数

> 注意: pool里面的已有空间是业务上的空间,也就是一个副本的空间;将业务上空间乘以副本数,和RAW USED是相等的。RAW USED是集群物理上已近使用的空间。

用户管理

在Ceph中，用户使用Ceph client和Ceph server交互，如下图:



默认情况下，Ceph启用了身份认证；我们必须提供用户名和keyring(钥匙串，包含密钥)。如果不指定用户名，Ceph会使用client.admin作为默认的用户名。如果不指定keyring，Ceph会使用/etc/ceph下和用户名同名的keyring文件。比如，下面的命令：

```
# ceph health
```

会被解析为：

```
# ceph -n client.admin --keyring=/etc/ceph/ceph.client.admin.key
ring health
```

其中"-n"表示client_name，即客户端的名称

对于所有的Ceph client(比如RBD、RGW或CephFS)，Ceph将数据作为对象存储在pools中。为了读写数据，Ceph的用户必须具有访问pools的权限。

User(用户)

一个user可以是一个individual(个体)或者一个应用程序。创建用户可以让我们控制谁可以访问Ceph集群，谁可以访问哪些pools。

Ceph中用户有一个type(类型)的属性，为了管理的目的，type通常为**client**。Ceph中用户的格式为：**TYPE.ID**，其中TYPE表示用户类型，ID为用户ID。使用type的原因是：Ceph Monitors、OSDs和Metadata Servers也使用Cephx协议，但他们不是client；使用type是为了方便把他们区分开来。

Authorization (Capabilities)

Ceph使用术语“capabilities”(caps)来描述用户操作monitors、OSDs和metadata servers时的权限检查。 Capabilities也可以限制访问pool中的数据或者命名空间。Ceph管理员可以在创建或者更新用户的时候设置用户的Capabilities。

POOL(池)

池是一个顶层的逻辑分区，一般是针对不同的应用、或者应用中不同的模块建立相应的用户和池。

比如，OpenStack使用Ceph作为后端存储时：

- nova模块: 建立vms池
- glance模块: 建立images池，创建client.glance用户
- cinder模块: 建立volumes和backups池，创建client.cinder用户

Namespace(命名空间)

namespace(命名空间)是池中对象的逻辑分组，池中的对象可以被关联到命名空间；一个用户的访问权限粒度可以到命名空间级别。

但是命名空间只对位于librados上层的应用程序有用。RBD、RGW和CephFS暂时不支持这种特性。

用户管理

ceph中用户管理的子命令是auth

添加用户

命令格式是

```
# auth add <entity> {<caps> [<caps>...]}
```

比如，添加一个名为frank6866的用户，在mon上具有读权限，在cinder-back这个pool上具有读写的权限:

```
# ceph auth add client.frank6866 mon 'allow r' osd 'allow rw poo
l=cinder-backup'
added key for client.frank6866
```

查看用户列表

```
# ceph auth list
installed auth entries:
......
client.frank6866
    key: xxx
    caps: [mon] allow r
    caps: [osd] allow rw pool=cinder-backup
......
```

获取某个用户的认证信息

```
# ceph auth get client.frank6866
exported keyring for client.frank6866
[client.frank6866]
    key = xxx
    caps mon = "allow r"
    caps osd = "allow rw pool=cinder-backup"
```

导出用户

导出client.frank6866用户到client.frank6866.auth文件中

```
# ceph auth export client.frank6866 -o client.frank6866.auth
```

查看导出的文件内容:

```
# cat client.frank6866.auth
[client.frank6866]
    key = AQC9vSFZCC/7NBAA5oGudHTnunQGu/QmbyK22A==
    caps mon = "allow r"
    caps osd = "allow rw pool=cinder-backup"
```

删除用户

删除client.frank6866用户

```
# ceph auth del client.frank6866
```

导入用户

从client.frank6866.auth文件中导入client.frank6866用户

```
# ceph auth import -i client.frank6866.auth
imported keyring
```

monitor管理

查询选举状态

```
# ceph quorum_status | python -mjson.tool
{
    "election_epoch": 118,
    "monmap": {
        "created": "2016-12-16 11:45:55.627125",
        "epoch": 5,
        "fsid": "0657f4e6-0601-430f-b604-bd1219e0ef09",
        "modified": "2016-12-25 09:46:57.098524",
        "mons": [
            {
                "addr": "10.10.10.26:6789/0",
                "name": "ceph-1",
                "rank": 0
            },
            {
                "addr": "10.10.10.27:6789/0",
                "name": "ceph-2",
                "rank": 1
            },
            {
                "addr": "10.10.10.28:6789/0",
                "name": "ceph-3",
                "rank": 2
            }
        ]
    },
    "quorum": [
        0,
        1,
        2
    ],
    "quorum_leader_name": "ceph-1",
    "quorum_names": [
        "ceph-1",
        "ceph-2",
        "ceph-3"
    ]
}
```

- election_epoch: 总共选举的次数
- quorum: mon节点rank列表
- quorum_leader_name: leader节点名称
- quorum_names: 所有成员的名称

monmap输出详解

- created: 创建时间
- epoch: 当前monmap的版本号
- mons: 包括每个mon的ip地址和端口号、主机名及rank。rank的计算公式是IP:port越小，rank越小

查看Monitor详细信息

```
# ceph mon_status | python -mjson.tool
{
    "election_epoch": 118,
    "extra_probe_peers": [],
    "monmap": {
        "created": "2016-12-16 11:45:55.627125",
        "epoch": 5,
        "fsid": "0657f4e6-0601-430f-b604-bd1219e0ef09",
        "modified": "2016-12-25 09:46:57.098524",
        "mons": [
            {
                "addr": "10.10.10.26:6789/0",
                "name": "ceph-1",
                "rank": 0
            },
            {
                "addr": "10.10.10.27:6789/0",
                "name": "ceph-2",
                "rank": 1
            },
            {
                "addr": "10.10.10.28:6789/0",
                "name": "ceph-3",
                "rank": 2
            }
        ]
    },
    "name": "ceph-2",
    "outside_quorum": [],
    "quorum": [
        0,
        1,
        2
    ],
    "rank": 1,
    "state": "peon",
    "sync_provider": []
}
```

查看Monitor概要信息

```
# ceph mon stat
e5: 3 mons at {ceph-1=10.10.10.26:6789/0,ceph-2=10.10.10.27:6789
/0,ceph-3=10.10.10.28:6789/0}, election epoch 118, quorum 0,1,2
ceph-1,ceph-2,ceph-3
```

- e5: 表示当前monmap的版本是第5版
- election epoch 118: 表示选举的总次数

移除Monitor

```
# ceph mon remove ceph-1
Error EINVAL: removing mon.ceph-1 at 10.10.10.75:6789/0, there w
ill be 2 monitors

# ceph mon stat
7f6b0032e700  0 -- :/3940951470 >> 10.10.10.75:6789/0 pipe(0x7f6
afc05cda0 sd=3 :0 s=1 pgs=0 cs=0 l=1 c=0x7f6afc05e060).fault
e2: 2 mons at {ceph-2=10.10.10.76:6789/0,ceph-3=10.10.10.77:6789
/0}, election epoch 20, quorum 0,1 ceph-2,ceph-3
```

添加Monitor

```
# ceph mon add ceph-1 10.10.10.75:6789
adding mon.ceph-1 at 10.10.10.75:6789/0

# ceph mon stat
e3: 3 mons at {ceph-1=10.10.10.75:6789/0,ceph-2=10.10.10.76:6789
/0,ceph-3=10.10.10.77:6789/0}, election epoch 22, quorum 1,2 cep
h-2,ceph-3
```

获取某个版本的monmap

版本号为ceph mon_status命令monmap中的epoch值

```
# ceph mon dump 3
dumped monmap epoch 3
epoch 3
fsid c712f08b-c001-4b1c-969d-abec240138f7
last_changed 2017-05-30 15:48:18.601129
created 2017-03-16 17:52:01.252939
0: 10.10.10.75:6789/0 mon.ceph-1
1: 10.10.10.76:6789/0 mon.ceph-2
2: 10.10.10.77:6789/0 mon.ceph-3
```

如果不加入版本号，默认获取最新的monmap

```
# ceph mon dump
dumped monmap epoch 3
epoch 3
fsid c712f08b-c001-4b1c-969d-abec240138f7
last_changed 2017-05-30 15:48:18.601129
created 2017-03-16 17:52:01.252939
0: 10.10.10.75:6789/0 mon.ceph-1
1: 10.10.10.76:6789/0 mon.ceph-2
2: 10.10.10.77:6789/0 mon.ceph-3
```

获取Monitor的metadata

```
# ceph mon metadata ceph-1
{
    "arch": "x86_64",
    "cpu": "Intel(R) Xeon(R) CPU E5-2420 v2 @ 2.20GHz",
    "distro": "CentOS",
    "distro_codename": "Core",
    "distro_description": "CentOS Linux release 7.1.1503 (Core)
",
    "distro_version": "7.1.1503",
    "hostname": "ceph-1",
    "kernel_description": "#1 SMP Fri Mar 6 11:36:42 UTC 2015",
    "kernel_version": "3.10.0-229.el7.x86_64",
    "mem_swap_kb": "1679356",
    "mem_total_kb": "1870496",
    "os": "Linux"
}
```

获取的主要是Monitor所在操作系统的信息

osd管理

对于每个osd，ceph都会启动一个进程，如下:

```
# ps -f -u ceph
UID        PID  PPID  C STIME TTY          TIME CMD
ceph     22937     1  0 Mar02 ?        15:46:29 /usr/bin/ceph-mo
n -f --cluster ceph --id ceph-3 --setuser ceph --setgroup
ceph     23849     1  3 Mar02 ?        2-13:36:15 /usr/bin/ceph-
osd -f --cluster ceph --id 10 --setuser ceph --setgroup ceph
ceph     24014     1  2 Mar02 ?        2-00:58:38 /usr/bin/ceph-
osd -f --cluster ceph --id 11 --setuser ceph --setgroup ceph
ceph     24177     1  3 Mar02 ?        2-12:34:28 /usr/bin/ceph-
osd -f --cluster ceph --id 9 --setuser ceph --setgroup ceph
ceph     24970     1  0 Mar02 ?        04:23:39 /usr/bin/radosgw
 -f --cluster ceph --name client.rgw.ceph-3 --se
```

创建osd可以使用ceph-deploy工具。

查询集群osd状态

```
# ceph osd stat
     osdmap e1952: 10 osds: 9 up, 9 in
            flags sortbitwise,require_jewel_osds
```

查看所有osd的id

```
# ceph osd ls
1
3
4
5
6
7
......
```

查看osd的空间使用率

```
# ceph osd df
ID WEIGHT  REWEIGHT SIZE  USE    AVAIL %USE  VAR  PGS
 3 0.86909  1.00000  889G 78631M  813G  8.63 0.99 329
 4 0.86909  0.98000  889G 81207M  810G  8.91 1.02 333
 5 0.86909  1.00000  889G 78747M  813G  8.64 0.99 346
......
              TOTAL 8016G   698G 7317G  8.72
MIN/MAX VAR: 0.74/1.21  STDDEV: 1.13
```

- ID: osd id
- WEIGHT: 权重，和osd容量有关系
- REWEIGHT: 自动以的权重
- SIZE: osd大小
- USE: 已用空间大小
- AVAIL: 可用空间大小
- %USE: 已用空间百分比
- PGS: pg数量

查询osd在哪个主机上

```
# ceph osd find 0
{
    "osd": 0,
    "ip": "10.10.10.75:6800\/2101",
    "crush_location": {
        "host": "ceph-1",
        "root": "default"
    }
}
```

osd会使用6800-7300之间可用的端口号，一个osd最多会用到4个端口号。

```
# ss -ntpl | grep ceph
LISTEN     0      128                          *:6800
       *:*      users:(("ceph-osd",2101,4))
LISTEN     0      128                          *:6801
       *:*      users:(("ceph-osd",2101,5))
LISTEN     0      128                          *:6802
       *:*      users:(("ceph-osd",2101,6))
LISTEN     0      128                          *:6803
       *:*      users:(("ceph-osd",2101,7))
```

查看osd的metadata

```
# ceph osd metadata 1
{
    "id": 1,
    "arch": "x86_64",
    "back_addr": "10.10.10.26:6801\/154023",
    "backend_filestore_dev_node": "unknown",
    "backend_filestore_partition_path": "unknown",
    "ceph_version": "ceph version 10.2.5-6099-gd9eaab4 (d9eaab45
6ff45ae88e83bd633f0c4efb5902bf07)",
    "cpu": "Intel(R) Xeon(R) CPU E5-2620 v2 @ 2.10GHz",
    "distro": "centos",
    "distro_description": "CentOS Linux 7 (Core)",
    "distro_version": "7",
    "filestore_backend": "xfs",
    "filestore_f_type": "0x58465342",
    "front_addr": "10.10.10.26:6800\/154023",
    "hb_back_addr": "10.10.10.26:6803\/154023",
    "hb_front_addr": "10.10.10.26:6804\/154023",
    "hostname": "ceph-1",
    "kernel_description": "#1 SMP Fri Mar 6 11:36:42 UTC 2015",
    "kernel_version": "3.10.0-229.el7.x86_64",
    "mem_swap_kb": "4194300",
    "mem_total_kb": "131812072",
    "os": "Linux",
    "osd_data": "\/var\/lib\/ceph\/osd\/ceph-1",
    "osd_journal": "\/var\/lib\/ceph\/osd\/ceph-1\/journal",
    "osd_objectstore": "filestore"
}
```

- id: osd的id
- filestore_backend: filestore所使用的文件系统类型
- hostname: osd所在的主机
- osd_data: osd数据存放位置
- osd_journal: osd的日志存放位置
- osd_objectstore: osd存储对象使用的store类型

查看osd tree

```
# ceph osd tree
ID WEIGHT  TYPE NAME                UP/DOWN REWEIGHT PRIMARY-AFF
INITY
-1 8.73654 root default
-2 3.51538     host ceph-1
 3 0.86909         osd.3                up  1.00000          1.
00000
 4 0.86909         osd.4                up  0.98000          1.
00000
 5 0.86909         osd.5                up  1.00000          1.
00000
 1 0.90810         osd.1              down        0          1.
00000
-3 2.61058     host ceph-2
 6 0.87019         osd.6                up  1.00000          1.
00000
 7 0.87019         osd.7                up  0.79999          1.
00000
 8 0.87019         osd.8                up  1.00000          1.
00000
-4 2.61058     host ceph-3
 9 0.87019         osd.9                up  0.98999          1.
00000
10 0.87019         osd.10               up  1.00000          1.
00000
11 0.87019         osd.11               up  0.98000          1.
00000
```

- ID: 如果为负数，表示的是主机或者root；如果是正数，表示的是osd的id
- WEIGHT: osd的weight，root的weight是所有host的weight的和。某个host的weight是它上面所有osd的weight的和
- NAME: 主机名或者osd的名称
- UP/DOWN: osd的状态信息
- REWEIGHT: osd的reweight值，如果osd状态为down，reweight值为0

查看**osd map**的概要信息

```
# ceph osd dump
epoch 38
fsid c712f08b-c001-4b1c-969d-abec240138f7
created 2017-03-16 17:52:12.901166
modified 2017-06-03 16:53:28.572763
flags sortbitwise,require_jewel_osds
pool 0 'rbd' replicated size 3 min_size 2 crush_ruleset 0 object
_hash rjenkins pg_num 64 pgp_num 64 last_change 1 flags hashpspo
ol stripe_width 0
pool 1 'pool-frank6866' replicated size 3 min_size 2 crush_rules
et 0 object_hash rjenkins pg_num 128 pgp_num 128 last_change 37
flags hashpspool stripe_width 0
max_osd 3
osd.0 up   in  weight 1 up_from 28 up_thru 37 down_at 26 last_cl
ean_interval [23,25) 10.10.10.75:6800/2101 10.10.10.75:6801/2101
 10.10.10.75:6802/2101 10.10.10.75:6803/2101 exists,up 79b807fa-
d0cd-4d70-a47b-acd4148c9d16
osd.1 up   in  weight 1 up_from 8 up_thru 37 down_at 0 last_clea
n_interval [0,0) 10.10.10.76:6800/20176 10.10.10.76:6801/20176 1
0.10.10.76:6802/20176 10.10.10.76:6803/20176 exists,up 9dded204-
2d3b-4a5d-a820-d817385a0e35
osd.2 up   in  weight 1 up_from 13 up_thru 37 down_at 0 last_cle
an_interval [0,0) 10.10.10.77:6800/20546 10.10.10.77:6801/20546
10.10.10.77:6802/20546 10.10.10.77:6803/20546 exists,up 7d01124b
-c6f9-46cd-b32b-34aa40493621
```

还可以看到pool的信息

pool管理

介绍pool管理相关的命令

创建pool

```
# ceph osd pool create pool-frank6866 128
pool 'pool-frank6866' created
```

列出所有pool

```
# ceph osd pool ls
rbd
pool-frank6866
```

也可以查看pool的详细信息:

```
# ceph osd pool ls detail
pool 0 'rbd' replicated size 3 min_size 2 crush_ruleset 0 object
_hash rjenkins pg_num 64 pgp_num 64 last_change 1 flags hashpspo
ol stripe_width 0
pool 1 'pool-frank6866' replicated size 3 min_size 2 crush_rules
et 0 object_hash rjenkins pg_num 128 pgp_num 128 last_change 37
flags hashpspool stripe_width 0
```

- replicated size: 副本数
- min_size: 最小的副本数
- pg_num: pg数量

查看某个pool的pg个数

查看名为pool-frank6866的pool中pg的个数

```
# ceph osd pool get pool-frank6866 pg_num
pg_num: 128
```

修改pool中对象的副本数

将pool-frank6866的副本数设置为5

```
# ceph osd pool set pool-frank6866 size 5
set pool 1 size to 5
```

查看所有pool的状态

```
# ceph osd pool stats
pool rbd id 0
  nothing is going on

pool volumes id 369
  client io 0 B/s rd, 264 kB/s wr, 44 op/s rd, 55 op/s wr

pool images id 370
  nothing is going on

pool vms id 371
  nothing is going on
```

- pool后面是pool的名称，比如rbd、volumes等，id后面是pool的id。
- io表示的是客户端使用这个pool的io情况，B/s rd表示读的速率，kB/s wr表示写速度；op/s rd表示读的iops，op/s wr表示写的iops

获取pool的配额信息

```
# ceph osd pool get-quota volumes
quotas for pool 'volumes':
  max objects: N/A
  max bytes  : N/A
```

- max objects: 最大对象数，默认为N/A，表示不限制
- max bytes: 最大空间，默认为N/A，表示不限制

往pool中上传对象

```
# dd if=/dev/zero of=data.img bs=1M count=32
# rados -p pool-frank6866 put object-data data.img
```

列出pool中的对象

```
# rados -p pool-frank6866 ls
object-data
```

```
# ceph osd map pool-frank6866 object-data
osdmap e42 pool 'pool-frank6866' (1) object 'object-data' -> pg
1.c9cf1b74 (1.74) -> up ([2,0,1], p2) acting ([2,0,1], p2)
```

- osdmap e42: 表示osdmap的版本是42
- pool 'pool-frank6866': 表示pool的名称是pool-frank6866
- (1): 表示pool的id是1
- object 'object-data': 表示对象名是object-data
- pg 1.c9cf1b74 (1.74): 表示对象所属pg的id是1.74,c9cf1b74表示的是对象的id
- up ([2,0,1], p2): 这里副本数设置的是3,up表示该对象所在的osd的id

查找id为2的osd所在的主机

```
# ceph osd find 2
{
    "osd": 2,
    "ip": "10.10.10.77:6800\/20546",
    "crush_location": {
        "host": "ceph-3",
        "root": "default"
    }
}
```

登录osd所在主机上查看挂载的目录信息:

```
# df -lTh /var/lib/ceph/osd/ceph-2
Filesystem     Type  Size  Used Avail Use% Mounted on
/dev/sdb1      xfs    45G   68M   45G   1% /var/lib/ceph/osd/cep
h-2
```

根据pg id查看该pg存放数据的地方:

```
# ls -al /var/lib/ceph/osd/ceph-2/current | grep 1.74
drwxr-xr-x   2 ceph ceph    67 Jun  3 17:20 1.74_head
drwxr-xr-x   2 ceph ceph     6 Jun  3 16:53 1.74_TEMP
```

查看pg所在目录的结构:

```
# tree /var/lib/ceph/osd/ceph-2/current/1.74_head/
/var/lib/ceph/osd/ceph-2/current/1.74_head/
├── __head_00000074__1
└── object-data__head_C9CF1B74__1
```

crush管理

查看crush tree

```
# ceph osd crush tree
[
    {
        "id": -1,
        "name": "default",
        "type": "root",
        "type_id": 10,
        "items": [
            {
                "id": -2,
                "name": "ceph-1",
                "type": "host",
                "type_id": 1,
                "items": [
                    {
                        "id": 3,
                        "name": "osd.3",
                        "type": "osd",
                        "type_id": 0,
                        "crush_weight": 0.869095,
                        "depth": 2
                    },
                    {
                        "id": 4,
                        "name": "osd.4",
                        "type": "osd",
                        "type_id": 0,
                        "crush_weight": 0.869095,
                        "depth": 2
                    },
                    {
                        "id": 5,
                        "name": "osd.5",
```

```
                        "type": "osd",
                        "type_id": 0,
                        "crush_weight": 0.869095,
                        "depth": 2
                    },
                    {
                        "id": 1,
                        "name": "osd.1",
                        "type": "osd",
                        "type_id": 0,
                        "crush_weight": 0.908096,
                        "depth": 2
                    }
                ]
            },
......
        ]
    }
]
```

- 最顶层的是type为root的结点,其name为default。
- items数组中的是type为host的结点,其name一般是物理机的主机名
- items.items数组中是该host上osd结点的集合,osd的type名是osd

rados

列出所有的pool:

```
$ rados lspools
rbd
cephfs_data
cephfs_metadata
```

查看集群空间使用情况:

```
$ rados df
pool name                 KB      objects       clones     degra
ded      unfound           rd        rd KB           wr        w
r KB
cephfs_data     380800880276     94270278            0        14
108            0    571582261 290341154894   2864318094 44751299
26776
cephfs_metadata        51919        28504            0
 28            0     22382925    260490674     91157812    39323
0340
rbd              10819757349      2667363            0
  4            0     17695478   1764066746     95173535  1896134
5038
  total used    1180265382228     96966145
  total avail   925249168572
  total space   2105514550800
```

可以看到,集群的总对象数是96966145,总使用空间是1180265382228,平均对象大小是1180031080452/96949317=12171KB

空间使用情况看起来不是很直观,如果不需要查看总对象数,可以使用ceph df命令,看起来更直观一点:

```
$ ceph df
GLOBAL:
    SIZE      AVAIL     RAW USED     %RAW USED
    1960T      861T        1098T         56.04
POOLS:
    NAME                ID     USED       %USED     MAX AVAIL
  OBJECTS
    rbd                 0      10318G      1.54          166T
   2667363
    cephfs_data         3        354T     54.25          166T
  94253531
    cephfs_metadata     4      50683k         0          166T
     28518
```

rgw简介

rgw是rados gateway的简称，是ceph基于librados提供的对象存储服务。

常见概念:

- AccessKey: 长度为20的字符串，用来标识client的身份
- SecretKey: 长度为40的字符串，用来签名。
- Region: 区域，用来表示位置，不如北美、中国
- Bucket: 存储对象的容器，Bucket中只能存储object，不能存储bucket
- object: 对象，存储的基本单元
- key: object的名称

radosgw-admin命令

radosgw-admin命令用来管理rgw服务,比如用户管理、权限管理。

radosgw-admin命令不能查看某个bucket下的所有对象,s3cmd等工具可以。

用户管理

> 注意: radosgw-admin命令用户不能列出所有的用户,可以通过查看xxx.rgw.users.uid这个pool去查看所有的用户id。

创建用户

创建一个id为frank6866的用户

```
# radosgw-admin user create --uid=frank6866 --display-name=frank
6866
{
    "user_id": "frank6866",
    "display_name": "frank6866",
    "email": "",
    "suspended": 0,
    "max_buckets": 1000,
    "auid": 0,
    "subusers": [],
    "keys": [
        {
            "user": "frank6866",
            "access_key": "xxx",
            "secret_key": "xxx"
        }
    ],
    "swift_keys": [],
    "caps": [],
    "op_mask": "read, write, delete",
    "default_placement": "",
    "placement_tags": [],
    "bucket_quota": {
        "enabled": false,
        "max_size_kb": -1,
        "max_objects": -1
    },
    "user_quota": {
        "enabled": false,
        "max_size_kb": -1,
        "max_objects": -1
    },
    "temp_url_keys": []
}
```

查看用户信息

查看id为frank6866的用户信息

```
# radosgw-admin user info --uid=frank6866
{
    "user_id": "frank6866",
    "display_name": "frank6866",
    "email": "",
    "suspended": 0,
    "max_buckets": 1000,
    "auid": 0,
    "subusers": [],
    "keys": [
        {
            "user": "frank6866",
            "access_key": "xxx",
            "secret_key": "xxx"
        }
    ],
    "swift_keys": [],
    "caps": [],
    "op_mask": "read, write, delete",
    "default_placement": "",
    "placement_tags": [],
    "bucket_quota": {
        "enabled": false,
        "max_size_kb": -1,
        "max_objects": -1
    },
    "user_quota": {
        "enabled": false,
        "max_size_kb": -1,
        "max_objects": -1
    },
    "temp_url_keys": []
}
```

suspend用户

suspend id为frank6866的用户

```
# radosgw-admin user suspend --uid=frank6866
{
    "user_id": "frank6866",
    "display_name": "frank6866",
    "email": "",
    "suspended": 1,
    "max_buckets": 1000,
    "auid": 0,
    "subusers": [],
    "keys": [
        {
            "user": "frank6866",
            "access_key": "xxx",
            "secret_key": "xxx"
        }
    ],
    "swift_keys": [],
    "caps": [],
    "op_mask": "read, write, delete",
    "default_placement": "",
    "placement_tags": [],
    "bucket_quota": {
        "enabled": false,
        "max_size_kb": -1,
        "max_objects": -1
    },
    "user_quota": {
        "enabled": false,
        "max_size_kb": -1,
        "max_objects": -1
    },
    "temp_url_keys": []
}
```

suspend用户后，用户的suspended字段会变为1

enable用户

用户创建后默认处于enabled状态，在suspend用户后，可以重新enable用户

```
# radosgw-admin user enable --uid=frank6866
{
    "user_id": "frank6866",
    "display_name": "frank6866",
    "email": "",
    "suspended": 0,
    "max_buckets": 1000,
    "auid": 0,
    "subusers": [],
    "keys": [
        {
            "user": "frank6866",
            "access_key": "xxx",
            "secret_key": "xxx"
        }
    ],
    "swift_keys": [],
    "caps": [],
    "op_mask": "read, write, delete",
    "default_placement": "",
    "placement_tags": [],
    "bucket_quota": {
        "enabled": false,
        "max_size_kb": -1,
        "max_objects": -1
    },
    "user_quota": {
        "enabled": false,
        "max_size_kb": -1,
        "max_objects": -1
    },
    "temp_url_keys": []
}
```

用户配额管理

设置配置

设置用户frank6866的最大使用空间为10G

```
# radosgw-admin quota set --quota-scope=user --uid=frank6866 --m
ax-size=10G
# radosgw-admin user info --uid=frank6866
{
    "user_id": "frank6866",
    "display_name": "frank6866",
    "email": "",
    "suspended": 0,
    "max_buckets": 1000,
    "auid": 0,
    "subusers": [],
    "keys": [
        {
            "user": "frank6866",
            "access_key": "xxx",
            "secret_key": "xxx"
        }
    ],
    "swift_keys": [],
    "caps": [],
    "op_mask": "read, write, delete",
    "default_placement": "",
    "placement_tags": [],
    "bucket_quota": {
        "enabled": false,
        "max_size_kb": -1,
        "max_objects": -1
    },
    "user_quota": {
        "enabled": false,
        "max_size_kb": 10485760,
        "max_objects": -1
    },
    "temp_url_keys": []
}
```

启用配额

设置配额后，默认并没有启用，需要启用配额:

```
# radosgw-admin quota enable --quota-scope=user --uid=frank6866
# radosgw-admin user info --uid=frank6866
{
    "user_id": "frank6866",
    "display_name": "frank6866",
    "email": "",
    "suspended": 0,
    "max_buckets": 1000,
    "auid": 0,
    "subusers": [],
    "keys": [
        {
            "user": "frank6866",
            "access_key": "xxx",
            "secret_key": "xxx"
        }
    ],
    "swift_keys": [],
    "caps": [],
    "op_mask": "read, write, delete",
    "default_placement": "",
    "placement_tags": [],
    "bucket_quota": {
        "enabled": false,
        "max_size_kb": -1,
        "max_objects": -1
    },
    "user_quota": {
        "enabled": true,
        "max_size_kb": 10485760,
        "max_objects": -1
    },
    "temp_url_keys": []
}
```

删除用户

删除id为frank6866的用户

```
# radosgw-admin user rm --uid=frank6866
```

查看所有用户

```
# rados lspools | grep user
cn-bj-1.rgw.users.uid

# rados ls -p cn-bj-1.rgw.users.uid
c7772511-68d3-4d98-879f-b682f0623242
......
```

bucket管理

```
  bucket list               list buckets
  bucket link               link bucket to specified user
  bucket unlink             unlink bucket from specified user
  bucket stats              returns bucket statistics
  bucket rm                 remove bucket
  bucket check              check bucket index
  bucket reshard            reshard bucket
```

列出bucket

```
# radosgw-admin bucket list
[
    "my-new-bucket-912b8493-a4d2-4fe4-8a92-a9d873e10760"
]
```

查看bucket的状态

```
# radosgw-admin bucket stats
[
    {
        "bucket": "my-new-bucket-912b8493-a4d2-4fe4-8a92-a9d873e
10760",
        "pool": "1.rgw.buckets.data",
        "index_pool": "1.rgw.buckets.index",
        "id": "fa251bb9-e7a0-46da-9599-90ab1546155b.804872.19",
        "marker": "fa251bb9-e7a0-46da-9599-90ab1546155b.804872.1
9",
        "owner": "frank6866",
        "ver": "0#1,1#1,2#1,3#1,4#1,5#1,6#1,7#1,8#1,9#1,10#1,11#
1,12#1,13#1,14#1,15#1,16#1,17#1,18#1,19#1,20#1,21#1,22#1,23#1,24
#1,25#1,26#1,27#1,28#1,29#1,30#1,31#1,32#1,33#1,34#1,35#1,36#1,3
7#1,38#1,39#1,40#1,41#1,42#1,43#1,44#1,45#1,46#1,47#1,48#1,49#1,
```

```
50#1,51#1,52#1,53#3,54#1,55#1,56#1,57#1,58#1,59#1,60#1,61#1,62#1
,63#1,64#1,65#1,66#1,67#1,68#1,69#1,70#1,71#1,72#1,73#1,74#1,75#
1,76#1,77#1,78#1,79#3,80#1,81#1,82#1,83#1,84#1,85#1,86#1,87#1,88
#1,89#1,90#1,91#1,92#1,93#1,94#1,95#1,96#1,97#1,98#1,99#1",
        "master_ver": "0#0,1#0,2#0,3#0,4#0,5#0,6#0,7#0,8#0,9#0,1
0#0,11#0,12#0,13#0,14#0,15#0,16#0,17#0,18#0,19#0,20#0,21#0,22#0,
23#0,24#0,25#0,26#0,27#0,28#0,29#0,30#0,31#0,32#0,33#0,34#0,35#0
,36#0,37#0,38#0,39#0,40#0,41#0,42#0,43#0,44#0,45#0,46#0,47#0,48#
0,49#0,50#0,51#0,52#0,53#0,54#0,55#0,56#0,57#0,58#0,59#0,60#0,61
#0,62#0,63#0,64#0,65#0,66#0,67#0,68#0,69#0,70#0,71#0,72#0,73#0,7
4#0,75#0,76#0,77#0,78#0,79#0,80#0,81#0,82#0,83#0,84#0,85#0,86#0,
87#0,88#0,89#0,90#0,91#0,92#0,93#0,94#0,95#0,96#0,97#0,98#0,99#0
",
        "mtime": "2017-03-22 20:15:43.810956",
        "max_marker": "0#,1#,2#,3#,4#,5#,6#,7#,8#,9#,10#,11#,12#
,13#,14#,15#,16#,17#,18#,19#,20#,21#,22#,23#,24#,25#,26#,27#,28#
,29#,30#,31#,32#,33#,34#,35#,36#,37#,38#,39#,40#,41#,42#,43#,44#
,45#,46#,47#,48#,49#,50#,51#,52#,53#00000000002.53236.3,54#,55#,
56#,57#,58#,59#,60#,61#,62#,63#,64#,65#,66#,67#,68#,69#,70#,71#,
72#,73#,74#,75#,76#,77#,78#,79#00000000002.55367.3,80#,81#,82#,8
3#,84#,85#,86#,87#,88#,89#,90#,91#,92#,93#,94#,95#,96#,97#,98#,9
9#",
        "usage": {
            "rgw.main": {
                "size_kb": 1,
                "size_kb_actual": 4,
                "num_objects": 1
            },
            "rgw.multimeta": {
                "size_kb": 0,
                "size_kb_actual": 0,
                "num_objects": 1
            }
        },
        "bucket_quota": {
            "enabled": false,
            "max_size_kb": -1,
            "max_objects": -1
        }
    }
```

```
]
```

删除bucket

```
# radosgw-admin bucket rm my-new-bucket-912b8493-a4d2-4fe4-8a92-
a9d873e10760
```

s3cmd

s3cmd是一个管理s3中对象的命令行工具，除了可以管理s3中的对象，还可以管理兼容s3接口的存储，比如Ceph rgw。

安装

CentOS7

```
# pip install s3cmd
```

macOS

```
brew install s3cmd
```

配置

生成配置文件，存储到rgw-26.cfg文件中

```
# s3cmd --configure -c ~/.s3cfg
```

~/.s3cfg文件需要修改的地方有两个

```
host_base = 10.10.10.10:7480
host_bucket = 10.10.10.10:7480/%(bucket)s
```

使用

创建bucket

创建一个名为b-frank6866的bucket

```
# s3cmd mb s3://b-frank6866
Bucket 's3://b-frank6866/' created
```

列出bucket

```
# s3cmd ls
16:11  s3://b-frank6866
```

上传对象

```
# s3cmd put /tmp/test.txt s3://b-frank6866
upload: '/tmp/test.txt' -> 's3://b-frank6866/test.txt'  [1 of 1]
 5 of 5   100% in    0s   121.90 B/s  done
```

> 上传时，使用-P选项可以将对象的权限为public

```
# s3cmd put -P /tmp/test.txt s3://b-frank6866
```

查看bucket中的对象

```
# s3cmd ls s3://b-frank6866
16:22         5   s3://b-frank6866/test.txt
```

输出的单位是Byte，s3://后面和之后的/之间是bucket的名称，最后面是对象的名称。

查看bucket中对象大小

以byte为单位查看对象的大小

```
# s3cmd du s3://test_lifecycle
2902467869 5 objects s3://test_lifecycle/
```

以容易阅读的方式查看对象大小，注意会去尾，大小不大是误差较大

```
# s3cmd du -H s3://test_lifecycle
2G       5 objects s3://test_lifecycle/
```

下载对象

```
# s3cmd get s3://b-frank6866/test.txt /tmp/test2.txt
```

删除对象

```
s3cmd del s3://bucket/object
```

批量上传文件

```
s3cmd put ./*.txt s3://b-frank6866
```

上传目录

```
s3cmd sync LOCAL_DIR s3://bucket[/prefix]
```

下载目录

```
s3cmd sync s3://bucket[/prefix] LOCAL_DIR
```

删除**bucket**

```
s3cmd rb s3://bucketname
```

S3简介

地址: https://aws.amazon.com/cn/s3/

官网介绍如下:

> Amazon Simple Storage Service (Amazon S3) 是一种对象存储，它具有简单的 Web 服务接口，可用于在 Web 上的任何位置存储和检索任意数量的数据。它能够提供 99.999999999% 的持久性，并且可以在全球大规模传递数万亿对象。

注意：官网介绍的**11**个**9**的表示的是数据的持久性，就是数据不会丢；并不表示数据会有**11**个**9**的可访问性。

s3基本概念：

- access_key、secret_key：认证需要的密钥对，由Ceph管理人员提供，需要妥善保存。
- bucket：存储桶，类似于顶层容器的概念，在使用前需要创建，bucket名称在一个集群中唯一。
- perfix：前缀，类似于底层文件夹概念，可以用于区分同名对象。

对比

公司	产品	数据持久性	月可靠性	年可靠性
AWS	S3(Simple Storage Service)	99.999999999%(11个9)	99.9%	99.99%
阿里云	OSS(Object Storage Service)	99.9999999%(9个9)	99.9%	
腾讯云	COS(Cloud Object Storage)	99.999999999%(11个9)		

S3的SLA规则

官网介绍: https://aws.amazon.com/cn/s3/sla/

相关概念

- 错误率: S3每隔5分钟就会计算一次错误率，计算公式为，错误率=5xx响应数(5分钟内)/总响应数(5分钟内)
- 每月服务可用率(monthly_uptime_percentage): 每个月的错误率(monthly_error_percentage)为该月每5分钟错误率的平均值，monthly_uptime_percentage =100%-monthly_error_percentage

补偿

- 如果monthly_uptime_percentage>=99.9%，服务正常
- 如果99.0%<=monthly_uptime_percentage<99.9%，赔偿一个月账单的10%
- 如果monthly_uptime_percentage<99.0%，赔偿一个月账单的25%

S3 java sdk demo

注意aws-java-sdk使用1.8.11版本

```
package com.opscloud.tutorial;

import com.amazonaws.ClientConfiguration;
import com.amazonaws.Protocol;
import com.amazonaws.auth.AWSCredentials;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.model.*;
import com.amazonaws.util.IOUtils;
import com.amazonaws.util.StringUtils;
import com.amazonaws.services.s3.transfer.TransferManager;
import com.amazonaws.services.s3.transfer.Upload;
import com.amazonaws.AmazonClientException;

import java.io.ByteArrayInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.List;
import java.util.UUID;
import java.io.PrintWriter;

public class CephS3Tutorial {
    // Please change it in your testing environment
    private static String objectFilePathForTesting = "/tmp/test.
txt";

    public static void main(String[] args) throws Exception {
        String accessKey = "xxx";
        String secretKey = "xxx";

        ClientConfiguration clientConfig = new ClientConfigurati
on();
```

```
        clientConfig.setProtocol(Protocol.HTTP);

        AWSCredentials credentials = new BasicAWSCredentials(acc
essKey, secretKey);
        AmazonS3 conn = new AmazonS3Client(credentials, clientCo
nfig);
        conn.setEndpoint("10.10.10.20:7480");

        uploadMultipart(conn);

//        propareFile();
//        listBucketsAndObjects(conn);

    }

    public static void uploadMultipart(AmazonS3 conn) throws Int
erruptedException {
        String bucketName = "test_lifecycle";
        String objectName = "multipart-test-" + UUID.randomUUID(
).toString();
        String filePath = "/tmp/bigfile.data";

        TransferManager tm = new TransferManager(conn);
        Upload upload = tm.upload(bucketName, objectName, new Fi
le(filePath));

        try {
            // Or you can block and wait for the upload to finis
h
            upload.waitForCompletion();
            System.out.println("Upload complete.");
            tm.shutdownNow();
        } catch (AmazonClientException amazonClientException) {
            System.out.println("Unable to upload file, upload wa
s aborted.");
            amazonClientException.printStackTrace();
        }

    }
```

```
    public static void propareFile() throws IOException {
        try{
            PrintWriter writer = new PrintWriter(objectFilePathF
orTesting, "UTF-8");
            writer.println("The first line");
            writer.println("The second line");
            writer.close();
        } catch (IOException e) {
            throw e;
        }
    }

    public static void createBucketAndObjects(AmazonS3 conn) thr
ows IOException {
        String uuid = UUID.randomUUID().toString();
        String bucketName = "my-new-bucket-" + uuid;
        // 1. create bucket
        Bucket bucket = conn.createBucket(bucketName);
        System.out.println("bucket created with name: " + bucket
.getName());

        // 2. put object in bucket
        String objectName = "object-" + UUID.randomUUID().toStri
ng();
        byte[] contentBytes = IOUtils.toByteArray(new FileInputS
tream(new File(objectFilePathForTesting)));
        Long contentLength = Long.valueOf(contentBytes.length);
        ByteArrayInputStream byteArrayInputStream = new ByteArra
yInputStream(contentBytes);

        ObjectMetadata metadata = new ObjectMetadata();
        metadata.setContentLength(contentLength);

        conn.putObject(bucket.getName(), objectName, byteArrayIn
putStream, metadata);

        // 3. GENERATE OBJECT DOWNLOAD URLS
        GeneratePresignedUrlRequest request = new GeneratePresig
nedUrlRequest(bucket.getName(), objectName);
        System.out.println("download url is:  " + conn.generateP
```

```
resignedUrl(request));

        // 4. list bucket and object
        listBucketsAndObjects(conn);

        // 5. delete created object
        // conn.deleteObject(bucket.getName(), objectName);

        // 6. delete bucket(bucket must be empty)
        // conn.deleteBucket(bucket.getName());
    }

    public static void listBucketsAndObjects(AmazonS3 conn) {
        System.out.println("\n===========list buckets and object
s");
        List<Bucket> buckets = conn.listBuckets();
        for (Bucket bucket : buckets) {
            System.out.println(bucket.getName() + "\t" +
                    StringUtils.fromDate(bucket.getCreationDate(
)));

            ObjectListing objects = conn.listObjects(bucket.getN
ame());
            do {
                for (S3ObjectSummary objectSummary : objects.get
ObjectSummaries()) {
                    System.out.println("\t\t" + objectSummary.ge
tKey() + "\t" +
                            objectSummary.getSize() + "\t" +
                            StringUtils.fromDate(objectSummary.g
etLastModified()));
                }
                objects = conn.listNextBatchOfObjects(objects);
            } while (objects.isTruncated());

        }
    }


}
```

异常

创建bucket命令如下,

```
# s3cmd mb s3://b-frank6866
```

错误信息如下:

```
gaierror: [Errno 8] nodename nor servname provided, or not known
```

异常原因,.s3cfg文件中,%(bucket)写在主机名前面了。

```
host_bucket = %(bucket)s.ceph-rgw-1:7480
```

在ceph rgw中,%(bucket)应该写在主机名后面,如下:

```
host_bucket = ceph-rgw-1:7480/%(bucket)
```

OSS简介

OSS(Object Storage Service)是阿里云提供的对象存储服务。分为三种存储类型:

- 标准类型
- 低频访问类型
- 归档类型

cosbench

cosbench是对对象存储进行基准测试的工具。

github地址: https://github.com/intel-cloud/cosbench

安装

cosbench使用java开发,从github上下载源码包。

```
# yum install java-1.8.0-openjdk

# wget https://github.com/intel-cloud/cosbench/releases/download
/v0.4.2.c4/0.4.2.c4.zip
# unzip v0.4.2.c4.zip

# cd 0.4.2.c4/

# chmod u+x *.sh

# unset http_proxy

# ./start-all.sh
Launching osgi framwork ...
Successfully launched osgi framework!
Booting cosbench driver ...
.
Starting    cosbench-log_0.4.2    [OK]
Starting    cosbench-tomcat_0.4.2    [OK]
......
......
Successfully started cosbench driver!
Listening on port 0.0.0.0/0.0.0.0:18089 ...
Persistence bundle starting...
Persistence bundle started.
---------------------------------------------
!!! Service will listen on web port: 18088 !!!
```

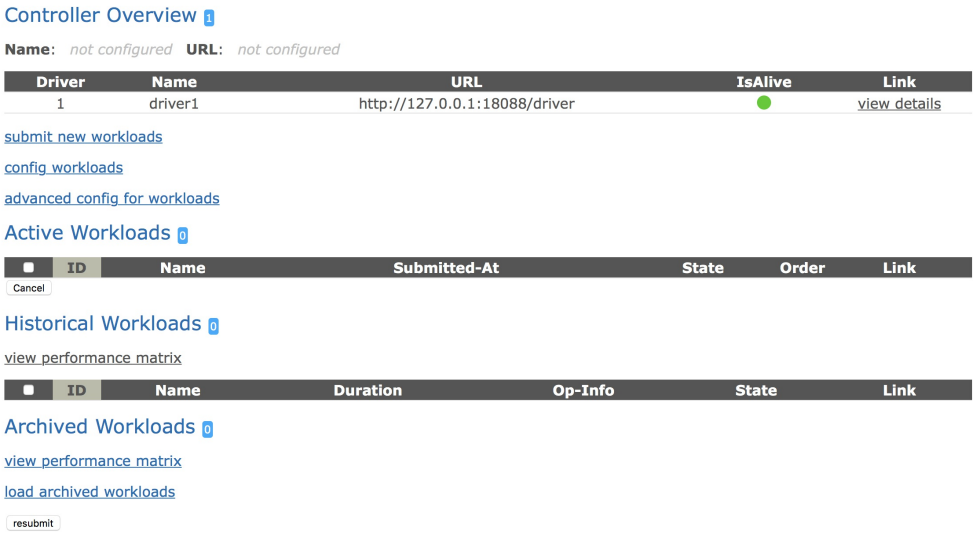
```
-----------------------------------------

=========================================================

Launching osgi framwork ...
Successfully launched osgi framework!
Booting cosbench controller ...
.
Starting    cosbench-log_0.4.2    [OK]
......
......
Starting    cosbench-controller-web_0.4.2    [OK]
Successfully started cosbench controller!
Listening on port 0.0.0.0/0.0.0.0:19089 ...
Persistence bundle starting...
Persistence bundle started.
-----------------------------------------
!!! Service will listen on web port: 19088 !!!
-----------------------------------------
```

查看18088和19088是否在监听:

```
# ss -ntpl | grep java
LISTEN     0      100          *:19088                    *:*
                users:(("java",pid=18306,fd=88))
LISTEN     0      50           *:19089                    *:*
                users:(("java",pid=18306,fd=18))
LISTEN     0      100          *:18088                    *:*
                users:(("java",pid=18137,fd=101))
LISTEN     0      50           *:18089                    *:*
                users:(("java",pid=18137,fd=18))
```

在浏览器中打开: http://ip:19088/controller/index.html

如下图:



作业管理

提交作业

```
# ./cli.sh submit conf/workload-config.xml
Accepted with ID: w1
```

查看作业信息

```
# ./cli.sh info
Drivers:
driver1    http://127.0.0.1:18088/driver
Total: 1 drivers

Active Workloads:
w1 PROCESSING    s2-prepare
Total: 1 active workloads
```

异常

1.启动失败

启动时失败,错误信息如下:

```
# ./start-all.sh
Launching osgi framwork ...
Successfully launched osgi framework!
Booting cosbench driver ...
Ncat: Connection refused.
.Ncat: Connection refused.
```

查看日志,如下:

```
# tail -f log/driver-boot.log
Error: Could not find or load main class org.eclipse.equinox.lau
ncher.Main
```

原因: 下载了错误的文件,在https://github.com/intel-cloud/cosbench/releases页面下载,而不是在github代码页下载。

rbd简介

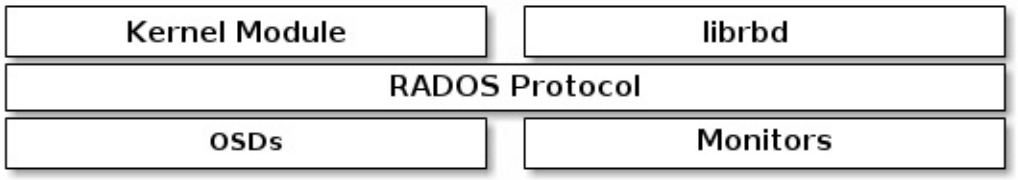
block是一串连续的字节，基于block的存储是存储数据最普遍的方式，其数据读写的最小单位是block，比如硬盘、CD、软盘等。

比如查看硬盘分区的块大小:

```
# blockdev --getbsz /dev/sdb1
512
# blockdev --getbsz /dev/sdb2
4096
```

/dev/sdb1分区的块大小是512Bytes，/dev/sdb2分区的块大小是4096Bytes.

Ceph block device的配置很精简，容量很容易扩展或变更，并且Ceph block device将数据条带化地存储在Ceph集群的多个OSD中，可以并发地读写。Ceph block device使用了RADOS的快照、副本和一致性等特点。**Ceph**的**RADOS Block Device(RBD)**通过**Linux kernel**中的**module**或者**librbd**库来和**OSD**进行交互，如下图:



rbd命令

通过rbd命令，我们可以创建、列出、查看或者移除块设备镜像，还可以克隆镜像、创建快照、查看快照、恢复快照等。

> 在ceph中，一个rbd设备称为image(镜像)

创建块设备

名为"rbd"的pool是Ceph中默认用来存放块设备的pool，使用rbd create命令默认会在rbd这个pool中创建块设备。

```
# rbd create --size 512 frank6866
```

如果需要在指定的pool，比如pool-frank6866这个pool中创建块设备，可以使用pool名作为前缀

```
# rbd create --size 512 pool-frank6866/frank6866
```

> --size的单位是MBytes

列出块设备

名为"rbd"的pool是Ceph中默认用来存放块设备的pool，使用rbd ls命令默认会列出rbd这个pool中的块设备。

```
# rbd ls
frank6866
```

查看pool-frank6866这个pool中的块设备:

```
# rbd ls pool-frank6866
frank6866
```

查看块设备的详细信息

```
# rbd info pool-frank6866/frank6866
rbd image 'frank6866':
    size 512 MB in 128 objects
    order 22 (4096 kB objects)
    block_name_prefix: rbd_data.1ad6e2ae8944a
    format: 2
    features: layering, exclusive-lock, object-map, fast-diff, d
eep-flatten
    flags:
```

- size: 512 MB in 128 objects，表示该块设备的大小是512MBytes，分布在128个objects中，RBD默认的块大小是4MBytes
- order: order 22 (4096 kB objects)，指定RBD在OSD中存储时的block size，block size的计算公式是1<<order(单位是bytes)，在本例中order=22，所有block size是1<<22bytes，也就是4096KBytes。order的默认值是22，RBD在OSD中默认的对象大小是4MBytes
- format: rbd image的格式，format1已经过期了。现在默认都是format2，被librbd和kernel3.11后面的版本支持
- block_name_prefix: 表示这个image在pool中的名称前缀，可以通过rados -p pool-frank6866 ls | grep rbd_data.1ad6e2ae8944a命令查看这个rbd image在rados中的所有object。但是要注意的是，刚创建的image，如果里面没有数据，不会在rados中创建object，只有写入数据时才会有。size字段中的objects数量表示的是最大的objects数量

调整块设备的大小

调大块设备的大小:

```
# rbd resize --size 1024 pool-frank6866/frank6866
Resizing image: 100% complete...done.

# rbd info pool-frank6866/frank6866
rbd image 'frank6866':
    size 1024 MB in 256 objects
    order 22 (4096 kB objects)
    block_name_prefix: rbd_data.1ad6e2ae8944a
    format: 2
    features: layering, exclusive-lock, object-map, fast-diff, d
eep-flatten
    flags:
```

减小块设备的大小:

```
# rbd resize --size 256 pool-frank6866/frank6866 --allow-shrink
Resizing image: 100% complete...done.

# rbd info pool-frank6866/frank6866
rbd image 'frank6866':
    size 256 MB in 64 objects
    order 22 (4096 kB objects)
    block_name_prefix: rbd_data.1ad6e2ae8944a
    format: 2
    features: layering, exclusive-lock, object-map, fast-diff, d
eep-flatten
    flags:
```

删除块设备

```
# rbd rm pool-frank6866/frank6866
Removing image: 100% complete...done.

# rbd ls pool-frank6866
```

Linux挂载RBD

以H版为例，对于RBD，如果使用Ceph Kernel Client，CentOS6内核在2.6.32及以上，CentOS7内核在3.10及以上。

对于需要挂载RBD的Linux系统，下面我们称为client,其主机名为rbd-client。

1.在client上安装ceph

client上需要安装ceph才能挂载RBD，我们可以通过ceph-deploy工具在client上安装ceph。

在ceph-deploy结点执行如下命令:

```
# ceph-deploy install rbd-client
```

2.将配置文件推送到client

在ceph-deploy结点执行如下命令:

```
# ceph-deploy admin rbd-client
```

3.创建rbd image

在client结点上执行如下命令:

```
# rbd create pool-frank6866/frank6866 --size 1024 --image-featur
e=layering
```

> 注意,创建rbd的时候,使用--image-feature=layering选项,不然在CentOS7上使用rbd map的时候会报"RBD image feature set mismatch"错误.

4.将rbd image映射到块设备

```
# rbd map pool-frank6866/frank6866
/dev/rbd0
```

```
# lsblk
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
......
rbd0   252:0    0     1G  0 disk
```

5.创建文件系统

```
# mkfs.ext4 /dev/rbd0
```

6.挂载rbd设备

```
# mkdir /mnt/rbd
# mount /dev/rbd0 /mnt/rbd/

# df -lTh
Filesystem     Type      Size  Used Avail Use% Mounted on
......
/dev/rbd0      ext4      976M  2.6M  907M   1% /mnt/rbd
```

Ceph监控指标

ceph的监控数据存在telegraf库的ceph表中，查看该表的字段列表:

```
> show field keys on telegraf from ceph
name: ceph
fieldKey                                    fieldType
--------                                    ---------
accept_timeout                              float
activating_latency.avgcount                 float
activating_latency.sum                      float
active_latency.avgcount                     float
active_latency.sum                          float
agent_evict                                 float
agent_flush                                 float
agent_skip                                  float
agent_wake                                  float
apply_latency.avgcount                      float
apply_latency.sum                           float
backfilling_latency.avgcount                float
backfilling_latency.sum                     float
begin                                       float
begin_bytes.avgcount                        float
begin_bytes.sum                             float
begin_keys.avgcount                         float
begin_keys.sum                              float
begin_latency.avgcount                      float
begin_latency.sum                           float
buffer_bytes                                float
bytes                                       float
bytes_dirtied                               float
bytes_wb                                    float
cached_crc                                  float
cached_crc_adjusted                         float
clean_latency.avgcount                      float
clean_latency.sum                           float
collect                                     float
collect_bytes.avgcount                      float
```

collect_bytes.sum	float
collect_keys.avgcount	float
collect_keys.sum	float
collect_latency.avgcount	float
collect_latency.sum	float
collect_timeout	float
collect_uncommitted	float
command_active	float
command_resend	float
command_send	float
commit	float
commit_bytes.avgcount	float
commit_bytes.sum	float
commit_keys.avgcount	float
commit_keys.sum	float
commit_latency.avgcount	float
commit_latency.sum	float
commitcycle	float
commitcycle_interval.avgcount	float
commitcycle_interval.sum	float
commitcycle_latency.avgcount	float
commitcycle_latency.sum	float
committing	float
complete_latency.avgcount	float
complete_latency.sum	float
copyfrom	float
election_call	float
election_lose	float
election_win	float
get	float
get_or_fail_fail	float
get_or_fail_success	float
get_sum	float
getinfo_latency.avgcount	float
getinfo_latency.sum	float
getlog_latency.avgcount	float
getlog_latency.sum	float
getmissing_latency.avgcount	float
getmissing_latency.sum	float
heartbeat_to_peers	float

```
history_alloc_Mbytes                          float
history_alloc_num                             float
incomplete_latency.avgcount                   float
incomplete_latency.sum                        float
initial_latency.avgcount                      float
initial_latency.sum                           float
inodes_dirtied                                float
inodes_wb                                     float
ios_dirtied                                   float
ios_wb                                        float
journal_bytes                                 float
journal_full                                  float
journal_latency.avgcount                      float
journal_latency.sum                           float
journal_ops                                   float
journal_queue_bytes                           float
journal_queue_ops                             float
journal_wr                                    float
journal_wr_bytes.avgcount                     float
journal_wr_bytes.sum                          float
lease_ack_timeout                             float
lease_timeout                                 float
leveldb_compact                               float
leveldb_compact_queue_len                     float
leveldb_compact_queue_merge                   float
leveldb_compact_range                         float
leveldb_get                                   float
leveldb_get_latency.avgcount                  float
leveldb_get_latency.sum                       float
leveldb_submit_latency.avgcount               float
leveldb_submit_latency.sum                    float
leveldb_submit_sync_latency.avgcount          float
leveldb_submit_sync_latency.sum               float
leveldb_transaction                           float
linger_active                                 float
linger_ping                                   float
linger_resend                                 float
linger_send                                   float
loadavg                                       float
map_epoch                                     float
```

map_full	float
map_inc	float
map_message_epoch_dups	float
map_message_epochs	float
map_messages	float
max	float
mds_epoch	float
messages_delayed_for_map	float
new_pn	float
new_pn_latency.avgcount	float
new_pn_latency.sum	float
notbackfilling_latency.avgcount	float
notbackfilling_latency.sum	float
num_bytes	float
num_elections	float
num_mds_failed	float
num_mds_in	float
num_mds_up	float
num_mon	float
num_mon_quorum	float
num_object	float
num_object_degraded	float
num_object_misplaced	float
num_object_unfound	float
num_osd	float
num_osd_in	float
num_osd_up	float
num_pg	float
num_pg_active	float
num_pg_active_clean	float
num_pg_peering	float
num_pool	float
num_sessions	float
numpg	float
numpg_primary	float
numpg_replica	float
numpg_stray	float
object_ctx_cache_hit	float
object_ctx_cache_total	float
omap_del	float

omap_rd	float
omap_wr	float
op	float
op_ack	float
op_active	float
op_cache_hit	float
op_commit	float
op_in_bytes	float
op_laggy	float
op_latency.avgcount	float
op_latency.sum	float
op_out_bytes	float
op_pg	float
op_prepare_latency.avgcount	float
op_prepare_latency.sum	float
op_process_latency.avgcount	float
op_process_latency.sum	float
op_queue_bytes	float
op_queue_max_bytes	float
op_queue_max_ops	float
op_queue_ops	float
op_r	float
op_r_latency.avgcount	float
op_r_latency.sum	float
op_r_out_bytes	float
op_r_prepare_latency.avgcount	float
op_r_prepare_latency.sum	float
op_r_process_latency.avgcount	float
op_r_process_latency.sum	float
op_resend	float
op_rmw	float
op_rw	float
op_rw_in_bytes	float
op_rw_latency.avgcount	float
op_rw_latency.sum	float
op_rw_out_bytes	float
op_rw_prepare_latency.avgcount	float
op_rw_prepare_latency.sum	float
op_rw_process_latency.avgcount	float
op_rw_process_latency.sum	float

op_rw_rlat.avgcount	float
op_rw_rlat.sum	float
op_send	float
op_send_bytes	float
op_w	float
op_w_in_bytes	float
op_w_latency.avgcount	float
op_w_latency.sum	float
op_w_prepare_latency.avgcount	float
op_w_prepare_latency.sum	float
op_w_process_latency.avgcount	float
op_w_process_latency.sum	float
op_w_rlat.avgcount	float
op_w_rlat.sum	float
op_wip	float
ops	float
osd_bytes	float
osd_bytes_avail	float
osd_bytes_used	float
osd_epoch	float
osd_laggy	float
osd_session_close	float
osd_session_open	float
osd_sessions	float
osd_tier_flush_lat.avgcount	float
osd_tier_flush_lat.sum	float
osd_tier_promote_lat.avgcount	float
osd_tier_promote_lat.sum	float
osd_tier_r_lat.avgcount	float
osd_tier_r_lat.sum	float
osdop_append	float
osdop_call	float
osdop_clonerange	float
osdop_cmpxattr	float
osdop_create	float
osdop_delete	float
osdop_getxattr	float
osdop_mapext	float
osdop_notify	float
osdop_other	float

osdop_pgls	float
osdop_pgls_filter	float
osdop_read	float
osdop_resetxattrs	float
osdop_rmxattr	float
osdop_setxattr	float
osdop_sparse_read	float
osdop_src_cmpxattr	float
osdop_stat	float
osdop_tmap_get	float
osdop_tmap_put	float
osdop_tmap_up	float
osdop_truncate	float
osdop_watch	float
osdop_write	float
osdop_writefull	float
osdop_zero	float
peering_latency.avgcount	float
peering_latency.sum	float
poolop_active	float
poolop_resend	float
poolop_send	float
poolstat_active	float
poolstat_resend	float
poolstat_send	float
primary_latency.avgcount	float
primary_latency.sum	float
pull	float
push	float
push_in	float
push_in_bytes	float
push_out_bytes	float
put	float
put_sum	float
queue_len	float
queue_transaction_latency_avg.avgcount	float
queue_transaction_latency_avg.sum	float
recovered_latency.avgcount	float
recovered_latency.sum	float
recovering_latency.avgcount	float

recovering_latency.sum	float
recovery_ops	float
refresh	float
refresh_latency.avgcount	float
refresh_latency.sum	float
replicaactive_latency.avgcount	float
replicaactive_latency.sum	float
repnotrecovering_latency.avgcount	float
repnotrecovering_latency.sum	float
reprecovering_latency.avgcount	float
reprecovering_latency.sum	float
repwaitbackfillreserved_latency.avgcount	float
repwaitbackfillreserved_latency.sum	float
repwaitrecoveryreserved_latency.avgcount	float
repwaitrecoveryreserved_latency.sum	float
reset_latency.avgcount	float
reset_latency.sum	float
restart	float
session_add	float
session_rm	float
session_trim	float
share_state	float
share_state_bytes.avgcount	float
share_state_bytes.sum	float
share_state_keys.avgcount	float
share_state_keys.sum	float
start_latency.avgcount	float
start_latency.sum	float
start_leader	float
start_peon	float
started_latency.avgcount	float
started_latency.sum	float
stat_bytes	float
stat_bytes_avail	float
stat_bytes_used	float
statfs_active	float
statfs_resend	float
statfs_send	float
store_state	float
store_state_bytes.avgcount	float

```
store_state_bytes.sum                          float
store_state_keys.avgcount                      float
store_state_keys.sum                           float
store_state_latency.avgcount                   float
store_state_latency.sum                        float
stray_latency.avgcount                         float
stray_latency.sum                              float
subop                                          float
subop_in_bytes                                 float
subop_latency.avgcount                         float
subop_latency.sum                              float
subop_pull                                     float
subop_pull_latency.avgcount                    float
subop_pull_latency.sum                         float
subop_push                                     float
subop_push_in_bytes                            float
subop_push_latency.avgcount                    float
subop_push_latency.sum                         float
subop_w                                        float
subop_w_in_bytes                               float
subop_w_latency.avgcount                       float
subop_w_latency.sum                            float
take                                           float
take_sum                                       float
tier_clean                                     float
tier_delay                                     float
tier_dirty                                     float
tier_evict                                     float
tier_flush                                     float
tier_flush_fail                                float
tier_promote                                   float
tier_proxy_read                                float
tier_proxy_write                               float
tier_try_flush                                 float
tier_try_flush_fail                            float
tier_whiteout                                  float
val                                            float
wait.avgcount                                  float
wait.sum                                       float
waitactingchange_latency.avgcount              float
```

```
waitactingchange_latency.sum                 float
waitlocalbackfillreserved_latency.avgcount  float
waitlocalbackfillreserved_latency.sum       float
waitlocalrecoveryreserved_latency.avgcount  float
waitlocalrecoveryreserved_latency.sum       float
waitremotebackfillreserved_latency.avgcount float
waitremotebackfillreserved_latency.sum      float
waitremoterecoveryreserved_latency.avgcount float
waitremoterecoveryreserved_latency.sum      float
waitupthru_latency.avgcount                 float
waitupthru_latency.sum                      float
```

Ceph空间使用监控

```
ceph_usage

bytes_used (float)
kb_used (float)
max_avail (float)
objects (float)
```

Ceph日志简介

Ceph的日志文件保存在/var/log/ceph路径下，包含以下类型的日志:

- audit日志: 审计日志，文件名是ceph.audit.log
- 集群日志: 集群的信息，文件名是ceph.log
- mds日志: mds的日志，命名格式是ceph-mds.\.log
- mon日志: monitor进程的日志，命名格式是ceph-mon.\.log
- osd日志: osd进程的日志，命名格式是ceph-osd.\.log

clock skew detected

ceph的健康状态为HEALTH_WARN,提示"Monitor clock skew detected":

```
# ceph -s
    cluster c712f08b-c001-4b1c-969d-abec240138f7
     health HEALTH_WARN
            clock skew detected on mon.frank-ceph-2, mon.frank-c
eph-3
            Monitor clock skew detected
......
```

skew是偏离的意思,ceph的各个节点上时间需要一致,提示"Monitor clock skew detected"表示节点上的时间不一致。

解决思路: 检查ntp服务状态