

第 13 章 k8s 对接 ceph 实现持久化存储

本节所讲内容:

13.1 初始化安装 ceph 的实验环境

13.2 安装 ceph 集群

13.3 激活 ceph osd

实战 1: 测试 k8s 挂载 ceph rbd

实战 2: 基于 ceph rbd 生成 pv

实战 3: 基于 storageclass 动态生成 pv

实战 4: k8s 挂载 cephfs: 作业

13.1 Ceph 概述

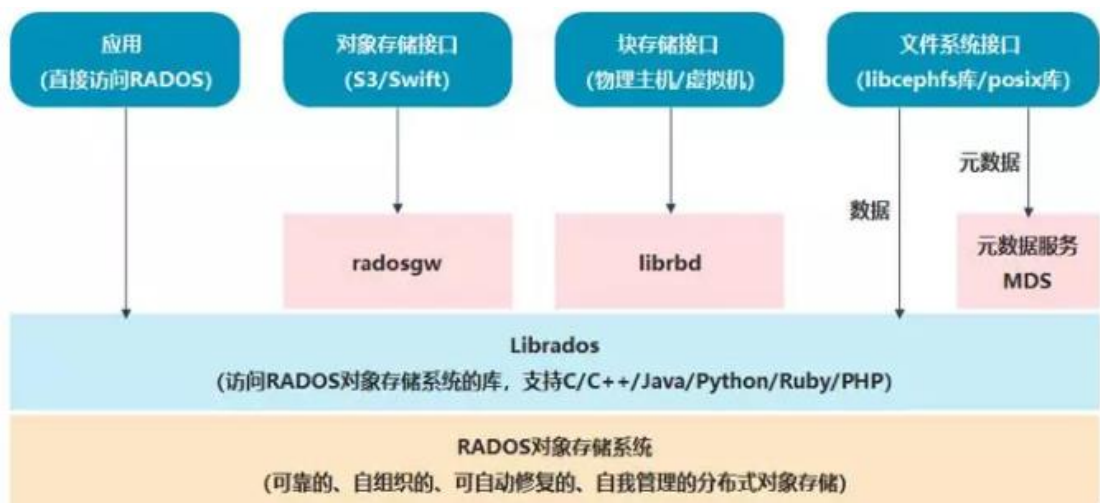
13.1.1 Ceph 介绍

Ceph 是一个开源的分布式存储系统, 设计初衷是提供较好的性能、可靠性和可扩展性。它还是一个可靠、自动重均衡、自动恢复的分布式存储系统。主要优点是分布式存储, 在存储每一个数据时, 都会通过计算得出该数据存储的位置, 尽量将数据分布均衡, 不存在传统的单点故障的问题, 可以水平扩展。

ceph 官方文档 <http://docs.ceph.org.cn/>

ceph 中文开源社区 <http://ceph.org.cn/>

Ceph 是一个开源的分布式文件系统。因为它还支持块存储、对象存储, 所以很自然的被用做云计算框架 openstack 或 cloudstack 整个存储后端。当然也可以单独作为存储, 例如部署一套集群作为对象存储、SAN 存储、NAS 存储等。



RADOS 自身是一个完整的分布式对象存储系统, 它具有可靠、智能、分布式等特性, Ceph 的高可靠、高可扩展、高性能、高自动化都是由这一层来提供的, 用户数据的存储最终也都是通过这一层来进行存储的, RADOS 可以说就是 Ceph 的核心组件。

RADOS 系统主要由两部分组成, 分别是 OSD 和 Monitor。

基于 RADOS 层的上一层是 LIBRADOS, LIBRADOS 是一个库, 它允许应用程序通过访问该库来与 RADOS 系统进行交互, 支持多种编程语言, 比如 C、C++、Python 等。

基于 LIBRADOS 层开发的又可以看到有三层, 分别是 RADOSGW、RBD 和 CEPH FS。

RADOSGW: RADOSGW 是一套基于当前流行的 RESTFUL 协议的网关, 并且兼容 S3 和 Swift。

RBD: RBD 通过 Linux 内核客户端和 QEMU/KVM 驱动来提供一个分布式的块设备。

CEPH FS: CEPH FS 通过 Linux 内核客户端和 FUSE 来提供一个兼容 POSIX 的文件系统。

ceph 支持

- 1、对象存储: 即 radosgw,兼容 Swift 和 S3 接口。通过 rest api 上传、下载文件。
- 2、文件系统: posix 接口。可以将 ceph 集群看做一个共享文件系统挂载到本地。
- 3、块存储: 即 rbd。有 kernel rbd 和 librbd 两种使用方式。支持快照、克隆。相当于一块硬盘挂到本地, 用法和用途和硬盘一样。

Ceph 相比其它分布式存储有哪些优点?

1、统一存储

虽然 ceph 底层是一个分布式文件系统, 但由于在上层开发了支持对象和块的接口。所以在开源存储软件中, 能够一统江湖。

2、高扩展性

扩容方便、容量大。能够管理上千台服务器、EB 级的容量。

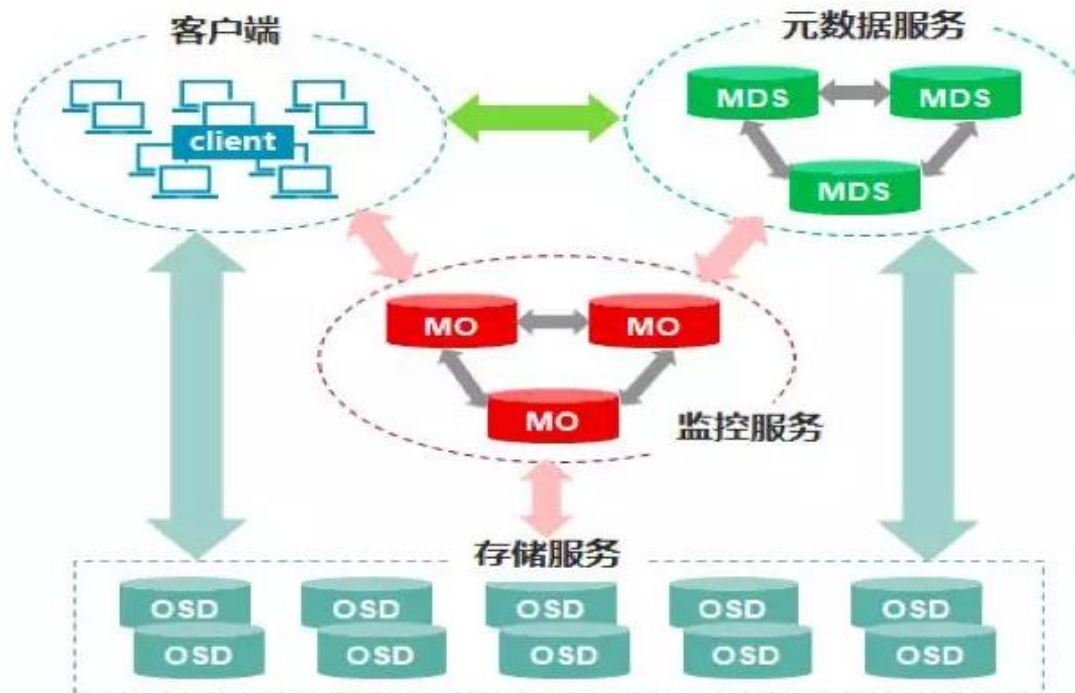
3、可靠性强

支持多份强一致性副本。副本能够跨主机、机架、机房、数据中心存放。所以安全可靠。存储节点可以自动管理、自动修复。无单点故障, 容错性强。

4、高性能

因为是多个副本, 因此在读写操作时候能够做到高度并行化。理论上, 节点越多, 整个集群的 IOPS 和吞吐量越高。另外一点 ceph 客户端读写数据直接与存储设备(osd) 交互。

13.1.2 Ceph 各组件介绍



Ceph 的核心组件包括 Client 客户端、MON 监控服务、MDS 元数据服务、OSD 存储服务，各组件功能如下：

1、Client 客户端：负责存储协议的接入，节点负载均衡。

2、MON (Monitor) 监控服务：监控整个集群 Cluster map 的状态，维护集群的 cluster MAP 二进制表，保证集群数据的一致性，维护集群状态的映射，包括监视器映射，管理器映射，OSD 映射，MDS 映射和 CRUSH 映射。这些映射是 Ceph 守护程序相互协调所需的关键群集状态。监视器还负责管理守护程序和客户端之间的身份验证。通常至少需要三个监视器才能实现冗余和高可用性。

注：元数据：保存了子目录和子文件的名称 及 inode 编号的数据，通过元数据，可以找到对应的真实的数据，及真实数据所在的服务器。

3、MDS 元数据服务（可选）：是元数据的内存缓存，为了加快元数据的访问。保存了文件系统的元数据(对象里保存了子目录和子文件的名称和 inode 编号)，还保存 cephfs 日志 journal，日志是用来恢复 mds 里的元数据缓存，重启 mds 的时候会通过 replay 的方式从 osd 上加载之前缓存的元数据。

在 ceph 中，**元数据是存储在 osd 节点中的**，mds 类似于元数据的内存缓存服务器，加快访问速度。

4、OSD (对象存储守护程序)：主要功能是存储数据、复制数据、平衡数据、恢复数据，以及与通过检查其他 Ceph OSD 守护程序的心跳来向 Ceph 监视器和管理器提供一些监视信息。通常至少需要 3 个 Ceph OSD 才能实现冗余和高可用性。

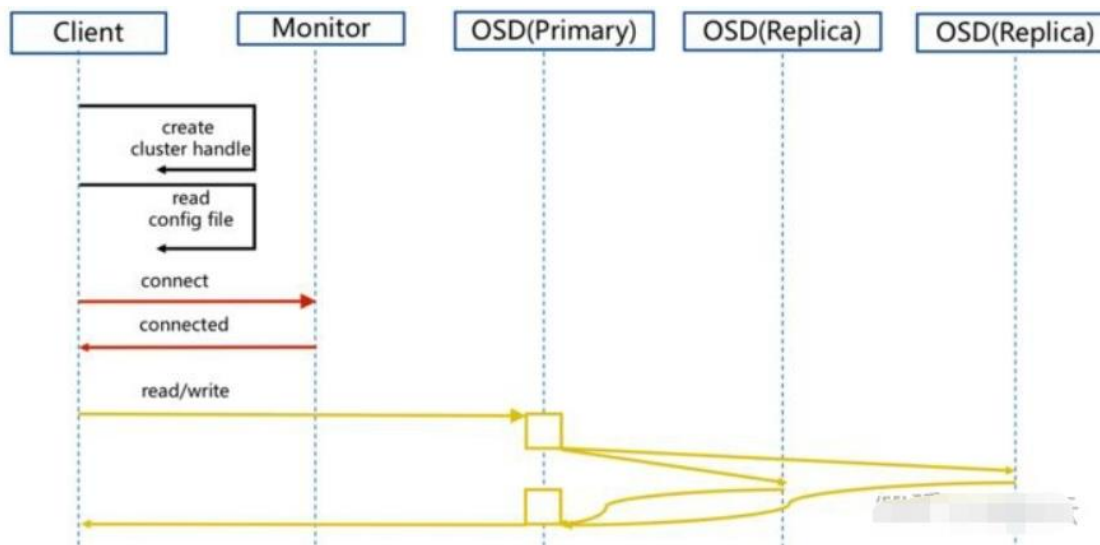
通常来说，一块磁盘和该磁盘对应的守护进程称为一个 OSD。守护进程的作用是从该磁盘读取和写入数据。该磁盘可以是一个硬盘或者 SSD 盘或者 RAID0，总之是一个逻辑磁盘。如果一个节点只有一个守护进程和对应的磁盘，那么该 OSD 就成了一个节点。通常一个节点有多个 OSD 守护进程和多个磁盘，所以通常来说 OSD 不是一个节点。

Ceph 要求必须是奇数个 Monitor 监控节点，一般建议至少是 3 个（如果是自己私下测试玩玩的话，可以是 1 个，但是生产环境绝不建议 1 个）用于维护和监控整个集群的状态，每个 Monitor 都有一

个 Cluster Map, 只要有这个 Map, 就能够清楚知道每个对象存储在什么位置了。客户端会先 tcp 连接到 Monitor, 从中获取 Cluster Map, 并在客户端进行计算, 当知道对象的位置后, 再直接与 OSD 通信 (去中心化的思想)。OSD 节点平常会向 Monitor 节点发送简单心跳, 只有当添加、删除或者出现异常状况时, 才会自动上报信息给 Monitor。

MDS 是可选的, 只有需要使用 Ceph FS 的时候才需要配置 MDS 节点。在 Ceph 中, 元数据也是存放在 OSD 中的, MDS 只相当于元数据的缓存服务器。

在 Ceph 中, 如果要写数据, 只能向主 OSD 写, 然后再由主 OSD 向从 OSD 同步地写, 只有当从 OSD 返回结果给主 OSD 后, 主 OSD 才会向客户端报告写入完成的消息。如果要读数据, 不会使用读写分离, 而是也需要先向主 OSD 发请求, 以保证数据的强一致性。



MON (Monitor) 监控服务, OSD (对象存储守护程序), MDS 元数据服务只有需要使用 CEPHFS (ceph 的文件系统), 才需要配置 MDS 节点。

Ceph 核心组件及概念介绍

Monitor :保存, 同步 OSD 元数据 (可以是 OSD 中的一员充当 Monitor) OSD

OSD 全称 Object Storage Device, 也就是负责响应客户端请求返回具体数据的进程。一个 Ceph 集群一般都有很多个 OSD。

MDS 全称 Ceph Metadata Server, 是 CephFS 服务依赖的元数据服务。

Object Ceph 最底层的存储单元是 Object 对象, 每个 Object 包含元数据和原始数据。

PG 全称 Placement Groups, 是一个逻辑的概念, 一个 PG 包含多个 OSD。引入 PG 这一层其实是为了更好的分配数据和定位数据。

RADOS 全称 Reliable Autonomic Distributed Object Store, 是 Ceph 集群的精华, 用户实现数据分配、Failover 等集群操作。

Librados 是 Rados 提供库, 因为 RADOS 是协议很难直接访问, 因此上层的 RBD、RGW 和 CephFS 都是通过 librados 访问的, 目前提供 PHP、Ruby、Java、Python、C 和 C++ 支持。

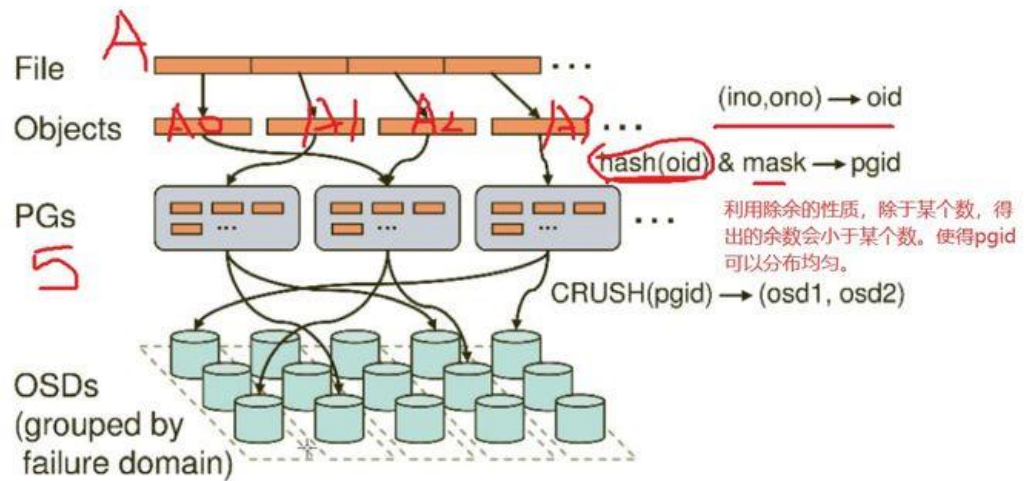
CRUSH 是 Ceph 使用的数据分布算法, 类似一致性哈希, 让数据分配到预期的地方。

RBD 全称 RADOS block device, 是 Ceph 对外提供的块设备服务。

RGW 全称 RADOS gateway, 是 Ceph 对外提供的对象存储服务, 接口与 S3 和 Swift 兼容。

CephFS 全称 Ceph File System, 是 Ceph 对外提供的文件系统服务。

Ceph IO 算法流程



知乎 @我的云

首先文件被切分为多个 Objects 并以 oid 标记。

引入 PG 逻辑概念 (为了防止 oid 存储时混乱不好搜索), 并用 crush 算法生成 pgid, 把 oid 均匀分布给 OSD 中存储。该算法利用求余数的概念, 使得可以分配均匀。

Ceph 为什么使用 Object 存储

块存储: 采用 SAN 架构组网时, 光纤交换机, 造价成本高。 主机之间无法共享数据。

文件系统: 读写速率低。 传输速率慢。

对象存储: 具备块存储的读写高速。 具备文件存储的共享等特性。

13.2 初始化实验环境

机器配置:

Centos7.6

网络模式: NAT

准备三台机器, 每台机器需要三个硬盘, 配置 4GiB/6vCPU/60G

master1-admin 是管理节点 : 192.168.40.200

 master1-admin

 开启此虚拟机

 编辑虚拟机设置

▼ 设备

 内存	4 GB
 处理器	6
 硬盘 (SCSI)	100 GB
 硬盘 2 (SCSI)	60 GB
 硬盘 3 (SCSI)	60 GB
 CD/DVD (IDE)	正在使用文件 D:...
 网络适配器	NAT
 USB 控制器	存在
 声卡	自动检测
 打印机	存在
 显示器	自动检测

node1-monitor 是监控节点: 192.168.40.201

 node1-monitor

 开启此虚拟机

 编辑虚拟机设置

▼ 设备

 内存	4 GB
 处理器	6
 硬盘 (SCSI)	100 GB
 硬盘 2 (SCSI)	60 GB
 硬盘 3 (SCSI)	60 GB
 CD/DVD (IDE)	正在使用文件 D:...
 网络适配器	NAT
 USB 控制器	存在
 声卡	自动检测
 打印机	存在
 显示器	自动检测

node2-osd 是对象存储节点: 192.168.40.202



1、配置静态 IP

把虚拟机或者物理机配置成静态 ip 地址, 这样机器重新启动后 ip 地址也不会发生改变。以 master1-admin 主机为例, 修改静态 IP:

修改/etc/sysconfig/network-scripts/ifcfg-ens33 文件, 变成如下:

```
TYPE=Ethernet
PROXY_METHOD=none
BROWSER_ONLY=no
BOOTPROTO=static
IPADDR=192.168.40.200
NETMASK=255.255.255.0
GATEWAY=192.168.40.2
DNS1=192.168.40.2
DEFROUTE=yes
IPV4_FAILURE_FATAL=no
IPV6INIT=yes
IPV6_AUTOCONF=yes
IPV6_DEFROUTE=yes
IPV6_FAILURE_FATAL=no
IPV6_ADDR_GEN_MODE=stable-privacy
NAME=ens33
DEVICE=ens33
ONBOOT=yes
```

#修改配置文件之后需要重启网络服务才能使配置生效, 重启网络服务命令如下:

```
service network restart
```

注: /etc/sysconfig/network-scripts/ifcfg-ens33 文件里的配置说明:

```
NAME=ens33    #网卡名字, 跟 DEVICE 名字保持一致即可
DEVICE=ens33   #网卡设备名, 大家 ip addr 可看到自己的这个网卡设备名, 每个人的机器可能
这个名字不一样, 需要写自己的
BOOTPROTO=static    #static 表示静态 ip 地址
ONBOOT=yes    #开机自启动网络, 必须是 yes
IPADDR=192.168.40.200    #ip 地址, 需要跟自己电脑所在网段一致
NETMASK=255.255.255.0    #子网掩码, 需要跟自己电脑所在网段一致
GATEWAY=192.168.40.2    #网关, 在自己电脑打开 cmd, 输入 ipconfig /all 可看到
DNS1=192.168.40.2    #DNS, 在自己电脑打开 cmd, 输入 ipconfig /all 可看到
```

2、配置主机名

在 192.168.40.200 上执行如下:

```
hostnamectl set-hostname master1-admin
```

在 192.168.40.201 上执行如下:

```
hostnamectl set-hostname node1-monitor
```

在 192.168.40.202 上执行如下:

```
hostnamectl set-hostname node2-osd
```

3、配置 hosts 文件

修改 master1-admin、node1-monitor、node2-osd 机器的/etc/hosts 文件, 增加如下三行:

```
192.168.40.200    master1-admin
192.168.40.201    node1-monitor
192.168.40.202    node2-osd
```

4、配置互信

生成 ssh 密钥对

```
[root@master1-admin ~]# ssh-keygen -t rsa    #一路回车, 不输入密码
```

把本地的 ssh 公钥文件安装到远程主机对应的账户

```
[root@master1-admin ~]# ssh-copy-id node1-monitor
```

```
[root@master1-admin ~]# ssh-copy-id node1-monitor
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/root/.ssh/id_rsa.pub"
The authenticity of host 'node1-monitor (192.168.40.201)' can't be established.
ECDSA key fingerprint is SHA256:R6n9KU5fY7acX+W90eSGPC8PHxSyl2ghdAXnNnuH0BI.
ECDSA key fingerprint is MD5:ef:6f:c3:9e:46:d8:8e:dd:62:c2:89:c2:c6:99:12:5f.
Are you sure you want to continue connecting (yes/no)? yes
```

```
[root@master1-admin ~]# ssh-copy-id node2-osd
[root@master1-admin ~]# ssh-copy-id master1-admin
```

```
[root@node1-monitor ~]# ssh-keygen    #一路回车, 不输入密码
```

把本地的 ssh 公钥文件安装到远程主机对应的账户

```
[root@node1-monitor ~]# ssh-copy-id master1-admin
[root@node1-monitor ~]# ssh-copy-id node1-monitor
[root@node1-monitor ~]# ssh-copy-id node2-osd
```

```
[root@node2-osd ~]# ssh-keygen    #一路回车, 不输入密码
```


把本地的 ssh 公钥文件安装到远程主机对应的账户

```
[root@node2-osd ~]# ssh-copy-id master1-admin
```

```
[root@node2-osd ~]# ssh-copy-id node1-monitor
```

```
[root@node2-osd ~]# ssh-copy-id node2-osd
```

5、关闭防火墙

```
[root@master1-admin ~]# systemctl stop firewalld ; systemctl disable firewalld
```

```
[root@node1-monitor ~]# systemctl stop firewalld ; systemctl disable firewalld
```

```
[root@node2-osd ~]# systemctl stop firewalld ; systemctl disable firewalld
```

6、关闭 selinux

#临时关闭

```
setenforce 0
```

#永久关闭

```
sed -i 's/SELINUX=enforcing/SELINUX=disabled/g' /etc/selinux/config
```

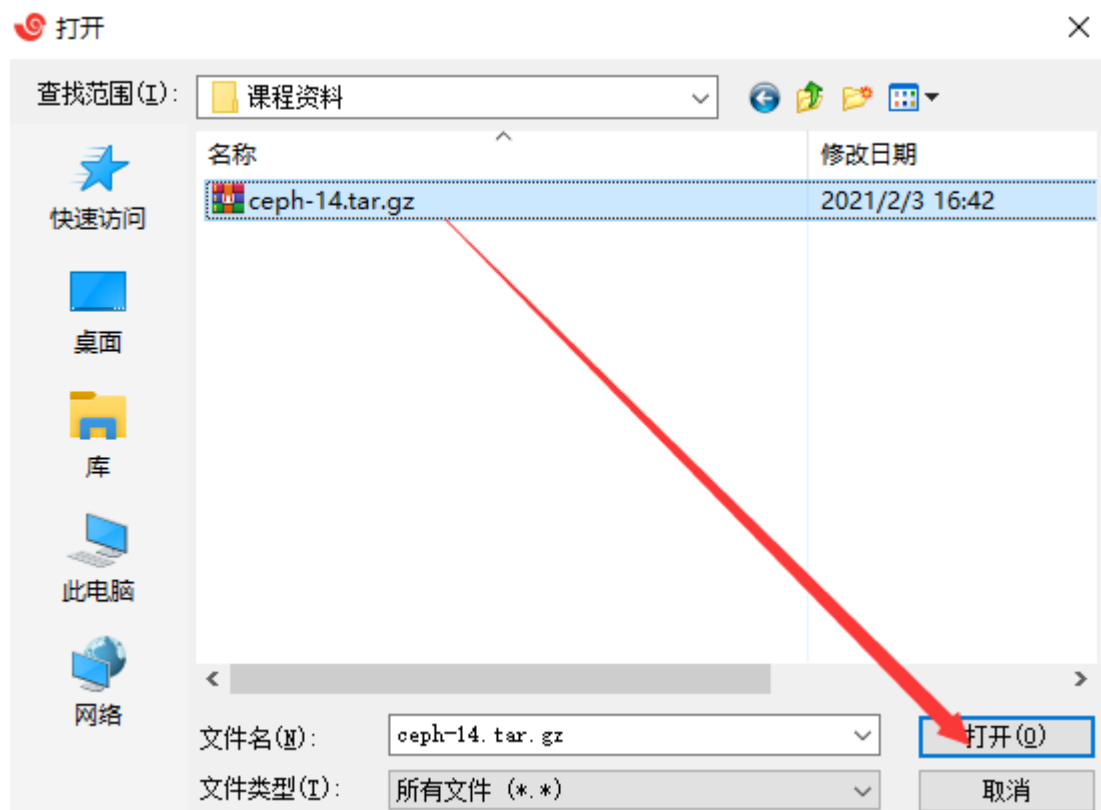
#注意: 修改 selinux 配置文件之后, 重启机器, selinux 才能永久生效

7、配置 Ceph 安装源

方法 1: 离线安装

创建本地 yum 源

```
[root@master1-admin ~]# rz
```



```
[root@master1-admin ~]# tar xf ceph-14-v2.tar.gz -C /opt/
```

```
[root@master1-admin ~]# vim /etc/yum.repos.d/ceph.repo
```

```
[ceph]
```

```
name=ceph
baseurl=file:///opt/ceph-14
enable=1
gpgcheck=0
```

复制离线 yum 仓库至 node1-monitor

```
[root@ master1-admin ~]# scp -r /opt/ceph-14/ root@node1-monitor:/opt/
[root@ master1-admin ~]# scp /etc/yum.repos.d/ceph.repo root@node1-monitor:/etc/yum.repos.d/
```

复制离线 yum 仓库至 node2-osd

```
[root@ master1-admin ~]# scp -r /opt/ceph-14/ root@node2-osd:/opt/
[root@ master1-admin ~]# scp /etc/yum.repos.d/ceph.repo root@node2-osd:/etc/yum.repos.d/
```

复制离线 yum 仓库至 k8s 控制节点 xuegod63

```
[root@ master1-admin ~]# scp -r /opt/ceph-14/ root@192.168.40.63:/opt/
[root@ master1-admin ~]# scp /etc/yum.repos.d/ceph.repo
192.168.40.63:/etc/yum.repos.d/
```

复制离线 yum 仓库至 k8s 工作节点 xuegod64

```
[root@ master1-admin ~]# scp -r /opt/ceph-14/ root@192.168.40.64:/opt/
[root@ master1-admin ~]# scp /etc/yum.repos.d/ceph.repo
192.168.40.64:/etc/yum.repos.d/
```

方法 2: 配置在线 ceph 源

在线安装

```
[root@ master1-admin ~]# vim /etc/yum.repos.d/ceph.repo
[ceph]
name=ceph
baseurl=https://mirrors.aliyun.com/ceph/rpm-nautilus/el7/x86_64/
gpgcheck=0
priority=1
enable=1
[ceph-noarch]
name=cephnoarch
baseurl=https://mirrors.aliyun.com/ceph/rpm-nautilus/el7/noarch/
gpgcheck=0
priority=1
enable=1
[ceph-source]
name=Ceph source packages
baseurl=https://mirrors.aliyun.com/ceph/rpm-nautilus/el7/SRPMS/
```

```
gpgcheck=0
priority=1
enable=1
```

```
[root@ master1-admin ~]# scp /etc/yum.repos.d/ceph.repo root@node1-
monitor:/etc/yum.repos.d/
```

```
[root@ master1-admin ~]# scp /etc/yum.repos.d/ceph.repo root@node2-
osd:/etc/yum.repos.d/
```

在线安装必须安装 epel 源

```
[root@ master1-admin ~]# yum install -y epel-release
```

8、配置机器时间跟网络时间同步，在 ceph 的每台机器上操作

```
service ntpd stop
ntpdate cn.pool.ntp.org
crontab -e
*/1 * * * /usr/sbin/ntpdate cn.pool.ntp.org
```

```
service crond restart
```

9、安装基础软件包

在 ceph 的每台节点上都操作

```
yum install -y yum-utils device-mapper-persistent-data lvm2 wget net-tools nfs-utils
lrzsz gcc gcc-c++ make cmake libxml2-devel openssl-devel curl curl-devel unzip sudo
ntp libaio-devel wget vim ncurses-devel autoconf automake zlib-devel python-devel
epel-release openssh-server socat ipvsadm conntrack ntpdate telnet deltarpm
```

10、更新内核

所有主机更新 Centos 内核，Ceph 的 iscsi 网关组件最低需要版本 v4.16。最好离线安装
导入 Public Key

```
[root@master1-admin ~]# rpm --import https://www.elrepo.org/RPM-GPG-KEY-elrepo.org
安装 elrepo 源
```

```
[root@master1-admin ~]# yum install https://www.elrepo.org/elrepo-release-7.0-
3.el7.elrepo.noarch.rpm -y
```

安装新版本内核 (ml 主线版本为最新 5.x 内核，长期支持版本 lt 为 4.x 内核)

```
[root@ master1-admin ~]# yum --enablerepo=elrepo-kernel install kernel-lt-devel kernel-lt
-y
```

时间可能会比较长 (3 分钟左右)

```
[root@ master1-admin ~]# rpm -qa | grep kernel-lt
```

```
[root@master1-admin ~]# rpm -qa | grep kernel-lt
```

kernel-lt-5.4.127-1.el7.elrepo.x86_64

kernel-lt-devel-5.4.127-1.el7.elrepo.x86_64

上传 kernel-lt 两个包，升级 rpm -Uvh kernel-lt-*

```
[root@ master1-admin ~]# grub2-set-default "kernel-lt-5.4.127-1"
```

```
[root@ master1-admin ~]# reboot
```

```
[root@ master1-admin ~]# uname -r
```

5.4.127-1.el7.elrepo.x86_64

```
[root@ node1-monitor ~]# yum --enablerepo=elrepo-kernel install kernel-lt-devel kernel-lt -y
```

时间可能会比较长 (3 分钟左右)

```
[root@node1-monitor ~]# rpm -qa | grep kernel-lt
```

```
[root@node1-monitor ~]# rpm -qa | grep kernel-lt
```

kernel-lt-5.4.127-1.el7.elrepo.x86_64

kernel-lt-devel-5.4.127-1.el7.elrepo.x86_64

上传 kernel-lt 两个包, 升级 rpm -Uvh kernel-lt-*

```
[root@node1-monitor ~]# grub2-set-default "kernel-lt-5.4.127-1"
```

```
[root@node1-monitor ~]# reboot
```

```
[root@node1-monitor ~]# uname -r
```

5.4.127-1.el7.elrepo.x86_64

```
[root@node2-osd ~]# yum --enablerepo=elrepo-kernel install kernel-lt-devel kernel-lt -y
```

时间可能会比较长 (3 分钟左右)

```
[root@node2-osd ~]# rpm -qa | grep kernel-lt
```

```
[root@node2-osd ~]# rpm -qa | grep kernel-lt
```

kernel-lt-5.4.127-1.el7.elrepo.x86_64

kernel-lt-devel-5.4.127-1.el7.elrepo.x86_64

上传 kernel-lt 两个包, 升级 rpm -Uvh kernel-lt-*

```
[root@node2-osd ~]# grub2-set-default "kernel-lt-5.4.127-1"
```

```
[root@node2-osd ~]# reboot
```

```
[root@node2-osd ~]# uname -r
```

5.4.127-1.el7.elrepo.x86_64

13.3 安装 ceph 集群

13.3.1 安装 ceph-deploy

#在 master1-admin 节点安装 ceph-deploy

```
[root@master1-admin ~]# yum install python-setuptools ceph-deploy -y
```

#在 master1-admin、node1-monitor 和 node2-osd 节点安装 ceph

```
[root@master1-admin]# yum install ceph ceph-radosgw -y
```

```
[root@node1-monitor ~]# yum install ceph ceph-radosgw -y
```

```
[root@node2-osd ~]# yum install ceph ceph-radosgw -y
```

```
[root@master1-admin ~]# ceph --version
```

ceph version 14.2.16 (762032d6f509d5e7ee7dc008d80fe9c87086603c) nautilus (stable)

13.3.2 创建 monitor 节点

#创建一个目录, 用于保存 ceph-deploy 生成的配置文件信息的

```
[root@master1-admin ceph ~]# cd /etc/ceph
```

```
[root@master1-admin ceph]# ceph-deploy new master1-admin node1-monitor node2-osd
```

```
[root@master1-admin ceph]# ls
```

#生成了如下配置文件

ceph.conf ceph-deploy-ceph.log ceph.mon.keyring

Ceph 配置文件、日志文件、keyring 是在增加 mon 的时候, mon 之间加密会用到的

备注:

ceph-deploy new 后面接的是要初始化成 monitor 的节点

13.3.3 安装 monitor 服务

1、修改 ceph 配置文件

#把 ceph.conf 配置文件里的默认副本数从 3 改成 2。把 osd_pool_default_size = 2 加入[global]段, 这样只有 2 个 osd 也能达到 active+clean 状态:

```
[root@master1-admin]# vim /etc/ceph/ceph.conf
```

```
[global]
```

```
fsid = af5cd413-1c53-4035-90c6-95368eef5c78
```

```
mon_initial_members = node1-monitor
```

```
mon_host = 192.168.40.201
```

```
auth_cluster_required = cephx
```

```
auth_service_required = cephx
```

```
auth_client_required = cephx
```

```
filestore_xattr_use_omap = true
```

```
osd_pool_default_size = 2
```

```
mon clock drift allowed = 0.500
```

```
mon clock drift warn backoff = 10
```

```
mon clock drift allowed #监视器间允许的时钟漂移量默认值 0.05
```

```
mon clock drift warn backoff #时钟偏移警告的退避指数。默认值 5
```

ceph 对每个 mon 之间的时间同步延时默认要求在 0.05s 之间, 这个时间有的时候太短了。如果时钟同步有问题可能导致 monitor 监视器不同步, 可以适当增加时间

2、安装 monitor、收集所有的密钥

```
[root@master1-admin]# cd /etc/ceph
```

```
[root@master1-admin]# ceph-deploy mon create-initial
```

```
[root@master1-admin]# ls *.keyring
```

```
ceph.bootstrap-mds.keyring ceph.bootstrap-mgr.keyring ceph.bootstrap-  
osd.keyring ceph.bootstrap-rgw.keyring ceph.client.admin.keyring  
ceph.mon.keyring
```

13.3.4 部署 osd 服务

```
[root@ master1-admin ceph]# cd /etc/ceph/  
[root@ master1-admin ceph]# ceph-deploy osd create --data /dev/sdb master1-admin  
[root@ master1-admin ceph]# ceph-deploy osd create --data /dev/sdb node1-monitor  
[root@ master1-admin ceph]# ceph-deploy osd create --data /dev/sdb node2-osd
```

查看状态:

```
[root@ master1-admin ceph]# ceph-deploy osd list master1-admin node1-monitor  
node2-osd
```

要使用 Ceph 文件系统, 你的 Ceph 的存储集群里至少需要存在一个 Ceph 的元数据服务器(mds)。

13.3.5 创建 ceph 文件系统

创建 mds

```
[root@ master1-admin ceph]# ceph-deploy mds create master1-admin node1-monitor  
node2-osd
```

查看 ceph 当前文件系统

```
[root@ master1-admin ceph]# ceph fs ls
```

No filesystems enabled

一个 cephfs 至少要求两个 librados 存储池, 一个为 data, 一个为 metadata。当配置这两个存储池时, 注意:

1. 为 metadata pool 设置较高级别的副本级别, 因为 metadata 的损坏可能导致整个文件系统不用
2. 建议, metadata pool 使用低延时存储, 比如 SSD, 因为 metadata 会直接影响客户端的响应速度。

创建存储池

```
[root@ master1-admin ceph]# ceph osd pool create cephfs_data 128  
pool 'cephfs_data' created  
[root@ master1-admin ceph]# ceph osd pool create cephfs_metadata 128  
pool 'cephfs_metadata' created
```

关于创建存储池

命令:

```
ceph osd pool create {pool-name} {pg-num}
```

确定 pg_num 取值是强制性的, 因为不能自动计算。下面是几个常用的值:

*少于 5 个 OSD 时可把 pg_num 设置为 128

- *OSD 数量在 5 到 10 个时, 可把 pg_num 设置为 512
- *OSD 数量在 10 到 50 个时, 可把 pg_num 设置为 4096
- *OSD 数量大于 50 时, 你得理解权衡方法、以及如何自己计算 pg_num 取值
- *自己计算 pg_num 取值时可借助 pgcalc 工具

随着 OSD 数量的增加, 正确的 pg_num 取值变得更加重要, 因为它显著地影响着集群的行为、以及出错时的数据持久性 (即灾难性事件导致数据丢失的概率)。

创建文件系统

创建好存储池后, 你就可以用 fs new 命令创建文件系统了

```
[root@ master1-admin ceph]# ceph fs new xuegod cephfs_metadata cephfs_data
new fs with metadata pool 2 and data pool 1
```

其中: new 后的 fsname 可自定义

```
[root@ master1-admin ceph]# ceph fs ls          #查看创建后的 cephfs
[root@ master1-admin ceph]# ceph mds stat       #查看 mds 节点状态
```

```
xuegod:1 {0=master1-admin=up:active} 2 up:standby
active 是活跃的, 另 1 个是处于热备份的状态
```

13.3.6 部署 mgr

安装 mgr 用于后面我们配置 dashboard 监控, 而且避免挂载 ceph 时可能会提示 warning 信息。

```
[root@ master1-admin ceph]# ceph-deploy mgr create master1-admin node1-monitor
node2-osd
```

安装后查看集群状态正常

```
[root@ master1-admin ceph]# ceph -s
cluster 84469565-5d11-4aeb-88bd-204dc25b2d50
health HEALTH_OK
monmap e1: 1 mons at {node1-monitor=192.168.40.201:6789/0}
election epoch 4, quorum 0 node1-monitor
osdmap e9: 1 osds: 1 up, 1 in
flags sortbitwise,require_jewel_osds
pgmap v107: 320 pgs, 2 pools, 16 bytes data, 3 objects
7371 MB used, 43803 MB / 51175 MB avail
320 active+clean
```

#查看 ceph 守护进程

```
[root@master1-admin ~]# systemctl status ceph-osd.target
[root@master1-admin ~]# systemctl status ceph-mon.target
```

#注意: 如果 ceph -s 看到报错:

```
cluster 84469565-5d11-4aeb-88bd-204dc25b2d50
health HEALTH_WARN
too many PGs per OSD (320 > max 300)
```

互动 1: 报错什么意思?

问题原是集群 osd 数量较少, 测试过程中建立了大量的 pool, 每个 pool 都要用一些 pg_num 和 pgs

解决办法如下:


修改 node1-monitor 节点 ceph 的配置文件 ceph.conf

```
[root@node1-monitor ~]# vim /etc/ceph/ceph.conf
```

最后一行增加如下内容:

```
mon_pg_warn_max_per_osd = 1000
```

```
[global]
fsid = 84469565-5d11-4aeb-88bd-204dc25b2d50
mon_initial_members = node1-monitor
mon_host = 192.168.40.201
auth_cluster_required = cephx
auth_service_required = cephx
auth_client_required = cephx
osd_pool_default_size = 1
mon_pg_warn_max_per_osd = 1000
```



#重启 monitor 进程

```
[root@node1-monitor ~]# systemctl restart ceph-mon.target
```

#在测试 ceph -s

```
[root@master1-admin ~]# ceph -s
```

```
cluster 84469565-5d11-4aeb-88bd-204dc25b2d50
health HEALTH_OK
monmap e1: 1 mons at {node1-monitor=192.168.40.201:6789/0}
election epoch 4, quorum 0 node1-monitor
osdmap e9: 1 osds: 1 up, 1 in
flags sortbitwise,require_jewel_osds
pgmap v107: 320 pgs, 2 pools, 16 bytes data, 3 objects
7371 MB used, 43803 MB / 51175 MB avail
320 active+clean
```

实战 1: 测试 k8s 挂载 ceph rbd

需要有一套 k8s 环境

```
[root@xuegod63 ~]# kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
xuegod63	Ready	control-plane,master	20d	v1.20.4
xuegod64	Ready	<none>	20d	v1.20.4

xuegod63 节点 ip: 192.168.40.63

xuegod64 节点 ip: 192.168.40.64

kubernetes 要想使用 ceph, 需要在 k8s 的每个 node 节点安装 ceph-common, 把 ceph 节

点上的 ceph.repo 文件拷贝到 k8s 各个节点/etc/yum.repos.d/目录下, 然后在 k8s 的各个节点 yum install ceph-common -y

#将 ceph 配置文件拷贝到 k8s 的控制节点和工作节点

```
[root@master1-admin ~]# scp /etc/ceph/* 192.168.40.63:/etc/ceph/
```

```
[root@master1-admin ~]# scp /etc/ceph/* 192.168.40.64:/etc/ceph/
```

#创建 ceph rbd

```
[root@master1-admin ~]# ceph osd pool create k8srbd1 56
```

```
pool 'k8srbd' created
```

```
[root@master1-admin ~]# rbd create rbd1 -s 1024 -p k8srbd1
```

```
[root@master1-admin ~]# rbd feature disable k8srbd1/rbd1 object-map fast-diff  
deep-flatten
```

备注:

rbd create rbd1 -s 1024 -p k8srbd1 #在 k8srbd1 这个 pool 中创建 rbd, 大小是 1024M

rbd 关闭一些特性:

关闭 CentOS7 内核不支持的选项:

object-map: 是否记录组成 image 的数据对象存在状态位图, 通过查表加速类似于导入、导出、克隆分离、已使用容量统计等操作、同时有助于减少 COW 机制带来的克隆 image 的 I/O 时延, 依赖于 exclusive-lock 特性

fast-diff: 用于计算快照间增量数据等操作加速, 依赖于 object-map

deep-flatten: 克隆分离时是否同时解除克隆 image 创建的快照与父 image 之间的关联

#创建 pod, 挂载 ceph rbd

#把 nginx.tar.gz 上传到 xuegod64 上, 手动解压

```
[root@xuegod64 ~]# docker load -i nginx.tar.gz
```

```
[root@xuegod63 ~]# vim pod.yaml
```

apiVersion: v1

kind: Pod

metadata:

name: testrbd

spec:

containers:

- image: nginx

name: nginx

imagePullPolicy: IfNotPresent

volumeMounts:

- name: testrbd

mountPath: /mnt

volumes:

- name: testrbd

```
rbd:
  monitors:
  - '192.168.40.201:6789'
  pool: k8srbd1
  image: rbda
  fsType: xfs
  readOnly: false
  user: admin
  keyring: /etc/ceph/ceph.client.admin.keyring
```

#更新资源清单文件

```
[root@xuegod63 ~]# kubectl apply -f pod.yaml
```

#查看 pod 是否创建成功

```
[root@xuegod63 ~]# kubectl get pods -o wide
```

```
[root@xuegod63 ~]# kubectl get pods -o wide
NAME      READY   STATUS    RESTARTS   AGE   IP            NODE      NOMINATED NODE   READINESS GATES
testrbd   1/1     Running   0           62s   10.244.209.138 xuegod64   <none>           <none>
```

注意: k8srbd1 下的 rbda 被 pod 挂载了, 那其他 pod 就不能占用这个 k8srbd1 下的 rbda 了

例: 创建一个 pod-1.yaml

```
[root@xuegod63 ~]# cat pod-1.yaml
```

apiVersion: v1

kind: Pod

metadata:

name: testrbd1

spec:

containers:

- image: nginx

name: nginx

volumeMounts:

- name: testrbd

mountPath: /mnt

volumes:

- name: testrbd

rbd:

monitors:

- '192.168.40.201:6789'

pool: k8srbd1

image: rbda

fsType: xfs

readOnly: false

user: admin

keyring: /etc/ceph/ceph.client.admin.keyring

```
[root@xuegod63 ~]# kubectl apply -f pod-1.yaml
```

```
pod/testrbd1 created
```

```
[root@xuegod63 ~]# kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
testrbd	1/1	Running	0	51s
testrbd1	0/1	Pending	0	15s

#查看 testrbd1 详细信息

```
[root@xuegod63 ~]# kubectl describe pods testrbd1
```

Warning FailedScheduling 48s (x2 over 48s) default-scheduler 0/2 nodes are available: 1 node(s) had no available disk, 1 node(s) had taint {node-role.kubernetes.io/master: }, that the pod didn't tolerate.

上面一直 pending 状态, 通过 warnning 可以发现是因为我的 pool: k8srbd1

image: rbda

已经被其他 pod 占用了。

实战 2: 基于 ceph rbd 生成 pv

1、创建 ceph-secret 这个 k8s secret 对象, 这个 secret 对象用于 k8s volume 插件访问 ceph 集群, 获取 client.admin 的 keyring 值, 并用 base64 编码, 在 master1-admin (ceph 管理节点) 操作

```
[root@master1-admin ~]# ceph auth get-key client.admin | base64
```

```
QVFBWk0zeGdZdDIhQXhBQVZsS0poYzIQUIBianBGSWJVbDNBenc9PQ==
```

2.创建 ceph 的 secret, 在 k8s 的控制节点操作:

```
[root@xuegod63 ~]# cat ceph-secret.yaml
```

```
apiVersion: v1
```

```
kind: Secret
```

```
metadata:
```

```
  name: ceph-secret
```

```
data:
```

```
  key: QVFBWk0zeGdZdDIhQXhBQVZsS0poYzIQUIBianBGSWJVbDNBenc9PQ==
```

```
[root@xuegod63 ~]# kubectl apply -f ceph-secret.yaml
```

3.回到 ceph 管理节点创建 pool 池

```
[root@master1-admin ~]# ceph osd pool create k8stest 56
```

```
pool 'k8stest' created
```

```
[root@master1-admin ~]# rbd create rbda -s 1024 -p k8stest
```

```
[root@master1-admin ~]# rbd feature disable k8stest/rbda object-map fast-diff deep-flatten
```

4、创建 pv

```
[root@xuegod63 ~]# cat pv.yaml
```

```
apiVersion: v1
```

```
kind: PersistentVolume
metadata:
  name: ceph-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  rbd:
    monitors:
      - 192.168.40.201:6789
    pool: k8stest
    image: rbd
    user: admin
    secretRef:
      name: ceph-secret
    fsType: xfs
    readOnly: false
  persistentVolumeReclaimPolicy: Recycle
```

```
[root@xuegod63 ~]# kubectl apply -f pv.yaml
```

```
persistentvolume/ceph-pv created
```

```
[root@xuegod63 ~]# kubectl get pv
```

```
[root@xuegod63 ~]# kubectl get pv
NAME          CAPACITY  ACCESS MODES  RECLAIM POLICY  STATUS   CLAIM  STORAGECLASS  REASON  AGE
ceph-pv       1Gi       RWO           Recycle         Available             12s
```

5、创建 pvc

```
[root@xuegod63 ~]# cat pvc.yaml
```

```
kind: PersistentVolumeClaim
```

```
apiVersion: v1
```

```
metadata:
```

```
  name: ceph-pvc
```

```
spec:
```

```
  accessModes:
```

```
    - ReadWriteOnce
```

```
  resources:
```

```
    requests:
```

```
      storage: 1Gi
```

```
[root@xuegod63 ~]# kubectl apply -f pvc.yaml
```

```
[root@xuegod63 ceph]# kubectl get pvc
```

```
NAME          STATUS  VOLUME  CAPACITY  ACCESS MODES  STORAGECLASS  AGE
ceph-pvc      Bound   ceph-pv  1Gi       RWO           6s
```


6、测试挂载 pvc

```
[root@xuegod63 ~]# cat pod-2.yaml
```

```
apiVersion: apps/v1
```

```
kind: Deployment
```

```
metadata:
```

```
  name: nginx-deployment
```

```
spec:
```

```
  selector:
```

```
    matchLabels:
```

```
      app: nginx
```

```
  replicas: 2 # tells deployment to run 2 pods matching the template
```

```
  template: # create pods using pod definition in this template
```

```
    metadata:
```

```
      labels:
```

```
        app: nginx
```

```
    spec:
```

```
      containers:
```

```
        - name: nginx
```

```
          image: nginx
```

```
          imagePullPolicy: IfNotPresent
```

```
      ports:
```

```
        - containerPort: 80
```

```
      volumeMounts:
```

```
        - mountPath: "/ceph-data"
```

```
          name: ceph-data
```

```
      volumes:
```

```
        - name: ceph-data
```

```
          persistentVolumeClaim:
```

```
            claimName: ceph-pvc
```

```
[root@xuegod63 ~]# kubectl apply -f pod-2.yaml
```

```
[root@xuegod63 ~]# kubectl get pods -l app=nginx
```

NAME	READY	STATUS	RESTARTS	AGE
nginx-deployment-fc894c564-8tfhn	1/1	Running	0	78s
nginx-deployment-fc894c564-l74km	1/1	Running	0	78s

```
[root@xuegod63 ~]# cat pod-3.yaml
```

```
apiVersion: apps/v1
```

```
kind: Deployment
```

```
metadata:
```

```
  name: nginx-1-deployment
```

```
spec:
```

```
selector:
  matchLabels:
    appv1: nginxv1
replicas: 2 # tells deployment to run 2 pods matching the template
template: # create pods using pod definition in this template
  metadata:
    labels:
      appv1: nginxv1
  spec:
    containers:
      - name: nginx
        image: nginx
        imagePullPolicy: IfNotPresent
        ports:
          - containerPort: 80
        volumeMounts:
          - mountPath: "/ceph-data"
            name: ceph-data
    volumes:
      - name: ceph-data
        persistentVolumeClaim:
          claimName: ceph-pvc
```

```
[root@xuegod63 ~]# kubectl apply -f pod-3.yaml
```

```
[root@xuegod63 ~]# kubectl get pods -l appv1=nginxv1
```

NAME	READY	STATUS	RESTARTS	AGE
nginx-1-deployment-cd74b5dd4-jqvxi	1/1	Running	0	56s
nginx-1-deployment-cd74b5dd4-lrddc	1/1	Running	0	56s

通过上面实验可以发现 pod 是可以 ReadWriteOnce 共享挂载相同的 pvc 的

注意:

ceph rbd 块存储的特点

ceph rbd 块存储能在同一个 node 上跨 pod 以 ReadWriteOnce 共享挂载

ceph rbd 块存储能在同一个 node 上同一个 pod 多个容器中以 ReadWriteOnce 共享挂载

ceph rbd 块存储不能跨 node 以 ReadWriteOnce 共享挂载

如果一个使用 ceph rbd 的 pod 所在的 node 挂掉, 这个 pod 虽然会被调度到其它 node, 但是由于 rbd 不能跨 node 多次挂载和挂掉的 pod 不能自动解绑 pv 的问题, 这个新 pod 不会正常运行

Deployment 更新特性:

deployment 触发更新的时候, 它确保至少所需 Pods 75% 处于运行状态 (最大不可用比例为 25%)。故像一个 pod 的情况, 肯定是新创建一个新的 pod, 新 pod 运行正常之后, 再关闭老的 pod。

默认情况下, 它可确保启动的 Pod 个数比期望个数最多多出 25%

问题:

结合 ceph rb 共享挂载的特性和 deployment 更新的特性, 我们发现原因如下:

由于 deployment 触发更新, 为了保证服务的可用性, deployment 要先创建一个 pod 并运行正常之后, 再去删除老 pod。而如果新创建的 pod 和老 pod 不在一个 node, 就会导致此故障。

解决办法:

- 1, 使用能支持跨 node 和 pod 之间挂载的共享存储, 例如 cephfs, GlusterFS 等
- 2, 给 node 添加 label, 只允许 deployment 所管理的 pod 调度到一个固定的 node 上。(不建议, 这个 node 挂掉的话, 服务就故障了)

实战 3: 基于 storageclass 动态生成 pv

1、创建 rbd 的供应商 provisioner

#把 rbd-provisioner.tar.gz 上传到 xuegod64 上, 手动解压

```
[root@xuegod64 ~]# docker load -i rbd-provisioner.tar.gz
```

```
[root@xuegod63 ~]# cat rbd-provisioner.yaml
```

kind: ClusterRole

apiVersion: rbac.authorization.k8s.io/v1

metadata:

name: rbd-provisioner

rules:

- apiGroups: [""]
resources: ["persistentvolumes"]
verbs: ["get", "list", "watch", "create", "delete"]
- apiGroups: [""]
resources: ["persistentvolumeclaims"]
verbs: ["get", "list", "watch", "update"]
- apiGroups: ["storage.k8s.io"]
resources: ["storageclasses"]
verbs: ["get", "list", "watch"]
- apiGroups: [""]
resources: ["events"]
verbs: ["create", "update", "patch"]
- apiGroups: [""]
resources: ["services"]
resourceNames: ["kube-dns", "coredns"]
verbs: ["list", "get"]

kind: ClusterRoleBinding

apiVersion: rbac.authorization.k8s.io/v1

metadata:

name: rbd-provisioner

subjects:

```
- kind: ServiceAccount
  name: rbd-provisioner
  namespace: default
roleRef:
  kind: ClusterRole
  name: rbd-provisioner
  apiGroup: rbac.authorization.k8s.io
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: rbd-provisioner
rules:
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["get"]
- apiGroups: [""]
  resources: ["endpoints"]
  verbs: ["get", "list", "watch", "create", "update", "patch"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: rbd-provisioner
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: rbd-provisioner
subjects:
- kind: ServiceAccount
  name: rbd-provisioner
  namespace: default
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: rbd-provisioner
spec:
  selector:
    matchLabels:
      app: rbd-provisioner
  replicas: 1
  strategy:
    type: Recreate
```

```
template:
  metadata:
    labels:
      app: rbd-provisioner
  spec:
    containers:
      - name: rbd-provisioner
        image: quay.io/xuegod/external_storage/rbd-provisioner:v1
        env:
          - name: PROVISIONER_NAME
            value: ceph.com/rbd
        serviceAccount: rbd-provisioner
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: rbd-provisioner

[root@xuegod63 ~]# kubectl apply -f rbd-provisioner.yaml
[root@xuegod63 ~]# kubectl get pods -l app=rbd-provisioner
```

NAME	READY	STATUS	RESTARTS	AGE
rbd-provisioner-685746688f-8mbz5	1/1	Running	0	111s

2、创建 ceph-secret

#创建 pool 池

```
[root@master1-admin ~]# ceph osd pool create k8stest1 56
```

```
[root@xuegod63 ~]# cat ceph-secret-1.yaml
```

```
apiVersion: v1
kind: Secret
metadata:
  name: ceph-secret-1
type: "ceph.com/rbd"
data:
  key: QVFBWk0zeGdZdDIhQXhBQVZsS0poYzlQUIBianBGSWJVbDNBenc9PQ==
```

```
[root@xuegod63 ~]# kubectl apply -f ceph-secret-1.yaml
```

3、创建 storageclass

```
[root@xuegod63 ~]# cat storageclass.yaml
```

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: k8s-rbd
provisioner: ceph.com/rbd
```

parameters:

```
monitors: 192.168.40.201:6789
adminId: admin
adminSecretName: ceph-secret-1
pool: k8stest1
userId: admin
userSecretName: ceph-secret-1
fsType: xfs
imageFormat: "2"
imageFeatures: "layering"
```

```
[root@xuegod63 ~]# kubectl apply -f storageclass.yaml
```

注意:

k8s1.20 版本通过 rbd provisioner 动态生成 pv 会报错:

```
[root@xuegod63 ~]# kubectl logs rbd-provisioner-685746688f-8mbz
```

```
E0418 15:50:09.610071      1 controller.go:1004] provision "default/rbd-pvc" class
"k8s-rbd": unexpected error getting claim reference: selfLink was empty, can't make
reference, 报错原因是 1.20 版本仅用了 selfLink, 解决方法如下;
```

编辑/etc/kubernetes/manifests/kube-apiserver.yaml

在这里:

spec:

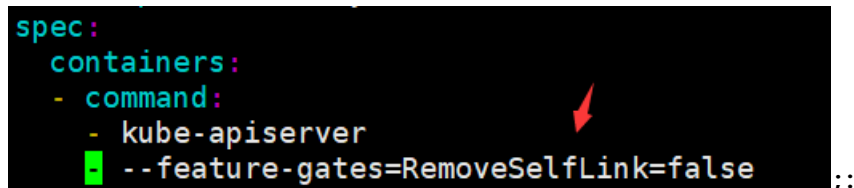
containers:

- command:

- kube-apiserver

添加这一行:

- --feature-gates=RemoveSelfLink=false



```
spec:
  containers:
  - command:
    - kube-apiserver
    - --feature-gates=RemoveSelfLink=false
```

然后应用它, 即可:

```
kubectl apply -f /etc/kubernetes/manifests/kube-apiserver.yaml
```

```
[root@xuegod63 ~]# kubectl get pods -n kube-system
```

NAME	READY	STATUS	RESTARTS
AGE			
kube-apiserver	0/1	CrashLoopBackOff	1
kube-apiserver-xuegod63	1/1	Running	0

kube-apiserver 状态是 crashloopbackoff, 这个 pod 可以删除, 因为 kube-apiserver-xuegod63 是提供服务的 pod

4、创建 pvc

```
[root@xuegod63 ~]# cat rbd-pvc.yaml
```



```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: rbd-pvc
spec:
  accessModes:
    - ReadWriteOnce
  volumeMode: Filesystem
  resources:
    requests:
      storage: 1Gi
  storageClassName: k8s-rbd
[root@xuegod63 ~]# kubectl apply -f rbd-pvc.yaml
[root@xuegod63 ~]# kubectl get pvc
```

NAME	STATUS	VOLUME	CAPACITY	ACCESS MODES	STORAGECLASS	AGE
ceph-pvc	Bound	ceph-pv	1Gi	RWO		38m
rbd-pvc	Bound	pvc-6e55efcf-3cbf-4bf0-a66a-f8d1e1dfa8c0	1Gi	RWO	k8s-rbd	6m23s

```
创建 pod, 挂载 pvc
[root@xuegod63 ~]# cat pod-sto.yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: rbd-pod
  name: ceph-rbd-pod
spec:
  containers:
    - name: ceph-rbd-nginx
      image: nginx
      imagePullPolicy: IfNotPresent
      volumeMounts:
        - name: ceph-rbd
          mountPath: /mnt
          readOnly: false
  volumes:
    - name: ceph-rbd
      persistentVolumeClaim:
        claimName: rbd-pvc
[root@xuegod63 ~]# kubectl apply -f pod-sto.yaml
[root@xuegod63 ~]# kubectl get pods -l test=rbd-pod
NAME          READY   STATUS    RESTARTS   AGE
ceph-rbd-pod  1/1     Running   0           50s
```

实战 4: k8s 挂载 cephfs

```
[root@master1-admin ~]# ceph fs ls
name: xuegod, metadata pool: cephfs_metadata, data pools: [cephfs_data]
```

1、创建 ceph 子目录

为了别的地方能挂载 cephfs, 先创建一个 secretfile

```
[root@master1-admin ~]# cat /etc/ceph/ceph.client.admin.keyring |grep key|awk -F" "
'{print $3}' > /etc/ceph/admin.secret
```

挂载 cephfs 的根目录到集群的 mon 节点下的一个目录, 比如 xuegod_data, 因为挂载后, 我们就可以直接在 xuegod_data 下面用 Linux 命令创建子目录了。

```
[root@master1-admin ~]# mkdir xuegod_data
[root@master1-admin ~]# mount -t ceph 192.168.40.201:6789:/ /root/xuegod_data -o
name=admin,secretfile=/etc/ceph/admin.secret
```

```
[root@master1-admin ~]# df -h
192.168.40.201:6789:/    165G  106M  165G   1% /root/xuegod_data
```

在 cephfs 的根目录里面创建了一个子目录 lucky, k8s 以后就可以挂载这个目录

```
[root@master1-admin ~]# cd /root/xuegod_data/
[root@master1-admin xuegod_data]# mkdir lucky
[root@master1-admin xuegod_data]# chmod 0777 lucky/
```

2、测试 k8s 的 pod 挂载 cephfs

1)创建 k8s 连接 ceph 使用的 secret

将/etc/ceph/ceph.client.admin.keyring 里面的 key 的值转换为 base64, 否则会有问题

```
[root@master1-admin xuegod_data]# echo
"AQBvBdZgsDSZKxAAan+5rnsjr2ziA/atqFnQOA==" | base64
QVFCdkJkWmdzRFNaS3hBQWFuKzVybnNqcjJ6aUEvYXRxRm5RT0E9PQo=
```

```
[root@xuegod63 ceph]# cat cephfs-secret.yaml
apiVersion: v1
kind: Secret
metadata:
  name: cephfs-secret
data:
  key: QVFCdkJkWmdzRFNaS3hBQWFuKzVybnNqcjJ6aUEvYXRxRm5RT0E9PQo=
```

```
[root@ xuegod63 ceph]# kubectl apply -f cephfs-secret.yaml
[root@ xuegod63 ceph]# cat cephfs-pv.yaml
```

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: cephfs-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteMany
  cephfs:
    monitors:
      - 192.168.40.201:6789
    path: /lucky
    user: admin
    readOnly: false
    secretRef:
      name: cephfs-secret
  persistentVolumeReclaimPolicy: Recycle
[root@ xuegod63 ceph]# kubectl apply -f cephfs-pv.yaml
```

```
[root@ xuegod63 ceph]# cat cephfs-pvc.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: cephfs-pvc
spec:
  accessModes:
    - ReadWriteMany
  volumeName: cephfs-pv
resources:
  requests:
    storage: 1Gi
```

```
[root@ xuegod63 ceph]# kubectl apply -f cephfs-pvc.yaml
```

```
[root@ xuegod63 ceph]# kubectl get pvc
```

NAME	STATUS	VOLUME	CAPACITY
cephfs-pvc	Bound	cephfs-pv	1Gi RWX

创建第一个 pod, 挂载 cephfs-pvc

```
[root@ xuegod63 ceph]# cat cephfs-pod-1.yaml
apiVersion: v1
```

```
kind: Pod
metadata:
  name: cephfs-pod-1
spec:
  containers:
    - image: nginx
      name: nginx
      imagePullPolicy: IfNotPresent
      volumeMounts:
        - name: test-v1
          mountPath: /mnt
  volumes:
    - name: test-v1
      persistentVolumeClaim:
        claimName: cephfs-pvc
```

```
[root@xuegodmaster1 ceph]# kubectl apply -f cephfs-pod-1.yaml
```

创建第二个 pod, 挂载 cephfs-pvc

```
[root@ xuegod63 ceph]# cat cephfs-pod-2.yaml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: cephfs-pod-2
spec:
  containers:
    - image: nginx
      name: nginx
      imagePullPolicy: IfNotPresent
      volumeMounts:
        - name: test-v1
          mountPath: /mnt
  volumes:
    - name: test-v1
      persistentVolumeClaim:
        claimName: cephfs-pvc
```

```
[root@ xuegod63 ceph]# kubectl apply -f cephfs-pod-2.yaml
```

```
[root@ xuegod63 ceph]# kubectl exec -it cephfs-pod-1 -- /bin/sh
```

```
# cd /mnt
```

```
# touch hello
```

```
[root@ xuegod63 ceph]# kubectl exec -it cephfs-pod-2 -- /bin/sh
```

```
# cd /mnt
```

```
# touch welcome
```

回到 master1-admin 上, 可以看到在 cephfs 文件目录下已经存在内容了

```
[root@master1-admin lucky]# pwd
```

```
/root/xuegod_data/lucky
```

```
[root@master1-admin lucky]# ls
```

```
hello  welcome
```

总结:

13.1 初始化安装 ceph 的实验环境

13.2 安装 ceph 集群

13.3 激活 ceph osd

实战 1: 测试 k8s 挂载 ceph rbd

实战 2: 基于 ceph rbd 生成 pv

实战 3: 基于 storageclass 动态生成 pv

实战 4: k8s 挂载 cephf