参考资料：

MinIO 简介：

<https://www.jianshu.com/p/c2b43ff67df0>

<https://www.jianshu.com/p/f39e7255805b>

单机安装minio

docker run -d -p 9800:9800 \

-p 9801:9801 \

--name minio-service \

--restart=always \

-e "MINIO\_ROOT\_USER=root" \

-e "MINIO\_ROOT\_PASSWORD=top@958958" \

-v /opt/minio/data:/data \

-v /etc/minio/config:/root/.minio \

minio/minio server /data \

--console-address ":9801"

# 1.分布式文件系统应用场景

互联网海量**非结构化数据**的存储需求

电商网站：海量商品图片

视频网站：海量视频文件

网盘：海量文件

社交网站：海量图片

# 2.什么是Minio

**MinIO是对象存储服务(Object Storage Service,OSS)**，**非常适合存储大容量非结构化的数据。例如图片、视频、日志文件等，对象文件可以是任意大小，从几KB到最大5T不等。**

官网：<https://min.io/>

# 3.MinIO的基本概念

Object:存储到MiniIO的基本对象（文件、字节流、anything）

Bucket:用来存储Object的逻辑空间。每个Bucket之间的数据是相互隔离的。对于客户端而言，就相当于一个存放文件的顶级文件夹。

Drive：即存储数据的磁盘，在MinIO启动的时候，以参数的方式传入。MinIO中所有的对象数据都会存储在Drive里。，

Set：即一组Drive的集合。分布式部署根据集群规模自动划分一个或多个 ，每个Set中的Drive分布在不同的位置。一个对象存储在一个Set上。

一个对象存储在一个Set上。

一个集群划分多个Set 。。

一个Set包含的Drive数量是固定的，默认由系统根据集群的规模自动计算出。

一个Set中的Drive尽可能分布在不同的节点上。

# 3.MinIO优点

1.MinIO支持海量存储，可按zone拓展（原zone不受任何影响），支持单个对象最大5TB。

2.低冗余高磁盘损坏容忍，标准且最高的数据冗余系数为2（即存储一个1M的数据对象，实际占用磁盘空间为2M）,但在任意的n/2块disk损坏的情况下依然可以读出数据（n为一个纠错码集合（Erasure Coding Set）中的disk数量）。并且这种损坏恢复是基于单个对象的，而不是整个存储卷的。

3.读写性能优异。

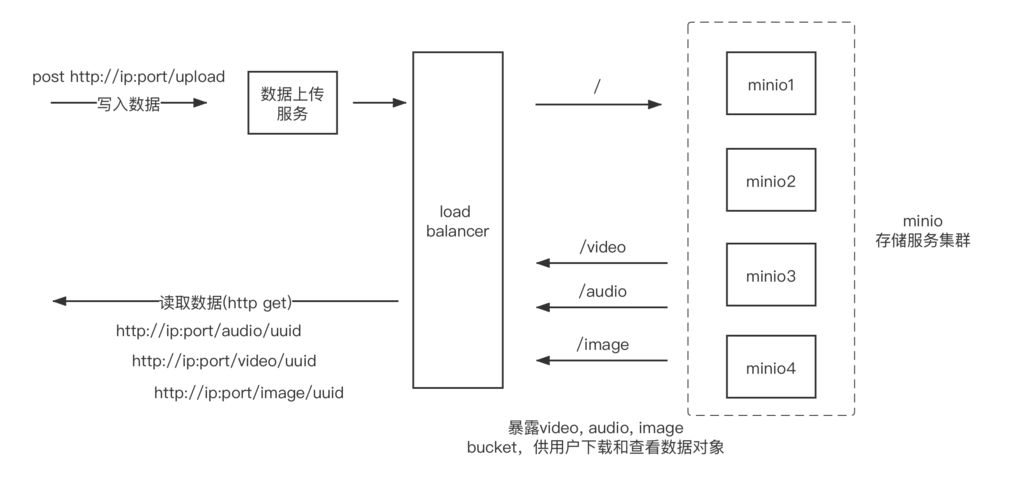
# 4.纠删码EC（Erasure Code）

MinIO使用纠删码机制来保证高可靠性。使用highwayhash来处理数据损坏（Bit Rot Protection）。关于纠错码，简单来说就是可以通过数学计算，把丢失的数据进行还原，增加m份数据，并通过n+m份中的任意n份数据，还原为原始数据。即如果任意小于等于m份的数据失效，仍然能通过剩下的数据还原出来。

# 5.存储形式

文件对象上传到MinIO，会在对应的存储磁盘中，以Bucket名称为目录，文件名称为下一级目录，文件名是part.1和xl.meta，前者是编码数据块及检验块，后者是元数据文件。

# 6.存储方案



# 7.MinIO环境搭建（基于Docker）

官方文档：<https://docs.min.io/docs/>

中文文档：<http://docs.minio.org.cn/docs/>

官方文档最新,建议使用官方文档;

## 7.1单机部署

MinIO server的standalone模式，即要管理的磁盘都在host本地。改启动模式一般仅用在试验环境、测试环境的验证和学习使用，在standalone模式下可以分为non-erasure code mode 和erasure code mode.

non-erasure code mode:在此模式下，对于每一份数据对象，MinIO直接在data下面存储这份数据，不会建立副本，也不会启动纠删码机制。因此，这种模式无论是服务实例还是磁盘都是“单点”，无任何高可用保障，磁盘损坏就表示数据丢失。

erasure code mode：此模式为MinIO server实例传入多个本地磁盘参数，一旦遇到多个磁盘参数，MinIO server 会自动启动erasure code mode。erasure code mode对磁盘的个数是有要求的，如果不满足要求，实例启动将失败。erasure code 启用后，要求传给MinIO server 的endpoint（standalone模式下，即本地磁盘上的目录），至少4个。

### 基于Docker

单机模式没有纠删码保存的是源文件。

docker run -d\

-p 9001:9001 \

-p 9002:9002 \

-e "MINIO\_ROOT\_USER=AKIAIOSFODNN7EXAMPLE" \

-e "MINIO\_ROOT\_PASSWORD=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY" \

-v /opt/minio/data:/data \

-v /etc/minio/config:/root/.minio \

quay.io/minio/minio server /data --console-address ":9001"

### MinIO纠删码模式

MinIO使用纠删码模式erasure code、校验和checksum来保护数据免受硬件故障和无声数据损坏。即是丢失一半数量（n/2）的硬盘，仍然可以恢复数据。

纠删码是一种恢复丢失和损坏数据的数学算法，MinIO采用Reed-Solomon code将对象拆分成N/2数据和N/2奇偶校验块。这就一意味着如果有12块硬盘，一个对象会被分为6个数据块、6个奇偶校验块，你可以丢失任意的6块盘（不管其是存放的数据块还是奇偶校验块），仍然可以从剩下的盘中的数据进行恢复。

使用MinIO Docker镜像，在8块盘中启动MinIO服务

docker run -d \

-p 9000:9000 \

-p 9001:9001 \

--name minio \

-v /mnt/data1 :/data1\

-v /mnt/data2 :/data2\

-v /mnt/data3 :/data3\

-v /mnt/data4 :/data4\

-v /mnt/data5 :/data5\

-v /mnt/data6 :/data6\

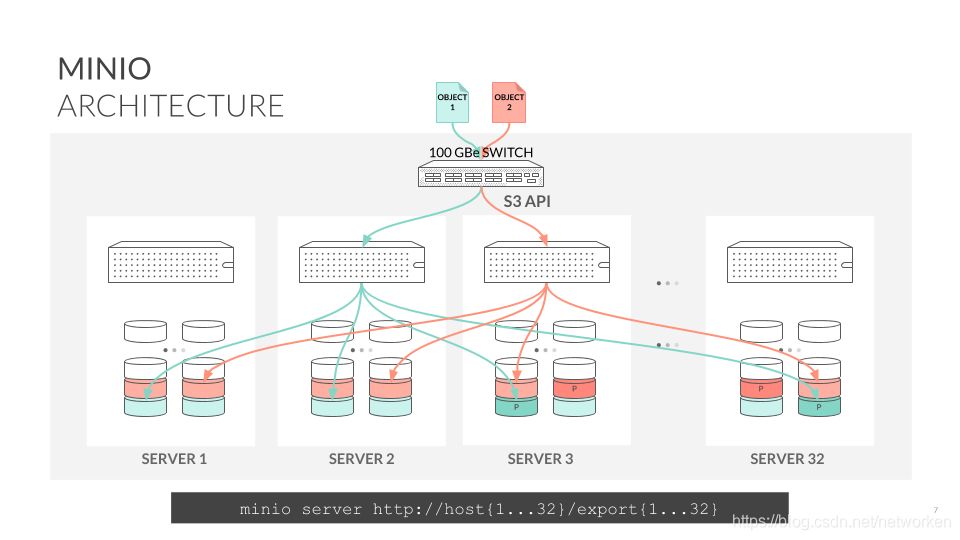
-v /mnt/data7 :/data7\

-v /mnt/data8 :/data8\

minio/minio server /data{1...8} \

--console-address ":9801"

## 7.2分布式集群部署



### 分布式存储可靠性常用方法

分布式存储，很关键的点在于数据的可靠性，即保证数据的完整，不丢失，不损坏。只有在可靠性实现的前提下，才有了追求一致性、高可用、高性能的基础。而对应存储领域，一般对数据可靠性的方法有两种，一类是冗余法，一类是校验法。

**冗余**

冗余发最简单直接，及对存储的数据进行副本备份，当数据出现丢失，损坏，即可以使用备份内容进行恢复，而副本的备份的多少，决定了数据可靠性的高低。这其中有成本的考量，副本数量越多，数据越可靠，单需要的设备就越多，成本就越高。可靠性是允许丢失其中一份数据，当前已有很多分布式系统采用此种方式实现。如Hadoop的文件系统（3个副本），Redis集群，MYSQL的主备模式等。

**校验**

校验法即通过校验码的数学计算方式，对出现丢失、损坏的数据进行校验，还原。注意，这里有两个作用，一个校验，通过对数据进行校验和（checksum）进行计算，可以检查数据是否完整，有无损坏或更改，在数据传输和保存中常用到，如TCP协议；二是恢复还原，通过对数据结合校验码，通过数学计算，还原丢失或损坏的数据。可以在保证数据可靠性的前提下，降低冗余，如单机硬盘存储中的RAID技术。纠删码（Erasure code）技术等。

MinIO采用的就是纠删码技术。

### 分布式MinIO的优势

**数据保护**

分布式的MinIO采用纠删码来防范多个节点宕机和位衰减bit rot。

分布式的MinIO至少需要4个硬盘，使用分布式的MinIO自动引入了纠删码功能。

**高可用**

单机MinIO服务存在单点故障，相反，如果是一个有N块硬盘的分布式MinIO，只要有N/2硬盘在线，数据就是安全的，不过需要至少有N/2+1块硬盘来创建新的对象。

例如，一个16节点的Minio集群，每个节点16块硬盘，就算8台服务器宕机，这个集群仍然是可读的，不过需要9台服务器才能写数据。

**一致性**

MinIO在分布式和单机模式下，所有读写操作都是严格遵守read-after-write一致性模型。

### 运行分布式MinIO

启动一个分布式MinIO实例，只需要把硬盘位置作为参数传个minio server 命令即可。然后在所有其他节点运行同样的命令。

分布式MinIO里所有的节点需要有同样的access秘钥和secret秘钥，这样这些节点才能建立联接。为了实现这个，需要执行minioserver 命令之前，香江access秘钥和secret秘钥export成环境变量，新版使用MINIO\_ROOT\_USER和MINIO\_ROOT\_PASSWORD。

分布式MinIO使用的磁盘必须是干净的，里面没有数据。

**分布式MinIO里的节点时间差不能超过3秒，可以使用NTP来保证时间一致**。

### 8个节点，每个节点1块盘

启动分布式MinIO实例，8个节点，每个节点1块盘，需要在8个节点上运行命令：

export MINIO\_ROOT\_USER=root

export MINIO\_ROOT\_PASSWORD=top@958958

minio server <http://192.168.1.11/export1> http://192.168.1.12/export2 \

<http://192.168.1.13/export3> http://192.168.1.14/export4 \

<http://192.168.1.15/export5> http://192.168.1.16/export6 \

<http://192.168.1.17/export7> <http://192.168.1.18/export8>

### 4个节点(主机)，每节点4个盘

启动分布式MinIO实例，4个节点，每节点4个盘，需要在4个节点上运行命令：

export MINIO\_ROOT\_USER=root

export MINIO\_ROOT\_PASSWORD=top@958958

minio server <http://192.168.1.11/export1> http://192.168.1.11/export2 \

<http://192.168.1.11/export3> \ [http://192.168.1.11/export4 \](http://192.168.1.11/export4%20\)

<http://192.168.1.12/export1> http://192.168.1.12/export2 \

<http://192.168.1.12/export3> \ [http://192.168.1.12/export4 \](http://192.168.1.12/export4%20\)

<http://192.168.1.13/export1>

http://192.168.1.13/export2 \

<http://192.168.1.13/export3>

[http://192.168.1.13/export4 \](http://192.168.1.13/export4%20\)

<http://192.168.1.14/export1> \

http://192.168.1.14/export2 \

http://192.168.1.14/export3 \

<http://192.168.1.14/export4>

### 基于Docker部署MinIO集群

配置：

docker run -d --net=host\

--name minio \

--restart=always\

-e "MINIO\_ROOT\_USER=admin" \

-e "MINIO\_ROOT\_PASSWORD=minio@958958" \

-v /opt/minio/data1:/data1 \

-v /opt/minio/data2:/data2 \

-v /opt/minio/data3:/data3 \

-v /opt/minio/data4:/data4 \

minio/minio server \

--address 192.168.1.88:9001 \

http://192.168.1.88/opt/minio/data{1...4} \

http://192.168.1.18/opt/minio/data{1...4} \

http://192.168.1.68/opt/minio/data{1...4}\

http://1.116.226.147/opt/minio/data{1...4}

docker run -d --net=host\

--name minio \

--restart=always\

-e "MINIO\_ROOT\_USER=admin" \

-e "MINIO\_ROOT\_PASSWORD=minio@958958" \

-v /opt/minio/data1:/data1 \

-v /opt/minio/data2:/data2 \

-v /opt/minio/data3:/data3 \

-v /opt/minio/data4:/data4 \

minio/minio server \

--address 192.168.1.18:9001 \

http://192.168.1.88/opt/minio/data{1...4} \

http://192.168.1.18/opt/minio/data{1...4}\

http://192.168.1.68/opt/minio/data{1...4}\

http://1.116.226.147/opt/minio/data{1...4}

docker run -d --net=host\

--name minio \

--restart=always\

-e "MINIO\_ROOT\_USER=admin" \

-e "MINIO\_ROOT\_PASSWORD=minio@958958" \

-v /opt/minio/data1:/data1 \

-v /opt/minio/data2:/data2 \

-v /opt/minio/data3:/data3 \

-v /opt/minio/data4:/data4 \

minio/minio server \

--address 192.168.1.68:9001 \

http://192.168.1.88/opt/minio/data{1...4}\

http://192.168.1.18/opt/minio/data{1...4}\

http://192.168.1.68/opt/minio/data{1...4}\

http://1.116.226.147/opt/minio/data{1...4}

docker run -d --net=host\

--name minio \

--restart=always\

-e "MINIO\_ROOT\_USER=admin" \

-e "MINIO\_ROOT\_PASSWORD=minio@958958" \

-v /opt/minio/data1:/data1 \

-v /opt/minio/data2:/data2 \

-v /opt/minio/data3:/data3 \

-v /opt/minio/data4:/data4 \

minio/minio server \

--address 1.116.226.147:9001 \

http://192.168.1.88/opt/minio/data{1...4}

http://192.168.1.18/opt/minio/data{1...4}

http://192.168.1.68/opt/minio/data{1...4}

http://1.116.226.147/opt/minio/data{1...4}

# 相关资料：

<https://www.cnblogs.com/sunnydou/p/15130684.html>

<https://www.icode9.com/content-4-830528.html>

基于Docker Compose部署分布式MinIO集群

<https://www.cnblogs.com/xiao987334176/p/13256553.html>

# 问题描述：

因为挂载错了，想取消挂载，但是umount报告如下错误：

[root@zabbix /]# umount /dev/sdc1

umount: /data1: target is busy.

(In some cases useful info about processes that use

the device is found by lsof(8) or fuser(1))

根据描述应该其他进程对挂载的目录有占用的情况

## 解决方案：

关闭占用的进程，但是在使用fuser命令的时候显示没有这个命令

[root@zabbix /]# fuser -m /dev/sdc1

-bash: fuser: command not found

然后需要安装一下：

[root@zabbix /]# yum install -y psmisc

再去使用fuser命令查看进程：

[root@zabbix /]# fuser -m /dev/sdc1

/dev/sdc1: 10246c

[root@zabbix /]# ps aux | grep 10246

root 10246 0.0 0.0 116460 3212 pts/0 Ss+ Oct19 0:00 -bash

使用kill命令关闭进程：

[root@zabbix /]# kill -9 10246

关闭进程以后再umount就成功了：

[root@zabbix /]# umount /dev/sdc1

最后可以使用df -h查看挂载情况，会发现已经取消挂载成功！