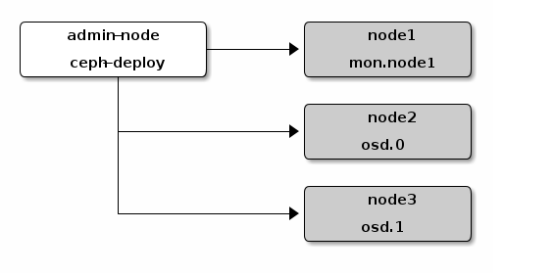
# Ceph 搭建

2018/10/4 徐志威

Ceph简介:

Ceph是一个分布式存储，可以提供对象存储、块存储和文件存储，其中对象存储和块存储可以很好地和各大云平台集成。一个Ceph集群中有Monitor节点、MDS节点（可选，用于文件存储）、至少两个OSD守护进程。   
Ceph OSD：OSD守护进程，用于存储数据、处理数据拷贝、恢复、回滚、均衡，并通过心跳程序向Monitor提供部分监控信息。一个Ceph集群中至少需要两个OSD守护进程。   
Monitor：维护集群的状态映射信息，包括monitor、OSD、Placement Group（PG）。还维护了Monitor、OSD和PG的状态改变历史信息。   
MDS：存储Ceph文件系统的元数据.

Ceph逻辑架构

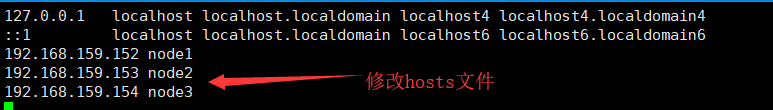


Ceph 环境准备

注意，以下操作千万不要弄错，因为很多错误网上八成都找不到！

准备三台服务器并修改主机名和hosts文件，这里我用的是centos7系统，建议每台2g内存，不然可能会出现无法预知的问题

三台服务器都修改,并关闭防火墙和selinux



**在各节点上安装启用软件仓库，启用可选软件库**

# yum install -y yum-utils && yum-config-manager --add-repo https://dl.fedoraproject.org/pub/epel/7/x86\_64/ && yum install --nogpgcheck -y epel-release && rpm --import /etc/pki/rpm-gpg/RPM-GPG-KEY-EPEL-7 && rm /etc/yum.repos.d/dl.fedoraproject.org\*

# yum install yum-plugin-priorities -y

安装ntp时间服务并开启

# yum install ntp –y

# systemctl start ntpd

**设置三台服务器的ssh互信**

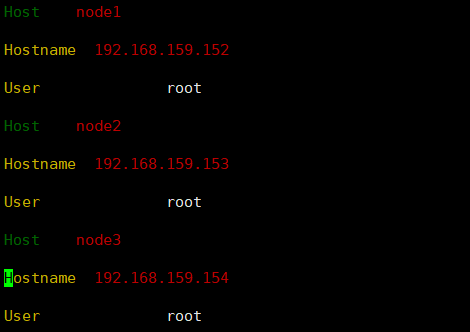
# ssh-kengen

# ssh-copy-id  root@node1

# ssh-copy-id  root@node2

# ssh-copy-id  root@node3

**在管理节点node1 上修改~/.ssh/config文件(如果没有则创建)增加一下内容**



**在各节点上创建ceph 源**

# vim /etc/yum.repos.d/ceph.repo

[Ceph]  
name=Ceph packages for $basearch  
baseurl=http://mirrors.163.com/ceph/rpm-jewel/el7/$basearch  
enabled=1  
gpgcheck=0  
type=rpm-md  
gpgkey=https://mirrors.163.com/ceph/keys/release.asc  
priority=1

[Ceph-noarch]  
name=Ceph noarch packages  
baseurl=http://mirrors.163.com/ceph/rpm-jewel/el7/noarch  
enabled=1  
gpgcheck=0  
type=rpm-md  
gpgkey=https://mirrors.163.com/ceph/keys/release.asc  
priority=1

[ceph-source]  
name=Ceph source packages  
baseurl=http://mirrors.163.com/ceph/rpm-jewel/el7/SRPMS  
enabled=1  
gpgcheck=0  
type=rpm-md  
gpgkey=https://mirrors.163.com/ceph/keys/release.asc  
priority=1

**在管理节点nod1上进行安装准备**

# mkdir ceph-cluster

# cd ceph-cluster

安装ceph –deploy

# yum install ceph-deploy –y

**安装ceph创建集群**

进入到创建ceph-cluster文件夹下，执行命令

# ceph-deploy new node1 node2 node3

在生成的ceph.conf中加入（写入[global] 段下）

osd pool default size = 2

安装ceph

# ceph-deploy install node1 node2 node3

配置初始 monitor(s)、并收集所有密钥

# ceph-deploy mon create-initial

添加两个 OSD ，登录到 Ceph 节点、并给 OSD 守护进程创建一个目录

# ssh node2

# mkdir /var/local/osd0

# exit

# ssh node3

# mkdir /var/local/osd1

# exit

然后，从管理节点执行 ceph-deploy 来准备 OSD

# ceph-deploy osd prepare node2:/var/local/osd0 node3:/var/local/osd1

最后，激活 OSD

# chmod 777 -R /var/local/osd0/

# chmod 777 -R /var/local/osd1/

# ceph-deploy osd activate node2:/var/local/osd0 node3:/var/local/osd1

确保你对 ceph.client.admin.keyring 有正确的操作权限

# chmod +r /etc/ceph/ceph.client.admin.keyring

检查集群的健康状况

# ceph -s

# ceph health



部署rgw网关

# ceph-deploy rgw create node1

验证ceph

当显示状态健康时，可写入数据并查看数据。   
创建一个普通文本文件test.txt，并向其写入数据。   
创建一个pool。格式为：rados mkpool {pool-name}，执行：

# rados mkpool data

将文件写入pool。格式为：rados put {object-name} {file-path} –pool={pool-name}，执行：

# cd

# vim text.txt

随便写入一点东西，这里我写入test

# rados put test-object-1 test.txt --pool=data

如果文件不大，应该很快就写完并成功了。如果卡主较长时间，则可能是出错了，需要排查问题。   
查看文件是否存在于pool中，格式为：rados -p {pool-name} ls，执行：

# rados -p data ls



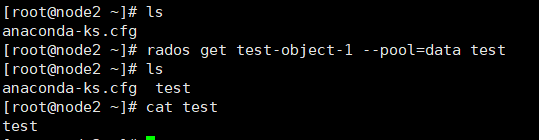
确定文件的位置。格式为：ceph osd map {pool-name} {object-name}，执行：

# ceph osd map data test-object-1



从pool中读取文件,这里我用node2读取。格式为：rados get {object-name} –pool={pool-name} {file-path} ，执行：

# rados get test-object-1 --pool=data test

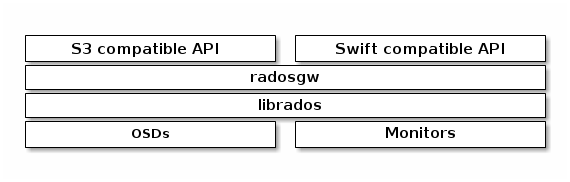


可比对读出的文件myfile和原文件testfile.txt是否相同，执行命令：diff myfile testfile.txt。   
从pool中删除文件。格式为：rados rm {object-name} –pool={pool-name}，执行：

# rados rm test-object-1 --pool=data

Ceph rgw搭建

Ceph RGW基于librados，是为应用提供RESTful类型的对象存储接口。RGW提供两种类型的接口：   
　　1) S3：兼容Amazon S3RESTful API；   
　　2) Swift：兼容OpenStack Swift API(即云计算平台)；



S3和Swift API共享同一个命名空间，所以可以使用两种API访问相同的数据

然后开始部署ceph rgw

# cd ceph-cluster/

安装CEPH OBJECT GATEWAY

实际上可以部署多个rgw节点，但由于我的环境中只有一个，所以

# ceph-deploy install -rgw node

管理RGW节点

Ceph CLI工具需要在管理员模式下运行，因此需要执行以下命令：

# ceph-deploy admin node1

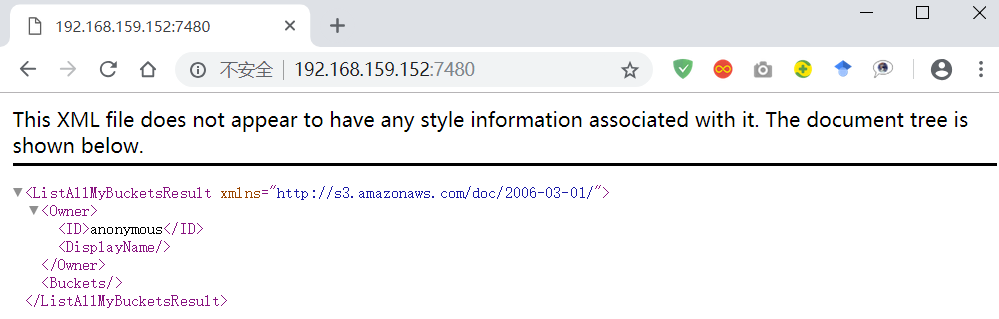
安装RGW实例

# ceph-deploy rgw create node1

现在我们可以通过7480端口来访问我的rgw节点node1的ip

<http://192.168.159.152:7480>

如果运行正常则会返回

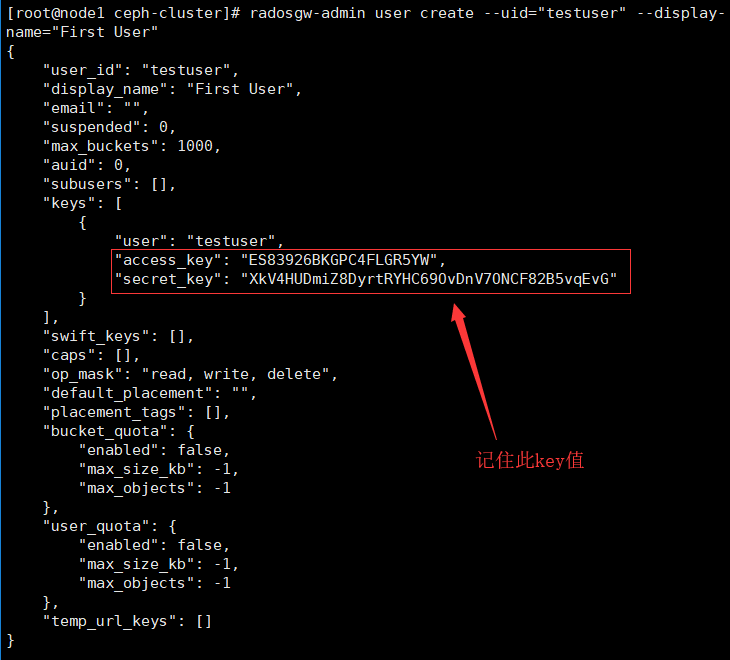


创建S3用户

想正常的访问RGW，需要创建相应的RGW用户，并赋予相应的权限，radosgw-admin命令实现了这些功能。   
　　执行下面命令，来创建一个名为testuser的用户

# radosgw-admin user create --uid="testuser" --display-name="First User"

产生如下输出



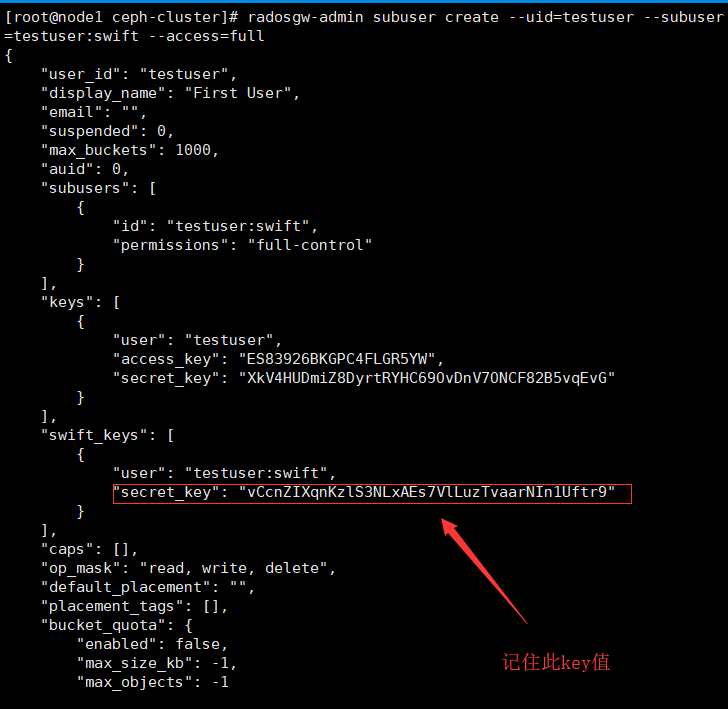
注意：需要记住返回结果中keys->access\_key和keys->secret\_key的值，用于S3接口访问确认

创建Swift用户

Swift用户是作为子用户subuser被创建的，执行以下命令：

# radosgw-admin subuser create --uid=testuser --subuser=testuser:swift --access=full

产生如下输出



注意：需要记住返回结果中swift\_keys->secret\_key的值，用于Swift接口访问确认

测试S3接口

需要创建一个Python测试脚本来测试S3访问。该脚本会连接RGW，创建一个bucket并列出所有的bucket。其中，变量access\_key和secret\_access的值，来自于创建S3用户命令时，radosgw-admin命令返回的keys->access\_key和keys->secret\_key。   
　　执行以下步骤，首先安装python-boto库，该库用于连接S3

# yum install python-boto –y

创建并编辑Python脚本：

# cd

# vim s3test.py

#!/usr/bin/python

import boto.s3.connection

access\_key = 'I0PJDPCIYZ665MW88W9R'

secret\_key = 'dxaXZ8U90SXydYzyS5ivamEP20hkLSUViiaR+ZDA'

conn = boto.connect\_s3(

aws\_access\_key\_id=access\_key,

aws\_secret\_access\_key=secret\_key,

host='{hostname}', port={port},

is\_secure=False, calling\_format=boto.s3.connection.OrdinaryCallingFormat(),)

bucket = conn.create\_bucket('my-new-bucket')

for bucket in conn.get\_all\_buckets():

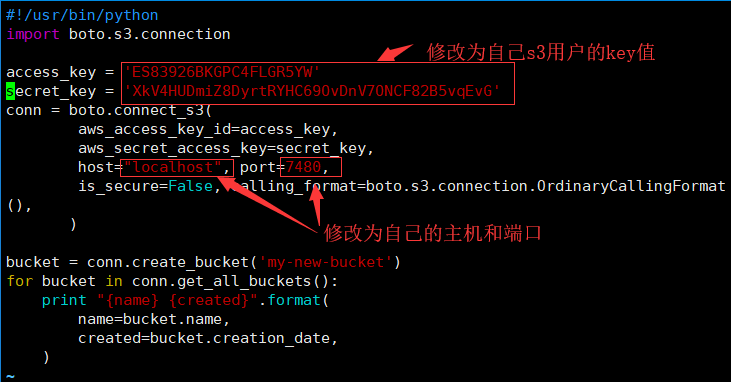
print "{name} {created}".format(

name=bucket.name,

created=bucket.creation\_date,

)

替换{hostname}、{port}为真实环境的hostname（或者ip）和RGW的端口。执行命令进行测试，例如：



# python s3test.py

输出应该如下



测试Swift接口

Swfit的访问可以通过命令行。首先安装相关软件包：

# yum install python-setuptools

# yum install python-swiftclient

如果python-swiftclient包yum源里没有的：

git clone https://github.com/openstack/python-swiftclient.git

# cd python-swiftclient

# python setup.py install

执行下列命令，范文Swift接口：

# swift -A http://{IP ADDRESS}:{port}/auth/1.0 -U testuser:swift -K '{swift\_secret\_key}' list

替换{IP ADDRESS}、{port}、{swift\_secret\_key}等相关参数，其中{swift\_secret\_key}为创建Swift用户时，radosgw-admin命令返回的swift\_keys->secret\_key的值。例如，输出：

# swift -A http://localhost:7480/auth/1.0 -U testuser:swift -K 'vCcnZIXqnKzlS3NLxAEs7VlLuzTvaarNIn1Uftr9' list

结果如下



Rgw测试完成

块存储rbd节点

创建映像，创建前先ceph-s查看集群状态是否健康，命令如下

# rbd create rbd/test --size 2G --image-format 2 --image-feature  layering

这里之所以不直接使用# rbd create rbd/test --size 2G --image-format 2 创建，是因为直接创建需要更高的内核版本，貌似需要3.1的内核版本，所以这里# --image-feature   选项指定使用特性。

# rbd ls

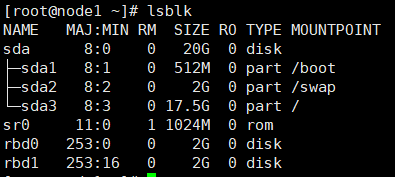


Test1是我之前创建的一个映像

这里直接映射到磁盘

# rbd map rbd/test





这里直接格式化挂载使用即可，我就不演示了