CEPH命令手册

目录

[CEPH命令手册 1](#_Toc406568822)

[1 Ceph-deploy命令 6](#_Toc406568823)

[1.1 创建集群监控节点信息并生成集群配置文件(ceph.conf) 6](#_Toc406568824)

[1.2 初始化集群监控节点 6](#_Toc406568825)

[1.3 从各节点上收集集群相关配置文件 6](#_Toc406568826)

[1.4 准备OSD数据节点 6](#_Toc406568827)

[1.5 激活OSD数据节点 6](#_Toc406568828)

[1.6 分发集群配置文件到各个节点上 6](#_Toc406568829)

[1.7 创建MDS元数据服务器（文件存储专用） 6](#_Toc406568830)

[1.8 列出节点上磁盘信息的命令 7](#_Toc406568831)

[1.9 在节点上安装ceph 7](#_Toc406568832)

[1.10 从节点上删除ceph 7](#_Toc406568833)

[1.11 从节点上删除ceph的配置信息 7](#_Toc406568834)

[1.12 清除集群中的认证信息 7](#_Toc406568835)

[2 Ceph命令 7](#_Toc406568836)

[2.1 Global 7](#_Toc406568837)

[集群进程操作命令 7](#_Toc406568838)

[1. 查看ceph进程名称 7](#_Toc406568839)

[2. 启动所有ceph守护进程 8](#_Toc406568840)

[3. 停止所有ceph守护进程 8](#_Toc406568841)

[集群状态查询命令 9](#_Toc406568842)

[1. 集群健康状况查询 9](#_Toc406568843)

[2. 集群详情 9](#_Toc406568844)

[3. 集群ID查询 9](#_Toc406568845)

[4. 集群认证列表查询 9](#_Toc406568846)

[5. 集群磁盘使用状况查询 10](#_Toc406568847)

[集群内存相关命令 10](#_Toc406568848)

[1. 集群内存使用状况查询 10](#_Toc406568849)

[2. 释放集群节点中空闲的内存 11](#_Toc406568850)

[2.2 Auth 11](#_Toc406568851)

[2.3 Config-Key 11](#_Toc406568852)

[2.4 Mon 11](#_Toc406568853)

[MON 状态查询命令 11](#_Toc406568854)

[1. ceph mon stat命令查询mon map信息 11](#_Toc406568855)

[2.5 Mds 12](#_Toc406568856)

[2.6 Osd 12](#_Toc406568857)

[OSD状态查询命令 12](#_Toc406568858)

[1. ceph osd stat命令查询osd map信息 12](#_Toc406568859)

[2. ceph osd dump命令查询osd详细信息 12](#_Toc406568860)

[3. ceph osd tree命令查询crush树 13](#_Toc406568861)

[2.7 pool 13](#_Toc406568862)

[Pool 13](#_Toc406568863)

[1. 创建cache pool 13](#_Toc406568864)

[2. 删除cache pool 15](#_Toc406568865)

[3. show cache pool 15](#_Toc406568866)

[4. 设置cache pool的参数和值 18](#_Toc406568867)

[5. 设置cache pool中的对象或字节数的配额 23](#_Toc406568868)

[6. 给cache pool制作快照 23](#_Toc406568869)

[7. 给cache pool重命名 24](#_Toc406568870)

[8. 从cache pool中删除快照 24](#_Toc406568871)

[9. 获取cache pool的状态 25](#_Toc406568872)

[10. 获取cache pool的信息列表 25](#_Toc406568873)

[11. 在与指定的object对应的pool中查找pg 25](#_Toc406568874)

[Cache tierring 26](#_Toc406568875)

[1. 添加second tier pool到base cache pool，建立一个非空的tierpool 26](#_Toc406568876)

[2. 添加second tier pool到base cache pool，指定tierpool的大小 27](#_Toc406568877)

[3. 为tire pool指定cache模式 27](#_Toc406568878)

[4. 从base tier pool移除second tier pool 28](#_Toc406568879)

[5. 从base tier pool移除overlay pool 29](#_Toc406568880)

[6. 给base pool设置overlay pool 29](#_Toc406568881)

[Erasure 30](#_Toc406568882)

[创建erasure profile，plugin=jerasure 30](#_Toc406568883)

[创建erasure profile，plugin=isa 31](#_Toc406568884)

[创建erasure profile 32](#_Toc406568885)

[查看erasure profile 33](#_Toc406568886)

[2.8 Pg 34](#_Toc406568887)

[3 Rados命令 34](#_Toc406568888)

[3.1 pool 34](#_Toc406568889)

[显示当前所有pool 34](#_Toc406568890)

[创建pool，可以指定owner\_auid和crush\_rule 34](#_Toc406568891)

[复制pool 34](#_Toc406568892)

[删除pool 34](#_Toc406568893)

[显示pool使用情况 35](#_Toc406568894)

[显示pool中的object 35](#_Toc406568895)

[修改pool的owner auid 35](#_Toc406568896)

[3.2 object 35](#_Toc406568897)

[获取obj从指定pool 35](#_Toc406568898)

[放入obj到指定pool 35](#_Toc406568899)

[截断obj 35](#_Toc406568900)

[创建obj，可以指定类型（NONE，MAIN，SHADOW，MULTIMETA） 35](#_Toc406568901)

[删除obj 35](#_Toc406568902)

[复制obj 36](#_Toc406568903)

[克隆obj的数据 36](#_Toc406568904)

[显示obj的附加属性 36](#_Toc406568905)

[获取obj的附加属性，需要指定attr，可由listxattr获取 36](#_Toc406568906)

[设置obj的附加属性 36](#_Toc406568907)

[删除obj附加属性 36](#_Toc406568908)

[显示obj状态 36](#_Toc406568909)

[额外map信息 36](#_Toc406568910)

[显示快照信息 36](#_Toc406568911)

[创建快照 37](#_Toc406568912)

[删除快照 37](#_Toc406568913)

[回滚obj到指定快照 37](#_Toc406568914)

[显示对象的快照 37](#_Toc406568915)

[列出omapkeys 37](#_Toc406568916)

[列出omap keys 和 vals 37](#_Toc406568917)

[获取omap 具体key的值 37](#_Toc406568918)

[设置omap具体key的值 37](#_Toc406568919)

[删除omap 37](#_Toc406568920)

[获取omap头 38](#_Toc406568921)

[设置omap头 38](#_Toc406568922)

[转换tmap到omap 38](#_Toc406568923)

[显示obj的观察者 38](#_Toc406568924)

[设置set-alloc-hint 38](#_Toc406568925)

[3.3 导入和导出 38](#_Toc406568926)

[3.4 锁相关 39](#_Toc406568927)

[3.5 cache pools 39](#_Toc406568928)

[3.6 bench，测试相关 40](#_Toc406568929)

[测试 40](#_Toc406568930)

[1. 清除测试数据 40](#_Toc406568931)

[LOAD GEN OPTIONS: 40](#_Toc406568932)

[4 Radosgw-admin命令 41](#_Toc406568933)

[4.1 用户管理 41](#_Toc406568934)

[创建用户 41](#_Toc406568935)

[创建子用户 43](#_Toc406568936)

[获取用户信息 44](#_Toc406568937)

[修改用户信息 44](#_Toc406568938)

[用户使能和挂起 44](#_Toc406568939)

[删除用户或子用户 44](#_Toc406568940)

[创建Key 45](#_Toc406568941)

[添加/删除接入Key和secret 45](#_Toc406568942)

[添加/删除管理权限 46](#_Toc406568943)

[4.2 配额管理 46](#_Toc406568944)

[设置用户配额 47](#_Toc406568945)

[使能/去使能用户配额 47](#_Toc406568946)

[设置Bucket配额 47](#_Toc406568947)

[使能/去使能Bucket配额 48](#_Toc406568948)

[获取配额配置 48](#_Toc406568949)

[更新配额状态 48](#_Toc406568950)

[获取用户使用信息 48](#_Toc406568951)

[读写全局配额 48](#_Toc406568952)

[4.3 使用状态 49](#_Toc406568953)

[显示已用配额 49](#_Toc406568954)

[释放已用配额 49](#_Toc406568955)

[5 rbd 50](#_Toc406568956)

[5.1 RBD命令 50](#_Toc406568957)

[5.2 块的增、删、改、拷贝、改名 51](#_Toc406568958)

[5.3 查看指定rbd块的信息（以plain格式和json格式显示结果） 51](#_Toc406568959)

[5.4 删除foo块 51](#_Toc406568960)

[5.5 创建image格式为2的rbd块 51](#_Toc406568961)

[5.6 块拷贝。将块拷贝到一个新块中，不能拷贝到已经存在的块。 52](#_Toc406568962)

[5.7 修改块名字。例子中，将foo\_c01块改名为foo\_c01\_mv。 52](#_Toc406568963)

[5.8 块的导入、导出 52](#_Toc406568964)

[5.9 快照的创建、删除、查询、回滚 55](#_Toc406568965)

[5.10 块的克隆 56](#_Toc406568966)

[5.11 块映射（map） 57](#_Toc406568967)

[5.12 块锁（加锁、解锁、查询） 58](#_Toc406568968)

# Ceph-deploy命令

ceph-deploy [-h] [-v | -q] [--version] [--username USERNAME]

[--overwrite-conf] [--cluster NAME]

COMMAND ...

## 创建集群监控节点信息并生成集群配置文件(ceph.conf)

ceph-deploy --overwrite-conf new node1 node2 node3 …

## 初始化集群监控节点

ceph-deploy --overwrite-conf mon create-initial

## 从各节点上收集集群相关配置文件

ceph-deploy gatherkeys node1 node2 node3 …

## 准备OSD数据节点

ceph-deploy --overwrite-conf osd prepare node1:/sdb1

## 激活OSD数据节点

ceph-deploy --overwrite-conf osd activate node1:/sdb1

## 分发集群配置文件到各个节点上

ceph-deploy admin node1 node2 node3 …

说明

按照1.4.1~1.4.6的步骤执行相关命令就可以正常创建一个集群了。

## 创建MDS元数据服务器（文件存储专用）

ceph-deploy mds create node1 node2 node3 …

## 列出节点上磁盘信息的命令

ceph-deploy disk list node1 node2 node3 …

## 在节点上安装ceph

ceph-deploy install node1 node2 node3 …

## 从节点上删除ceph

ceph-deploy purge node1 node2 node3 …

## 从节点上删除ceph的配置信息

ceph-deploy purgedata node1 node2 node3 …

## 清除集群中的认证信息

ceph-deploy forgetkeys

# Ceph命令

## Global

### 集群进程操作命令

### 查看ceph进程名称

命令： initctl list | grep ceph

结果：

ceph-osd-all start/running

ceph-osd-all-starter stop/waiting

ceph-all start/running

ceph-mon-all start/running

ceph-mon-all-starter stop/waiting

ceph-mon (ceph/snode01) start/running, process 4064

ceph-create-keys stop/waiting

ceph-osd stop/waiting

### 启动所有ceph守护进程

命令：start ceph-all

要启动特定类型的Ceph的节点上所有的守护进程，执行下列操作之一：

sudo start ceph-osd-all

sudo start ceph-mon-all

sudo start ceph-mds-all

要启动特定的守护进程实例，在Ceph节点上，执行下列操作之一：

sudo start ceph-osd id={id}

sudo start ceph-mon id={hostname}

sudo start ceph-mds id={hostname}

### 停止所有ceph守护进程

命令：stop ceph-all

要停止特定类型的Ceph的节点上所有的守护进程，请执行下列操作之一：

sudo stop ceph-osd-all

sudo stop ceph-mon-all

sudo stop ceph-mds-all

要停止一个特定的守护进程实例Ceph的节点上，执行下列操作之一：

sudo stop ceph-osd id={id}

sudo stop ceph-mon id={hostname}

sudo stop ceph-mds id={hostname}

### 集群状态查询命令

### 集群健康状况查询

root@node1:~# **ceph health**

HEALTH\_OK

### 集群详情

root@node1:~# **ceph -s**

cluster 3aa4e0e0-4275-435e-bd91-868213af787f

health HEALTH\_OK

monmap e1: 1 mons at {node1=172.16.20.100:6789/0},

election epoch 1, quorum 0 node1

osdmap e25: 2 osds: 2 up, 2 in

pgmap v1559: 320 pgs, 4 pools, 380 MB data, 103 objects

802 MB used, 38135 MB / 38937 MB avail

320 active+clean

### 集群ID查询

root@node1:~# **ceph fsid**

3aa4e0e0-4275-435e-bd91-868213af787f

### 集群认证列表查询

root@node1:~# **ceph auth list**

installed auth entries:

osd.0

key: AQBPnEZU4GwXMhAAw1mo3MtOSEqmxCI5VEBF+Q==

caps: [mon] allow profile osd

caps: [osd] allow \*

osd.1

key: AQCgU0hUIHUTGxAA4B/J8IDopw46dFy1kf9iAA==

caps: [mon] allow profile osd

caps: [osd] allow \*

client.admin

key: AQD8m0ZUqH0qFBAA+AQSCMLODT7ssyCuckm+Qw==

caps: [mds] allow

caps: [mon] allow \*

caps: [osd] allow \*

client.bootstrap-mds

key: AQD9m0ZUkGzdFhAAYwPXyc9t/Vl96o5ysGgMBw==

caps: [mon] allow profile bootstrap-mds

client.bootstrap-osd

key: AQD8m0ZU4B4MNhAAT7Djie6FZxVPvzE1PgOtMQ==

caps: [mon] allow profile bootstrap-osd

### 集群磁盘使用状况查询

root@node1:~# **ceph df [detail]**

GLOBAL:

SIZE AVAIL RAW USED %RAW USED

38937M 38135M 802M 2.06

POOLS:

NAME ID USED %USED OBJECTS

data 0 0 0 0

test 1 380M 0.98 103

说明

输入ceph并按回车可以进入交互模式，在交互模式下输入命令可以省掉ceph前缀，执行结果跟非交互模式是一致的。

### 集群内存相关命令

### 集群内存使用状况查询

root@node1:~# **ceph heap stats**

mon.node1tcmalloc heap stats:------------------------------------------------

MALLOC: 4935944 ( 4.7 MiB) Bytes in use by application

MALLOC: + 49152 ( 0.0 MiB) Bytes in page heap freelist

MALLOC: + 615880 ( 0.6 MiB) Bytes in central cache freelist

MALLOC: + 195264 ( 0.2 MiB) Bytes in transfer cache freelist

MALLOC: + 1494640 ( 1.4 MiB) Bytes in thread cache freelists

MALLOC: + 1159320 ( 1.1 MiB) Bytes in malloc metadata

MALLOC: ------------

MALLOC: = 8450200 ( 8.1 MiB) Actual memory used (physical + swap)

MALLOC: + 49152 ( 0.0 MiB) Bytes released to OS (aka unmapped)

MALLOC: ------------

MALLOC: = 8499352 ( 8.1 MiB) Virtual address space used

MALLOC:

MALLOC: 389 Spans in use

MALLOC: 14 Thread heaps in use

MALLOC: 8192 Tcmalloc page size

### 释放集群节点中空闲的内存

root@node1:~# **ceph heap release**

mon.node1 releasing free RAM back to system.

## Auth

## Config-Key

## Mon

### MON 状态查询命令

### ceph mon stat命令查询mon map信息

例子

root@snode01:~# ceph mon stat

e1: 1 mons at {snode01=192.168.7.101:6789/0}, election epoch 2, quorum 0 snode01

## Mds

## Osd

### OSD状态查询命令

### ceph osd stat命令查询osd map信息

例子

root@snode01:~# ceph osd stat

osdmap e62: 3 osds: 3 up, 3 in

### ceph osd dump命令查询osd详细信息

例子

root@snode01:~# ceph osd dump

epoch 62

fsid 7f8f00ac-c059-46a6-b656-4a1ce781607b

created 2014-07-31 22:42:07.125753

modified 2014-08-06 19:42:08.919214

flags

pool 0 'data' replicated size 2 min\_size 1 crush\_ruleset 0 object\_hash rjenkins pg\_num 64 pgp\_num 64 last\_change 56 flags hashpspool crash\_replay\_interval 45 stripe\_width 0

snap 1 'data\_snap\_01' 2014-08-05 10:37:44.093237

pool 1 'metadata' replicated size 2 min\_size 1 crush\_ruleset 0 object\_hash rjenkins pg\_num 64 pgp\_num 64 last\_change 1 flags hashpspool stripe\_width 0

pool 2 'rbd' replicated size 2 min\_size 1 crush\_ruleset 0 object\_hash rjenkins pg\_num 64 pgp\_num 64 last\_change 1 flags hashpspool stripe\_width 0

pool 4 'jfl' replicated size 2 min\_size 1 crush\_ruleset 0 object\_hash rjenkins pg\_num 200 pgp\_num 200 last\_change 60 flags hashpspool stripe\_width 0

max\_osd 4

osd.1 up in weight 1 up\_from 10 up\_thru 60 down\_at 0 last\_clean\_interval [0,0) 192.168.7.103:6800/4494 192.168.7.103:6801/4494 192.168.7.103:6802/4494 192.168.7.103:6803/4494 exists,up ffd9c912-6b77-4be1-83cc-6efcc9b17c74

osd.2 up in weight 1 up\_from 23 up\_thru 61 down\_at 0 last\_clean\_interval [0,0) 192.168.7.104:6800/5021 192.168.7.104:6801/5021 192.168.7.104:6802/5021 192.168.7.104:6803/5021 exists,up 9a532a58-6a88-40ad-aefe-76cf70ba1495

osd.3 up in weight 1 up\_from 50 up\_thru 60 down\_at 0 last\_clean\_interval [0,0) 192.168.7.102:6800/9623 192.168.7.102:6801/9623 192.168.7.102:6802/9623 192.168.7.102:6803/9623 exists,up 48c060ef-894f-4791-b510-992732c6b36d

### ceph osd tree命令查询crush树

例子

root@snode01:~# ceph osd tree

# id weight type name up/down reweight

-1 3 root default

-3 1 host snode03

1 1 osd.1 up 1

-4 1 host snode04

2 1 osd.2 up 1

-5 1 host snode02

3 1 osd.3 up 1

## pool

### Pool

### 创建cache pool

【命令】

**ceph osd pool create** <poolname> <int[0-]> {<int[0-]>} {replicated|erasure} {<erasure\_code\_profile>} {<ruleset>}

【目录】

任意目录

【参数】

<poolname>：

创建的pool名字，格式为String。

<int[0-]>：

pg的个数。

{<int[0-]>}：

pgp的个数。

{replicated|erasure} ：

Replicated说明Cache的类型是副本类型。

Erasure说明Cache的类型是纠删码方式的。

{<erasure\_code\_profile>}：

纠删码的策略。

{<ruleset>}

CRUSH的规则集。

【举例】

# 创建pg为8，pgp为2的副本类型的poo3。

root@compute1:~# ceph osd pool create poo3 8 2 replicated

# 创建pg为8，pgp为8的纠删码类型的poo5。

root@compute1:~# ceph osd pool create poo5 12 12 erasure

# 创建pg为8，pgp为8的纠删码类型的poo6，纠删码

策略是myprofile5。

1、创建配置文件myprofile

root@compute1:~# ceph osd erasure-code-profile set myprofile5 \

> k=3\

> m=2\

> ruleset-failure-domain=host

2、基于myprofile5创建纠删码pool，名为ecpool1

root@compute1:~# ceph osd pool create ecpool1 12 12 erasure myprofile5

3、基于这个pool创建一个对象

root@compute1:/test# vi aaaa

root@compute1:/test# rados --pool ecpool1 put obj\_normal5 /test/aaaa -

4、读取对象

root@compute1:/test# rados --pool ecpool1 get obj\_normal5 –

### 删除cache pool

【命令】

**ceph osd pool delete** <poolname> {<poolname>} {--yes-i-really-really-mean-it}

【目录】

任意目录

【参数】

<poolname>：

删除的pool名字，格式为String。

{<poolname>}：

删除的pool名字，格式为String。

【举例】

# 删除名称为ecpool的cache pool。

root@openstack:~# ceph osd pool delete ecpool ecpool

--yes-i-really-really-mean-it

**pool 'ecpool' removed**

### show cache pool

【命令】

**ceph osd pool get** <poolname> size|

min\_size|

crash\_replay\_interval|

pg\_num|

pgp\_num|

crush\_ruleset|

hit\_set\_type|

hit\_set\_period|

hit\_set\_count|

hit\_set\_fpp|

target\_max\_objects|

target\_max\_bytes|

cache\_target\_dirty\_ratio|

cache\_target\_full\_ratio|

cache\_min\_flush\_age|

cache\_min\_evict\_age|

erasure\_code\_profile|

min\_read\_recency\_for\_promote|

auid

【目录】

任意目录

【参数】

<poolname>：pool的名字。

size：需要查找pool的大小。

min\_size：最小大小。

crash\_replay\_interval：重复crash的时间间隔。

pg\_num：pg的数量。

pgp\_num：pgp的数量。

crush\_ruleset：crush的规则集。

hit\_set\_type：hitset类型。

hit\_set\_period：hitset周期。

hit\_set\_count：hitset数量。

hit\_set\_fpp：hitset fpp。

target\_max\_objects：最大对象数量。

target\_max\_bytes：最大字节数量。

cache\_target\_dirty\_ratio：cache中dirty数据的比例。

cache\_target\_full\_ratio：cache中full数据的比例。

cache\_min\_flush\_age：cache中的最小flush时间。

cache\_min\_evict\_age：cache中的最小evict时间。

erasure\_code\_profile：纠删码的策略。

auid：获取pool对应的授权用户ID，0表示成功，负数表示失败，正数表示对应的pool id。

【举例】

# 获取名为.users的pool的大小。

root@openstack:~# ceph osd pool get .users size

size: 3

# 获取名为.users的pool的最小大小。

root@openstack:~# ceph osd pool get .users min\_size

min\_size: 1

# 获取名为.users的pool的重复crash的时间间隔。

root@openstack:~# ceph osd pool get .users crash\_replay\_interval

crash\_replay\_interval: 0

# 获取名为.users的pool的pg的数量。

root@openstack:~# ceph osd pool get .users pg\_num

# 获取名为.users的pool的pgp的数量。

root@openstack:~# ceph osd pool get .users pgp\_num

# 获取名为.users的pool的crush的规则集。

root@openstack:~# ceph osd pool get .users crush\_ruleset

# 获取名为.users的pool的hitset类型。

root@openstack:~# ceph osd pool get .users hit\_set\_type

# 获取名为.users的pool的hitset周期。

root@openstack:~# ceph osd pool get .users hit\_set\_period

# 获取名为.users的pool的hitset数量。

root@openstack:~# ceph osd pool get .users hit\_set\_count

# 获取名为.users的pool的hitset fpp。

root@openstack:~# ceph osd pool get .users hit\_set\_fpp

# 获取名为.users的pool的最大对象数量。

root@openstack:~# ceph osd pool get .users target\_max\_objects

# 获取名为.users的pool的最大字节数量。

root@openstack:~# ceph osd pool get .users target\_max\_bytes

# 获取名为.users的pool的cache中dirty数据的比例。

root@openstack:~# ceph osd pool get .users cache\_target\_dirty\_ratio

# 获取名为.users的pool的cache中full数据的比例。

root@openstack:~# ceph osd pool get .users cache\_target\_full\_ratio

# 获取名为.users的pool的cache中的最小flush时间。

root@openstack:~# ceph osd pool get .users cache\_min\_flush\_age

# 获取名为.users的pool的cache中的最小evict时间。

root@openstack:~# ceph osd pool get .users cache\_min\_evict\_age

# 获取名为.users的pool的纠删码的策略。

root@openstack:~# ceph osd pool get .users erasure\_code\_profile

# 获取名为.users的pool的对应的授权用户ID。

root@openstack:~# ceph osd pool get .users auid

### 设置cache pool的参数和值

【命令】

**ceph osd pool set** <poolname> size|

min\_size|

crash\_replay\_interval|

pg\_num|

pgp\_num|

hashpspool|

crush\_ruleset|

hit\_set\_type|

hit\_set\_period|

hit\_set\_count|

hit\_set\_ fpp|

debug\_fake\_ec\_pool|

target\_max\_ bytes|

target\_max\_objects|

cache\_target\_dirty\_ratio|

cache\_target\_full\_ratio|

cache\_min\_flush\_age|

cache\_min\_evict\_age|

auid|

min\_read\_recency\_for\_promote

<val>

【目录】

任意目录

【参数】

<poolname>：pool的名字。

size： pool的大小。

min\_size：最小大小。

crash\_replay\_interval：重复crash的时间间隔，单位是秒。

pg\_num：pg的数量。

pgp\_num：pgp的数量，<= pg\_num。

hashpspool：在已有的pool中设置HASHPSPOOL标志。

类型：整形

范围：1 表示 sets flag

0表示 unsets flag

crush\_ruleset：设置crush的规则集，规则集的目的是在集群中映射对象存储的位置。

hit\_set\_type：hit set类型。

类型：String

包括：bloom： 表示bloom filter类型；（默认类型）

explicit\_hash：表示explicit hash类型；（测试类型）

explicit\_object：表示explicit object类型。（测试类型）

hit\_set\_period：hit set的持续时间周期。这个值越高，内存占用率高的可能性越大。

hit\_set\_count：hit set数量。这个值越高，内存占用率高的可能性越大。

hit\_set\_fpp：hit set fpp。Bloom filter的hit set的假阳性值。

有效值：0.0～1.0

默认值：0.05

target\_max\_objects：最大对象数量。当达到这个阈值时，Ceph将开始flush或扔掉对象。

target\_max\_bytes：最大字节数量。当达到这个阈值时，Ceph将开始flush或仍掉对象。

类型：整形

cache\_target\_dirty\_ratio：cache中dirty数据的比例。

类型：double

默认值：.4

cache\_target\_full\_ratio：cache中full数据的比例。

类型：double

默认值：.8

cache\_min\_flush\_age：cache中的最小flush时间。当达到这个阈值时，Ceph将从cache pool flush一个对象到存储池。

单位：秒

cache\_min\_evict\_age：cache中的最小evict时间。当达到这个阈值时，Ceph将从cache pool 扔掉一个对象。

单位：秒

erasure\_code\_profile：纠删码的策略。

auid：获取pool对应的授权用户ID，0表示成功，负数表示失败，正数表示对应的pool id。

min\_read\_recency\_for\_promote：延迟时间。

单位：秒

【举例】

# 设置rbd pool的大小为9。

root@openstack:~# ceph osd pool set rbd size 9

set pool 0 size to 9

# 设置rbd pool的最小大小为8。

root@openstack:~# ceph osd pool set rbd min\_size 8

set pool 0 min\_size to 8

# 设置rbd pool的重复时间间隔为8。

root@openstack:~# ceph osd pool set rbd crash\_replay\_interval 8

set pool 0 crash\_replay\_interval to 8

# 设置rbd pool的pg的数量为9。

root@openstack:~# ceph osd pool set rbd pg\_num 9

set pool 0 pg\_num to 9

#设置rbd pool的pgp的数量为9。

root@openstack:~# ceph osd pool set rbd pgp\_num 9

set pool 0 pgp\_num to 9

# 设置rbd pool的crush规则集为编号1的规则集。

root@openstack:~# ceph osd pool set rbd crush\_ruleset 1

set pool 0 crush\_ruleset to 1

# 设置名为.users 的 tier的hit\_set\_type为bloom filter。

root@openstack:~# ceph osd pool set .users hit\_set\_type bloom

set pool 9 hit\_set\_type to bloom

# 设置名为.users 的 tier的hit\_set\_type为explicit\_hash。

root@openstack:~# ceph osd pool set .users hit\_set\_type explicit\_hash

set pool 9 hit\_set\_type to explicit\_hash

# 设置名为.users 的 tier的hit\_set\_type为explicit\_object。

root@openstack:~# ceph osd pool set .users hit\_set\_type explicit\_object

set pool 9 hit\_set\_type to explicit\_object

# 设置名为.users 的 tier的hit\_set\_period为8。

root@openstack:~# ceph osd pool set .users hit\_set\_period 8

set pool 9 hit\_set\_period to 8

# 设置名为.users 的 tier的hit\_set\_count为8。

root@openstack:~# ceph osd pool set .users hit\_set\_count 8

set pool 9 hit\_set\_count to 8

# 设置名为.users 的 tier的hit\_set\_fpp为8。

root@openstack:~# ceph osd pool set .users hit\_set\_fpp .8

set pool 9 hit\_set\_fpp to .8

# 设置名为.users 的 tier的最大用户数量为8。

root@openstack:~# ceph osd pool set .users target\_max\_objects 8

set pool 9 target\_max\_objects to 8

#设置名为.users 的 tier的最大字节数为20。

root@openstack:~# ceph osd pool set .users target\_max\_bytes 20

set pool 9 target\_max\_bytes to 20

# 设置名为.users 的 tier的cache\_target\_dirty\_ratio 为0.4。

root@openstack:~# ceph osd pool set .users cache\_target\_dirty\_ratio 0.4

set pool 9 cache\_target\_dirty\_ratio to 0.4

# 设置名为.users 的 tier的cache\_target\_full\_ratio 为0.4。

root@openstack:~# ceph osd pool set .users cache\_target\_full\_ratio 0.4

set pool 9 cache\_target\_full\_ratio to 0.4

#设置名为.users 的 tier的cache\_min\_flush\_age 为3600。

root@openstack:~# ceph osd pool set .users cache\_min\_flush\_age 3600

set pool 9 cache\_min\_flush\_age to 3600

#设置名为.users 的 tier的cache\_min\_evict\_age 为3600。

root@openstack:~# ceph osd pool set .users cache\_min\_evict\_age 3600

set pool 9 cache\_min\_evict\_age to 3600

#设置名为.users 的 tier的min\_read\_recency\_for\_promote 为3600。

root@openstack:~# ceph osd pool set .users min\_read\_recency\_for\_promote 3600

set pool 9 min\_read\_recency\_for\_promote to 3600

### 设置cache pool中的对象或字节数的配额

【命令】

**ceph osd pool set-quota** <poolname> max\_objects|max\_bytes <val>

【目录】

任意目录

【参数】

<poolname> pool名称

max\_objects 说明是配置object的最大值

max\_bytes 说明是配置byte的最大值

<val> 具体配额的值

【举例】

# 设置.users的object的最大配额为10000。

root@compute1:~# ceph osd pool set-quota .users max\_objects 10000

set-quota max\_objects = 10000 for pool .users

### 给cache pool制作快照

【命令】

**ceph osd pool mksnap** <poolname> <snap>

【目录】

任意目录

【参数】

<poolname> pool的名称

<snap> 快照名称

【举例】

# 为名为.users的pool制作快照.u。

root@compute1:~# ceph osd pool mksnap .users .u

created pool .users snap .u

### 给cache pool重命名

【命令】

**ceph osd pool rename** <poolname> <poolname>

【目录】

任意目录

【参数】

无。

【举例】

# 将名为.rgw的pool重命名为.rgw\_bak。

root@compute1:~# ceph osd pool rename .rgw .rgw\_bak

pool '.rgw' renamed to '.rgw\_bak'

### 从cache pool中删除快照

【命令】

**ceph osd pool rmsnap**<poolname> <snap>

【目录】

任意目录

【参数】

<poolname> pool的名称

<snap> 快照名称

【举例】

# 删除.users pool中的快照.u。

root@compute1:~# ceph osd pool rmsnap .users .u

removed pool .users snap .u

### 获取cache pool的状态

【命令】

**ceph osd pool stats** {<name>}

【目录】

任意目录

【参数】

{<name>} pool的名称

【举例】

# 获取.users pool的状态信息。

root@compute1:~# ceph osd pool stats .users

pool .users id 18

nothing is going on

### 获取cache pool的信息列表

【命令】

**ceph osd** **lspools**

【目录】

任意目录

【参数】

无

【举例】

# 显示整个系统的pool信息。

root@openstack:~# ceph osd lspools

### 在与指定的object对应的pool中查找pg

【命令】

**Ceph osd map** <poolname> <objectname>

【目录】

任意目录

【参数】

<poolname> Cache pool的名字；

<objectname> object的名称。

【举例】

# 通过object对应的pool，查找pg。

1. 获取pool的信息

root@openstack:~# ceph osd lspools

0 rbd,1 cephfs\_data,2 cephfs\_metadata,3 .rgw.root,4 .rgw.control,5 .rgw,6 .rgw.gc,7 .users.uid,8 .users.email,9 .users,10 .users.swift,11 ecpool,

1. 显示ecpool中的object情况

root@openstack:~# rados -p ecpool ls

3、通过object对应的pool，查找pg。

root@openstack:~# ceph osd map ecpool ecobj3

osdmap e103 pool 'ecpool' (11) object 'ecobj3' -> pg 11.ca1875b2 (11.2) -> up ([2,0,2147483647], p2) acting ([2,0,2147483647], p2)

### Cache tierring

### 添加second tier pool到base cache pool，建立一个非空的tierpool

【命令】

**ceph osd tier add** <poolname><poolname> {--force-nonempty}

【目录】

任意目录

【参数】

<poolname> second tier pool。

<poolname> base cache pool。

{--force-nonempty} 强制非空参数。

【举例】

# 将.users作为.users.uid上的一个分层。

root@openstack:~# ceph osd tier add .users.uid .users --force-nonempty

pool '.users' is now (or already was) a tier of '.users.uid'

### 添加second tier pool到base cache pool，指定tierpool的大小

【命令】

**ceph osd tier add-cache** <poolname><poolname> <int[0-]>

【目录】

任意目录

【参数】

无

【举例】

# # 将.users作为.users.uid上的一个分层，并且分层大小为12。

root@openstack:~# ceph osd tier add-cache .users.uid .users.email 14

pool '.users.email' is now (or already was) a tier of '.users.uid'

### 为tire pool指定cache模式

【命令】

**ceph osd tier cache-mode** <poolname> none

|writeback

|forward

|readonly

【目录】

任意目录

【参数】

<poolname> tier的名字。

none 设置tier的cache-mode为none。

writeback 设置tier的cache-mode为writeback。

forward 设置tier的cache-mode为forward。

readonly 设置tier的cache-mode为readonly。

【举例】

#指定tier .users 的cache-mode为none。

root@openstack:~# ceph osd tier cache-mode .users none

set cache-mode for pool '.users' to none

#指定tier .users 的cache-mode为writeback。

root@openstack:~# ceph osd tier cache-mode .users writeback

set cache-mode for pool '.users' to writeback

#指定tier .users 的cache-mode为forward。

root@openstack:~# ceph osd tier cache-mode .users forward

set cache-mode for pool '.users' to forward

#指定.users.email 的cache-mode为readonly。

root@openstack:~# ceph osd tier cache-mode .users.email readonly

set cache-mode for pool '.users.email' to readonly

### 从base tier pool移除second tier pool

【命令】

**ceph osd tier remove** <poolname> <poolname>

【目录】

任意目录

【参数】

<poolname> second tier pool。

<poolname> base cache pool。

【举例】

#移除.users.email tier。

root@openstack:~# ceph osd tier remove .users.uid .users.email

pool '.users.email' is now (or already was) not a tier of '.users.uid'

### 从base tier pool移除overlay pool

【命令】

**ceph osd tier remove-overlay** <poolname>

【目录】

任意目录

【参数】

无

【举例】

# 删除.users.uid pool上的overlay。

root@openstack:~# ceph osd tier remove-overlay .users.uid

there is now (or already was) no overlay for '.users.uid'

### 给base pool设置overlay pool

【命令】

**osd tier set-overlay** <poolname><poolname>

【目录】

任意目录

【参数】

无

【举例】

# 设置.users.uid的overlay pool为.users.email。

1、设置.users.email为.users.uid的tier。

root@openstack:~# ceph osd tier add .users.uid .users.email --force-nonempty

pool '.users.email' is now (or already was) a tier of '.users.uid'

2、设置.users.email为.users.uid的overlay pool。

root@openstack:~# ceph osd tier set-overlay .users.uid .users.email

overlay for '.users.uid' is now (or already was) '.users.email'

### Erasure

### 创建erasure profile，plugin=jerasure

【命令】

**ceph osd erasure-code-profile set {name} \**

**plugin=jerasure \**

**k={data-chunks} \**

**m={coding-chunks} \ technique={reed\_sol\_van|reed\_sol\_r6\_op|cauchy\_orig|cauchy\_good|liberation|blaum\_roth|liber8tion} \**

**[ruleset-root={root}] \**

**[ruleset-failure-domain={bucket-type}] \**

**[directory={directory}] \**

**[--force]**

【参数】

**name：**

profile 名字，创建erasure pool时使用

**plugin**：

使用哪个算法库，目前支持jerasure/lrc/isa

默认使用jerasure

**k={data-chunks}**：

数据块个数

**m={coding-chunks}**：

校验块个数

**technique:**

**plugin=jerasure {reed\_sol\_van|reed\_sol\_r6\_op|cauchy\_orig|cauchy\_good|liberation|blaum\_roth|liber8tion}**

**[ruleset-root={root}]**：

为first step of the ruleset指定crush bucket type

Default: default

**[ruleset-failure-domain={bucket-type}]：**

如果被设为host，则保证两个数据块不在同一host，用于ruleset，如step chooseleaf host

**[directory={directory}]：**

纠删码库的存放路径，default：/usr/lib/ceph/erasure-code

【举例】

### 创建erasure profile，plugin=isa

【命令】

**ceph osd erasure-code-profile set {name} \**

**plugin=isa \**

**technique={reed\_sol\_van|cauchy} \**

**[k={data-chunks}] \**

**[m={coding-chunks}] \**

**[ruleset-root={root}] \**

**[ruleset-failure-domain={bucket-type}] \**

**[directory={directory}] \**

**[--force]**

【参数】

**name：**

profile 名字，创建erasure pool时使用

**plugin**：

使用哪个算法库，目前支持jerasure/lrc/isa

默认使用jerasure

**k={data-chunks}**：

数据块个数

**m={coding-chunks}**：

校验块个数

**technique:**

plugin=jerasure {reed\_sol\_van|cauchy}

**l={locality}**  ：

k=4,m=2,l=3

则数据块和校验块被分成2组，每个组都可以独立恢复数据

**[ruleset-root={root}]**：

为first step of the ruleset指定crush bucket type

Default: default

**[ruleset-locality={bucket-type}]：**

**[ruleset-failure-domain={bucket-type}]：**

如果被设为host，则保证两个数据块不在同一host，用于ruleset，如step chooseleaf host

**[directory={directory}]：**

纠删码库的存放路径，default：/usr/lib/ceph/erasure-code

【举例】

### 创建erasure profile

【命令】

**ceph osd erasure-code-profile set {name} \**

**plugin=lrc \**

**k={data-chunks} \**

**m={coding-chunks} \**

**l={locality} \**

**[ruleset-root={root}] \**

**[ruleset-locality={bucket-type}] \**

**[ruleset-failure-domain={bucket-type}] \**

**[directory={directory}] \**

**[--force]**

【参数】

**name：**

profile 名字，创建erasure pool时使用

**plugin**：

使用哪个算法库，目前支持jerasure/lrc/isa

默认使用jerasure

**k={data-chunks}**：

数据块个数

**m={coding-chunks}**：

校验块个数

**l={locality}**  ：

k=4,m=2,l=3

则数据块和校验块被分成2组，每个组都可以独立恢复数据

**[ruleset-root={root}]**：

为first step of the ruleset指定crush bucket type

Default: default

**[ruleset-locality={bucket-type}]：**

**[ruleset-failure-domain={bucket-type}]：**

如果被设为host，则保证两个数据块不在同一host，用于ruleset，如step chooseleaf host

**[directory={directory}]：**

纠删码库的存放路径，default：/usr/lib/ceph/erasure-code

【举例】

### 查看erasure profile

【命令】

**ceph osd erasure-code-profile get {name}**

【参数】

**name：**

profile 名字

【举例】

## Pg

# Rados命令

rados命令用主要用来管理和测试ceph集群。

命令格式：rados [commands] [options]

举例：rados ls -p data

其中ls属于cmd，表示获取显示pool中对象信息，-p属于options，指明获取data pool下的的对象信息

详细信息运行 rados --help 获取

## pool

### 显示当前所有pool

rados lspools

### 创建pool，可以指定owner\_auid和crush\_rule

rados mkpool <pool-name> [owner\_auid[crush\_rule]]

rados mkpool test 123 4

### 复制pool

rados cppool <pool-name> <dest-pool>

rados cpool test1 test2

### 删除pool

rados rmpool <pool-name> [<pool-name> --yes-i-really-really-mean-it]

rados rmpool test test --yes-i-really-really-mean-it

### 显示pool使用情况

rados df

### 显示pool中的object

rados ls –p test

### 修改pool的owner auid

rados chown [owner\_auid]

## object

### 获取obj从指定pool

rados get object-name object-name.outfile –p test

### 放入obj到指定pool

rados put object-name object-name.infile –p test

### 截断obj

rados truncate object-name length –p test

### 创建obj，可以指定类型（NONE，MAIN，SHADOW，MULTIMETA）

rados create <obj-name> [category]

rados create object-name –p test

### 删除obj

rados rm object-name –p test

### 复制obj

rados cp object-name1 object-name2 –p test

### 克隆obj的数据

rados clonedata object-src object-dst –p test

### 显示obj的附加属性

rados listxattr object-name –p test

### 获取obj的附加属性，需要指定attr，可由listxattr获取

rados getxattr object-name attr –p test

### 设置obj的附加属性

rados setxattr object-name attr val –p test

### 删除obj附加属性

rados rmxattr object-name attr –p test

### 显示obj状态

rados stat objname –p test

### 额外map信息

rados mapext <obj-name> –p test

### 显示快照信息

rados lssnap –p test

### 创建快照

rados mksnap <snap-name> –p test

### 删除快照

rados rmsnap <snap-name> –p test

### 回滚obj到指定快照

rados rollback <obj-name> <snap-name> –p test

### 显示对象的快照

rados listsnaps <obj-name> –p test

### 列出omapkeys

rados listomapkeys <obj-name> –p test

### 列出omap keys 和 vals

rados listomapvals <obj-name> –p test

### 获取omap 具体key的值

rados getomapval <obj-name> <key> [file] –p test

### 设置omap具体key的值

rados setomapval <obj-name> <key> <val> -p test

### 删除omap

rados rmomapkey <obj-name> <key> -p test

### 获取omap头

rados getomapheader <obj-name> [file] –p test

### 设置omap头

rados setomapheader <obj-name> <val> -p test

### 转换tmap到omap

rados tmap-to-omap <obj-name> -p test

### 显示obj的观察者

rados listwatchers <obj-name> -p test

### 设置set-alloc-hint

rados set-alloc-hint <obj-name> <expected-object-size> <expected-write-size>

## 导入和导出

import [options] <local-directory> <rados-pool>

Upload <local-directory> to <rados-pool>

export [options] rados-pool> <local-directory>

Download <rados-pool> to <local-directory>

options:

-f / --force Copy everything, even if it hasn't changed.

-d / --delete-after After synchronizing, delete unreferenced

files or objects from the target bucket

or directory.

--workers Number of worker threads to spawn

(default 5)

## 锁相关

lock list <obj-name>

List all advisory locks on an object

lock get <obj-name> <lock-name>

Try to acquire a lock

lock break <obj-name> <lock-name> <locker-name>

Try to break a lock acquired by another client

lock info <obj-name> <lock-name>

Show lock information

options:

--lock-tag Lock tag, all locks operation should use

the same tag

--lock-cookie Locker cookie

--lock-description Description of lock

--lock-duration Lock duration (in seconds)

--lock-type Lock type (shared, exclusive)

## cache pools

CACHE POOLS: (for testing/development only)

cache-flush <obj-name> flush cache pool object (blocking)

cache-try-flush <obj-name> flush cache pool object (non-blocking)

cache-evict <obj-name> evict cache pool object

cache-flush-evict-all flush+evict all objects

cache-try-flush-evict-all try-flush+evict all objects

## bench，测试相关

### 测试

rados bench <seconds> write|seq|rand [-t concurrent\_operations] [--no-cleanup] [--run-name run\_name] [-b op\_size]

default is to clean up after write benchmark

default run-name is 'benchmark\_last\_metadata'

-t N 线程数，default is 16 concurrent IOs and 4 MB ops

-b op\_size 单个对象大小，default is 16 concurrent IOs and 4 MB ops

--concurrent-ios=N

Set number of concurrent I/O operations

--show-time

prefix output with date/time

举例：测试写操作

rados bench 30 write

### 清除测试数据

rados cleanup [--run-name run\_name] [--prefix prefix]

### LOAD GEN OPTIONS:

--num-objects total number of objects

--min-object-size min object size

--max-object-size max object size

--min-ops min number of operations

--max-ops max number of operations

--max-backlog max backlog (in MB)

--percent percent of operations that are read

--target-throughput target throughput (in MB)

--run-length total time (in seconds)

# Radosgw-admin命令

radosgw-admin命令用来管理radosgw。

命令格式：radosgw-admin <cmd> [options...]

举例：rados-admin user info –uid=test

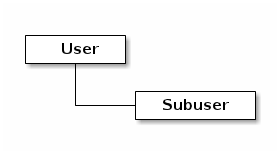
其中user info属于cmd，表示获取用户信息，--uid=test属于options，指明获取test的用户信息

详细信息运行 radosgw-admin --help 获取

## 用户管理

用户管理中的用户指的是使用radosgw的终端用户。

有两种用户类型:

* **User:**根用户，对于S3接口，也是管理员用户，Swift接口不能直接使用根用户。
* **Subuser:** 管理员用户，主要由Swift接口使用。一个子用户与一个用户相关联。

您可以创建、修改、查看、暂停、恢复和删除用户和子用户。

除了User ID和Subuser ID，您可以额外添加显示名称和电子邮件地址。

您可以指定Key和secret，或自动生成Key和secret。

生成或指定key时，注意user对应S3密钥类型，subuser应该对应Swift类型。S3类型为默认值，Swift类型需要特别指定。

### 创建用户

* 创建用户 (S3 interface)， 执行以下命令：

**radosgw-admin user create** **--uid=***{username}* **--display-name="***{display-name}***"** [**--email=***{email}*]

**For example:**

r**adosgw-admin user create --uid=**johndoe **--display-name="**John Doe**" --email=**john@example.com

{ "user\_id": "johndoe",

"display\_name": "John Doe",

"email": "john@example.com",

"suspended": 0,

"max\_buckets": 1000,

"auid": 0,

"subusers": [],

"keys": [

{ "user": "johndoe",

"access\_key": "11BS02LGFB6AL6H1ADMW",

"secret\_key": "vzCEkuryfn060dfee4fgQPqFrncKEIkh3ZcdOANY"}],

"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": []}

创建用户的同时会自动创建S3兼容的客户端接入key和secret

* 同时创建用户和子用户 (Swift interface)， 并且指定secret执行以下命令

**radosgw-admin user create** **--subuser=**tester:testing **--display-name=**Tester-Subuser **--key-type=**swift **--secret=**123456

注：

请检查输出的key。有时radosgw-admin生成的key中包含JSON中的转义字符（\），但是一些Clients无法处理（\），挽救措施是重新生成key，或者手工指定key，确保key中不包含（\）。

### 创建子用户

创建子用户（Swift interface）时，必须指定用户ID、子用户ID和访问等级，必须指定--key-type=swift，否则swift鉴权会失败：

**radosgw-admin subuser create --uid=***{uid}* **--subuser=***{uid}* **--access=**[ **read** | **write** | **readwrite** | **full** ] **–key-type=swift --gen-secret**

For example:

**radosgw-admin subuser create --uid=**johndoe **--subuser=**johndoe:swift **--access=full --key-type=swift --gen-secret**

注：

full is not readwrite, as it also includes the access control policy.

{ "user\_id": "johndoe",

"display\_name": "John Doe",

"email": "john@example.com",

"suspended": 0,

"max\_buckets": 1000,

"auid": 0,

"subusers": [

{ "id": "johndoe:swift",

"permissions": "full-control"}],

"keys": [

{ "user": "johndoe",

"access\_key": "B1I5FSZKYDTU2S5YD3BH",

"secret\_key": "yaOY2NKsOQWkRJuqhieAiNkbYshmOsnPFU\/HLZra"}],

"swift\_keys": [

{ "user": "johndoe:swift",

"secret\_key": " nEYkTukJqfDhDtzhND1BkrZZjce+ff2s4qMcVanG "}],

"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": []}

### 获取用户信息

获取用户信息时，必须指定用户ID (--uid={username}) 或者子用户ID（--subuser={username}）.

**radosgw-admin user info --uid=***johndoe*

### 修改用户信息

修改用户信息时, 必须指定用户ID (--uid={username}) 和你想修改的属性. 比如keys and secrets、email addresses,、display names 和 access levels.

For example:

**radosgw-admin user modify --uid=**johndoe **--display-name=**"John E. Doe"

修改子用户时，需要指明修改子用户，并且指定子用户ID

For example:

**radosgw-admin subuser modify --subuser=j**ohndoe:swift **--access=full**

### 用户使能和挂起

新建用户默认是使能的，可以挂起用户并且之后再次使能。挂起用户时，需要指明挂起动作并且指定用户ID

**radosgw-admin user suspend --uid=**johndoe

再次使能用户，指明使能动作和用户ID.

**radosgw-admin user enable --uid=**johndoe

注：

禁用用户的时候也会禁用属于此用户的子用户

### 删除用户或子用户

删除用户时，指明删除操作并且指定用户ID.

**radosgw-admin user rm --uid=**johndoe

只删除子用户，指明子用户删除操作并且指定子用户ID.

**radosgw-admin subuser rm --subuser=**johndoe:swift

可选项：

* **Purge Data:** The --purge-data option purges all data associated to the UID.
* **Purge Keys:** The --purge-keys option purges all keys associated to the UID.

### 创建Key

为用户创建Key，需要指明生成Key操作，指定用户ID和s3 key类型，为子用户创建Key，指定子用户ID和swift key类型。

For example:

**radosgw-admin key create --subuser=**johndoe:swift **--key-type=**swift **--gen-secret**

{ "user\_id": "johndoe",

"rados\_uid": 0,

"display\_name": "John Doe",

"email": "john@example.com",

"suspended": 0,

"subusers": [

{ "id": "johndoe:swift",

"permissions": "full-control"}],

"keys": [

{ "user": "johndoe",

"access\_key": "QFAMEDSJP5DEKJO0DDXY",

"secret\_key": "iaSFLDVvDdQt6lkNzHyW4fPLZugBAI1g17LO0+87"}],

"swift\_keys": [

{ "user": "johndoe:swift",

"secret\_key": "E9T2rUZNu2gxUjcwUBO8n\/Ev4KX6\/GprEuH4qhu1"}]}

### 添加/删除接入Key和secret

用户和子用户必须拥有接入Key来访问S3接口，Swift接口不需要接入Key。 当你创建用户和子用户时没有指定接入Key和接入secret，那么会自动创建接入Key和接入secret，你可以创建接入Key或者自动生成。你也可以删除接入Key并且包括以下可选项：

* --secret=<key> specifies a secret key (e.g,. manually generated).
* --gen-access-key generates random access key (for S3 user by default).
* --gen-secret generates a random secret key.
* --key-type=<type> specifies a key type. The options are: swift, s3

为指定用户添加接入Key.

**radosgw-admin key create --uid=**johndoe **--key-type=s3** **--gen-access-key --gen-secret**

You may also specify a key and a secret.

为指定用户删除接入Ke

**radosgw-admin key rm --uid=**johndoe

### 添加/删除管理权限

Ceph集群提供REST API管理API。默认用户没有管理权限。使用户拥有管理权限必须指定。

添加用户管理权限:

**radosgw-admin caps add --uid=***{uid}* **--caps=***{caps}*

可以为users, buckets, metadata and usage配置read, write or all capabilities。

For example:

**--caps=**"[**users|buckets|metadata|usage|zone**]=[**\*|read|write|read, write**]"

For example:

**radosgw-admin caps add --uid=**johndoe **--caps=**"users=\*"

删除用户管理权限：:

**radosgw-admin caps remove --uid=**johndoe **--caps=**{*caps*}

## 配额管理

可以为用户和Bucket分配限额。配额包括一个Bucket的最大对象数和最大存储空间。.

* **Bucket:** The --bucket option allows you to specify a quota for buckets the user owns.
* **Maximum Objects:** The --max-objects setting allows you to specify the maximum number of objects. A negative value disables this setting.
* **Maximum Size:** The --max-size option allows you to specify a quota for the maximum number of bytes. A negative value disables this setting.
* **Quota Scope:** The --quota-scope option sets the scope for the quota. The options are bucket and user. Bucket quotas apply to buckets a user owns. User quotas apply to a user.

### 设置用户配额

配置时需要指明配额操作，

For example:

**radosgw-admin quota set --quota-scope=user --uid=**<*uid*> [**--max-objects=**<*num objects*>] [**--max-size=**<*max size*>]

For example:

**radosgw-admin quota set --quota-scope=user --uid=**johndoe **--max-objects=**1024 **--max-size=**1024

负数和/ 或者最大值表示不需要配额检查。

### 使能/去使能用户配额

为用户配置配额后，你可以使能它，For example:

**radosgw-admin quota enable --quota-scope=user --uid=**<*uid*>

你可以可以去使能，For example:

**radosgw-admin quota-disable --quota-scope=user --uid=**<*uid*>

### 设置Bucket配额

Bucket quotas 适用于指定用户的bucket.。他们独立于用户的。

**radosgw-admin quota set --uid=**<*uid*> **--quota-scope=bucket** [**--max-objects=**<*num objects*>] [**--max-size=**<*max size*]

负数和/ 或者最大值表示不检查配额

### 使能/去使能Bucket配额

当你设置配额之后，你可以使能它。For example:

**radosgw-admin quota enable --quota-scope=bucket --uid=**<*uid*>

你可以去使能它，For example:

**radosgw-admin quota-disable --quota-scope=bucket --uid=**<uid>

### 获取配额配置

通过用户接口可以获取配额配置信息。如果使用命令行，执行如下命令：

**radosgw-admin user info --uid=**<*uid*>

### 更新配额状态

配置状态是异步更新的。你可以手动更新所有用户和Bucket，来获取最新配置信息：

**radosgw-admin user stats --uid=**<uid> **--sync-stats**

### 获取用户使用信息

查看用户消耗了多少配额，执行以下命令：

**radosgw-admin user stats --uid=**<*uid*>

注：

建议使用**--sync-stats**参数获取最新信息

### 读写全局配额

获取region map：

**radosgw-admin regionmap get >** *regionmap.json*

简单修改上面导出文件之后，再设置region map.

**radosgw-admin region set <** *regionmap.json*

Note

设置之后需要重启radosgw

## 使用记录

Ceph集群记录所有用户使用信息，你可以跟踪特定期间的信息：.

Options include:

* **Start Date:** The --start-date option allows you to filter usage stats from a particular start date (**format:** yyyy-mm-dd[HH:MM:SS]).
* **End Date:** The --end-date option allows you to filter usage up to a particular date (**format:** yyyy-mm-dd[HH:MM:SS]).
* **Log Entries:** The --show-log-entries option allows you to specify whether or not to include log entries with the usage stats (options: true | false).

Note

你可以指定分钟和秒，但是分辨率只达到了小时级别.

### 显示使用记录

显示指定用户使用记录，你可以指定开始结束日期，并且可以指定是否显示日志占用空间：

**radosgw-admin usage show --uid=johndoe --start-date=**2012-03-01 **--end-date=**2012-04-01

不指定用户则显示所有用户总额

**radosgw-admin usage show --show-log-entries=**false

### 释放使用记录占用的空间

日志占用大量空间，你可以释放他们所占空间

可以指定日期

**radosgw-admin usage trim --start-date=**2010-01-01 **--end-date=**2010-12-31

**radosgw-admin usage trim --uid=**johndoe

**radosgw-admin usage trim --uid=**johndoe **--end-date=**2013-12-31

# rbd

## RBD命令

我们可以对RBD块执行创建、查询、删除、快照、克隆、回滚等操作。

RBD命令格式

rbd [ -c ceph.conf ] [ -m monaddr ] [ -p | –pool pool ] [ –size size ] [ –order bits ] [ command ... ]

命令选项

-c ceph.conf --- 显示指定配置文件，默认为“/etc/ceph/ceph.conf”；

-m monaddr --- 指定要monitor地址。如果不提供的话，直接从ceph.conf文件中查找；

-p pool --- 给定RBD块所在的pool，默认为rbd pool；

--no-progress --- 不输出进度(progress)信息

命令参数

--image-format format --- 块格式，默认格式为1。格式1，可以被所有版本的librbd和内核rbd模块识别，但是不支持新特性，例如克隆；格式2，可以更好地支持新特性（比如，克隆以及将来的一些特性）

--size size-in-mb --- 指定新RBD块的大小，单位是字节；

--order bits --- 默认值为22（4M），指定RBD块条带化的对象的大小。

--strip-unit size-in-bytes --- 指定条带单元（strip unit）的大小。单位为字节。

--strip-count num --- 指定条带的横跨的对象数目

--snap snap --- （为特定的操作）指定快照名字；

--id username --- 指定用户名（不带client前缀）

--keyfile filename --- 指定用户的keyring文件目录，默认使用client.admin用户的keyring文件；

--keyring filename --- 指定用户的keyring文件，如果没有指定，将在默认的Kerying位置搜索文件；

--shared tag --- lock add操作的一个选项。如果多个用户使用相同的tag，那么允许它们同时lock住同个rbd块。Tag是个随机的字符串。在块需要被多个客户同时打开的场景中，这个功能非常有用，比如虚拟机动态迁移，或者作为集群文件系统的块。(不是很明白？？？)

--format format --- 指定输出格式（默认为plain，可以为json,xml）

--o map-options --- 映射（map）选项。

--read-only --- 将image以只读的方式映射，同-o ro.

## 块的增、删、改、拷贝、改名

创建foo块（大小为1024MB，在rbd目录下）以及查看给定pool下的所有块。

root@snode01:~# rbd create foo --size 1024 --pool rbd

root@snode01:~# rbd ls rbd

foo

## 查看指定rbd块的信息（以plain格式和json格式显示结果）

root@snode01:~# rbd --image foo -p rbd info

rbd image 'foo':

size 1024 MB in 256 objects

order 22 (4096 kB objects)

block\_name\_prefix: rb.0.1079.2ae8944a

format: 1

root@snode01:~# rbd --image foo -p rbd info --format json {"name":"foo","size":1073741824,"objects":256,"order":22,"object\_size":4194304,"block\_name\_prefix":"rb.0.1079.2ae8944a","format":1}

修改块大小(为2048MB)

root@snode01:~# rbd resize --image foo --size 2048

Resizing image: 100% complete...done.

root@snode01:~# rbd info foo | egrep size

size 2048 MB in 512 objects

## 删除foo块

root@snode01:~# rbd rm foo -p rbd

Removing image: 100% complete...done.

root@snode01:~# rbd ls rbd

root@snode01:~#

## 创建image格式为2的rbd块

root@snode01:~# rbd -p rbd create foo --size 1024 --image-format 2

root@snode01:~# rbd -p rbd ls

foo

root@snode01:~# rbd -p rbd info foo

rbd image 'foo': size 1024 MB in 256 objects

order 22 (4096 kB objects)

block\_name\_prefix: rbd\_data.10842ae8944a

format: 2

features: layering

## 块拷贝。将块拷贝到一个新块中，不能拷贝到已经存在的块。

root@snode01:~# rbd -p rbd cp foo\_c01 foo\_c02

Image copy: 0% complete...failed. rbd: copy failed: (17) File exists 2014-11-04 21:46:13.380770 7fe57cced780 -1 librbd: rbd image foo\_c02 already exists 2014-11-04 21:46:13.380783 7fe57cced780 -1 librbd: header creation failed

root@snode01:~# rbd -p rbd cp foo\_c01 foo\_c01\_cp

Image copy: 100% complete...done.

root@snode01:~#

## 修改块名字。例子中，将foo\_c01块改名为foo\_c01\_mv。

root@snode01:~# rbd -p rbd mv foo\_c01 foo\_c01\_mv

root@snode01:~#

root@snode01:~# rbd -p rbd ls

foo\_c01\_imp

foo

foo\_c01\_cp

foo\_c01\_mv

foo\_c02 foo\_c03

## 块的导入、导出

格式化块，并挂载到指定目录

root@snode01:~# rbd -p rbd showmapped

id pool image snap device

1 rbd foo\_c01 - /dev/rbd1

root@snode01:~# mkfs.ext4 /dev/rbd1

root@snode01:~# pwd

/root

root@snode01:~# mkdir testrbd

root@snode01:~# mount /dev/rbd1 testrbd/

root@snode01:~# mount | egrep rbd1

/dev/rbd1 on /root/testrbd type ext4 (rw)

准备一个文件hello.txt，内容为hello,world

root@snode01:~/testrbd# echo "hello,world" > hello.txt && ls && cat hello.txt hello.txt hello,world

将块的内容export到文件。例子中将foo\_c01块的内容导出到foo\_c01\_local文件。**注意** 虽然foo\_c01中只有一个hello.txt文件，但导出后的文件大小和块大小相同都是1G。

root@snode01:~# rbd -p rbd export foo\_c01 foo\_c01\_local

Exporting image: 100% complete...done.

root@snode01:~# ls -lh foo\_c01\_local

-rw-r--r-- 1 root root 1.0G Nov 4 21:20 foo\_c01\_local

导出差异部分

1、创建foo\_c01的快照foo\_c01\_s01

root@snode01:~/testrbd# rbd -p rbd snap create --image foo\_c01 --snap foo\_c01\_s01

root@snode01:~/testrbd# rbd -p rbd snap ls foo\_c01

SNAPID NAME SIZE

5 foo\_c01\_s01 1024 MB

2、修改foo\_c01的数据

root@snode01:~/testrbd# echo "This is the appended line" >> hello.txt && cat hello.txt

hello,world

This is the appended line

root@snode01:~/testrbd#

3、导出差异

root@snode01:~# rbd -p rbd export-diff foo\_c01 foo\_c01\_s01\_diff --from-snap foo\_c01\_s01

Exporting image: 100% complete...done.

root@snode01:~# cat foo\_c01\_s01\_diff (二进制文件无法完全显示)

rbd diff v1

f

foo\_c01\_s01s@w

This is the appended line

将文件导入(Import)到一个新块中，新块是在导入的过程中创建的。

root@snode01:~# rbd -p rbd import foo\_c01\_local foo\_c01\_imp

Importing image: 100% complete...done.

root@snode01:~# rbd -p rbd ls | egrep foo\_c01

foo\_c01\_imp

foo\_c01

root@snode01:~# rbd -p rbd info foo\_c01\_imp

rbd image 'foo\_c01\_imp':

size 1024 MB in 256 objects

order 22 (4096 kB objects)

block\_name\_prefix: rb.0.10c7.2ae8944a

format: 1

root@snode01:~#

将普通文件导入到新块

root@snode01:~# rbd -p rbd import pkglist.txt foo\_pkg\_01

Importing image: 100% complete...done.

root@snode01:~# rbd -p rbd ls | egrep foo\_pkg

foo\_pkg\_01

root@snode01:~# rbd -p rbd info foo\_pkg\_01

rbd image 'foo\_pkg\_01':

size 868 bytes in 1 objects

order 22 (4096 kB objects)

block\_name\_prefix: rb.0.10cb.74b0dc51

format: 1

注意，不能把文件导入到一个已经存在的块中。

root@snode01:~# rbd -p rbd import pkglist.txt foo\_c01\_imp

rbd: image creation failed

Importing image: 0% complete...failed.

rbd: import failed: (17) File exists

2014-11-04 21:37:57.236559 7fcfa9f7c780 -1 librbd: rbd image foo\_c01\_imp already exists

导入差异部分

1、块foo\_c01\_s01\_c01和foo\_c01\_s01\_c02都是基于foo\_c01\_s01快照的克隆块，将这两个块映射到本地并挂载到下面两个目录。

root@snode01:~# mount | egrep '(rbd4|rb3)'

/dev/rbd4 on /root/testrbd\_s01\_c01 type ext4 (rw)

/dev/rbd4 on /root/testrbd\_s01\_c02 type ext4 (rw)

2、修改foo\_c01\_s01\_c01块的数据。

root@snode01:~/testrbd\_s01\_c01# echo "This line is added by foo\_c01\_s01\_c01" >> hello.txt

root@snode01:~/testrbd\_s01\_c01# cat hello.txt

hello,world

This line is added by foo\_c01\_s01\_c01

3、导出foo\_c01\_s01\_c01

root@snode01:~# rbd -p rbd export-diff foo\_c01\_s01\_c01 s01\_c01\_diff

Exporting image: 100% complete...done.

4、导入差异部分到foo\_c01\_s01\_c02块(注意映射到本地的块，没有立即生效，需要重新挂载)

root@snode01:~# rbd -p rbd import-diff s01\_c01\_diff foo\_c01\_s01\_c02

Importing image diff: 100% complete...done.

root@snode01:~# cat testrbd\_s01\_c02/hello.txt

hello,world

5、重新挂载被patch的块（即foo\_c01\_s01\_c02）后，才可以看到修改后的结果

root@snode01:~# rbd -p rbd showmapped

id pool image snap device

1 rbd foo\_c01 - /dev/rbd1

2 rbd foo\_c01\_cp - /dev/rbd2

3 rbd foo\_c01\_s01\_c01 - /dev/rbd3

4 rbd foo\_c01\_s01\_c02 - /dev/rbd4

root@snode01:~# umount /dev/rbd4

root@snode01:~# mount /dev/rbd4 testrbd\_s01\_c02

root@snode01:~# cat testrbd\_s01\_c02/hello.txt

hello,world

This line is added by foo\_c01\_s01\_c01

显示快照和块间的差异部分(字节范围)

root@snode01:~# rbd -p rbd diff foo\_c01 --from-snap foo\_c01\_s01

Offset Length Type

397312 4096 data

138424320 4096 data

537235456 12288 data

## 快照的创建、删除、查询、回滚

创建快照(为foo块创建名为foo\_s01的快照)并查看foo块的所有快照

root@snode01:~# rbd -p rbd snap create --image foo --snap foo\_s01

root@snode01:~# rbd -p rbd snap ls foo

SNAPID NAME SIZE

2 foo\_s01 1024 MB

3 foo\_s02 1024 MB

删除快照foo\_s01，注意以--image指定源块名字，以--snap指定快照名字

root@snode01:~# rbd -p rbd snap rm --snap foo\_s01 --image foo

root@snode01:~# rbd -p rbd snap ls foo

SNAPID NAME SIZE

3 foo\_s02 1024 MB

快照回滚。

1、当前块中新增了“This is the appended line”一行

root@snode01:~# cat testrbd/hello.txt

hello,world

This is the appended line

2、快照回滚（此时映射到本地的hello.txt文件内容没有发生改变）

root@snode01:~# rbd -p rbd snap rollback --image foo\_c01 --snap foo\_c01\_s01

Rolling back to snapshot: 100% complete...done.

root@snode01:~# cat testrbd/hello.txt

hello,world

This is the appended line

root@snode01:~#

3、重新挂载块后可以发现rbd1的数据已经回滚到快照的状态

root@snode01:~# umount /dev/rbd1

root@snode01:~# ls testrbd

root@snode01:~# mount /dev/rbd1 testrbd

root@snode01:~# cat testrbd/hello.txt

hello,world

## 块的克隆

被克隆的快照必须是protected的，否则会报下面的错误信息。这个要求的目的是为了保证在克隆过程中，快照不会被删除。

root@snode01:~# rbd -p rbd clone --image foo --snap foo\_s02 --dest foo\_c01

rbd: clone error: (22) Invalid argument

2014-11-04 05:06:16.957593 7fdf8775f780 -1 librbd: parent snapshot must be protected

Protect一个快照(foo\_s02快照)

root@snode01:~# rbd -p rbd snap protect --image foo --snap foo\_s02

Unprotect快照(当快照下存在克隆块时，不允许unprotect快照)

root@snode01:~# rbd -p rbd children --image foo --snap foo\_s02

rbd/foo\_c03

root@snode01:~# rbd -p rbd snap unprotect --image foo --snap foo\_s02

2014-11-04 20:05:42.504123 7f6c645c3780 -1 librbd: snap\_unprotect: can not unprotect; at least 1 child(ren) in pool rbd

rbd: unprotecting snap failed: (16) Device or resource busy

root@snode01:~# rbd -p rbd flatten foo\_c03 Image flatten: 100% complete...done. root@snode01:~# rbd -p rbd children --image foo --snap foo\_s02 root@snode01:~# root@snode01:~# rbd -p rbd snap unprotect --image foo --snap foo\_s02

克隆一个快照

root@snode01:~# rbd -p rbd clone --image foo --snap foo\_s02 --dest foo\_c01

root@snode01:~# rbd list rbd

foo

foo\_c01

root@snode01:~# rbd -p rbd snap list --image foo

SNAPID NAME SIZE

3 foo\_s02 1024 MB

查看指定快照的克隆块(查看基于foo\_s02快照的克隆)

root@snode01:~# rbd -p rbd children --image foo --snap foo\_s02

rbd/foo\_c01

root@snode01:~# rbd children --image foo --snap foo\_s02

rbd/foo\_c01

flatten一个克隆块，使其断开和快照的关系（复制共享数据的块），称为一个独立的块。

root@snode01:~# rbd -p rbd flatten foo\_c02

Image flatten: 100% complete...done.

root@snode01:~# rbd -p rbd children --image foo --snap foo\_s02

root@snode01:~#

## 块映射（map）

将rbd块（foo\_c01)映射到本地

root@snode01:~# rbd -p rbd map --image foo\_c01

root@snode01:~#

root@snode01:~# rbd -p rbd showmapped

id pool image snap device

1 rbd foo\_c01 - /dev/rbd1

root@snode01:~#

root@snode01:~# tree /dev/rbd

/dev/rbd

└── rbd

└── foo\_c01 -> ../../rbd1

1 directory, 1 file

可以将一个rbd块多次映射到本地

root@snode01:~# rbd -p rbd map --image foo\_c01 -o share

root@snode01:~# rbd -p rbd map --image foo\_c01 -o crc

root@snode01:~# rbd -p rbd showmapped

id pool image snap device

1 rbd foo\_c01 - /dev/rbd1

2 rbd foo\_c01 - /dev/rbd2

解除块映射

root@snode01:~# rbd -p rbd unmap /dev/rbd1

root@snode01:~# rbd -p rbd showmapped

root@snode01:~#

## 块锁（加锁、解锁、查询）

基本概念：锁id、锁Tag、锁类型（独占锁和共享锁）

对块加锁。默认的是独占锁(exclusive lock)，并且不会protect块删除。如果要添加共享锁，可以使用-shared选项。例子中，foo\_lock01代表锁的名字。

root@snode01:~# rbd -p rbd lock list foo

root@snode01:~# rbd -p rbd lock add foo foo\_lock01

root@snode01:~# rbd -p rbd lock list foo

There is 1 exclusive lock on this image.

Locker ID Address

client.4280 foo\_lock01 192.168.7.10:0/1020463

移除锁。

root@snode01:~# rbd -p rbd lock remove foo foo\_lock01 client.4280

root@snode01:~# rbd -p rbd lock list foo

root@snode01:~#

添加共享锁。注意和独占锁相比，多了个lock tag。

root@snode01:~# rbd -p rbd lock add foo foo\_lock01 --shared foo\_tag01

rbd: lock is alrady held by someone else with a different tag

root@snode01:~# rbd -p rbd lock add foo foo\_lock02 --shared foo\_tag01

root@snode01:~# rbd -p rbd lock list foo

There is 1 shared lock on this image.

Lock tag: foo\_tag01

Locker ID Address

client.4287 foo\_lock02 192.168.7.10:0/1020627

这时,可以多个client可以同时添加共享锁。

root@snode01:~# rbd -p rbd lock add foo foo\_lock02 --shared foo\_tag01

root@snode01:~# rbd -p rbd lock list foo

There are 2 shared locks on this image.

Lock tag: foo\_tag01

Locker ID Address

client.4287 foo\_lock02 192.168.7.10:0/1020627

client.4289 foo\_lock02 192.168.7.10:0/1020667

**注意** 加锁和protect的区别，加锁不能保证块不被删除。