

分布式文件存储系统Minio实战

主讲老师: Fox

本课程minio 版本: RELEASE.2021-07-*及以上

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

1.1 Minio介绍

MinIO 是一个基于Apache License v2.0开源协议的对象存储服务。它兼容亚马逊S3云存储服务接口, 非常适合于存储大容量非结构化的数据, 例如图片、视频、日志文件、备份数据和容器/虚拟机镜像等, 而一个对象文件可以是任意大小, 从几kb到最大5T不等。

MinIO是一个非常轻量的服务,可以很简单的和其他应用的结合, 类似 NodeJS, Redis 或者 MySQL。



官网: <https://min.io/> <http://www.minio.org.cn/>

对象存储服务 (Object Storage Service, OSS) 是一种海量、安全、低成本、高可靠的云存储服务, 适合存放任意类型的文件。容量和处理能力弹性扩展, 多种存储类型供选择, 全面优化存储成本。

对于中小型企业, 如果不选择存储上云, 那么 Minio 是个不错的选择, 麻雀虽小, 五脏俱全。当然 Minio 除了直接作为对象存储使用, 还可以作为云上对象存储服务的网关层, 无缝对接到 Amazon S3、Microsoft Azure。

在中国: 阿里巴巴、腾讯、百度、中国联通、华为、中国移动等等9000多家企业也都在使用MinIO产品。

Minio优点

- 部署简单: 一个single二进制文件即是一切, 还可支持各种平台。
- minio支持海量存储, 可按zone扩展(原zone不受任何影响), 支持单个对象最大5TB;
- 兼容Amazon S3接口, 充分考虑开发人员的需求和体验;

- 低冗余且磁盘损坏高容忍，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n/2块disk损坏的情况下依然可以读出数据(n为一个纠删码集合(Erasure Coding Set)中的disk数量)。并且这种损坏恢复是基于单个对象的，而不是基于整个存储卷的。
- 读写性能优异

Our HDD results running on 16 node Minio cluster were:

Setup	Avg Read Throughput (GET)	Avg Write Throughput (PUT)
Distributed	10.81 GB/s	8.57 GB/s
Distributed with Encryption	9.38 GB/s	6.91 GB/s

Our NVMe results running on an 8 node MinIO cluster were:

Setup	Avg Read Throughput (GET)	Avg Write Throughput (PUT)
Distributed	38.7 GB/s	34.4 GB/s
Distributed with Encryption	36.9 GB/s	34.6 GB/s

1.2 MinIO的基础概念

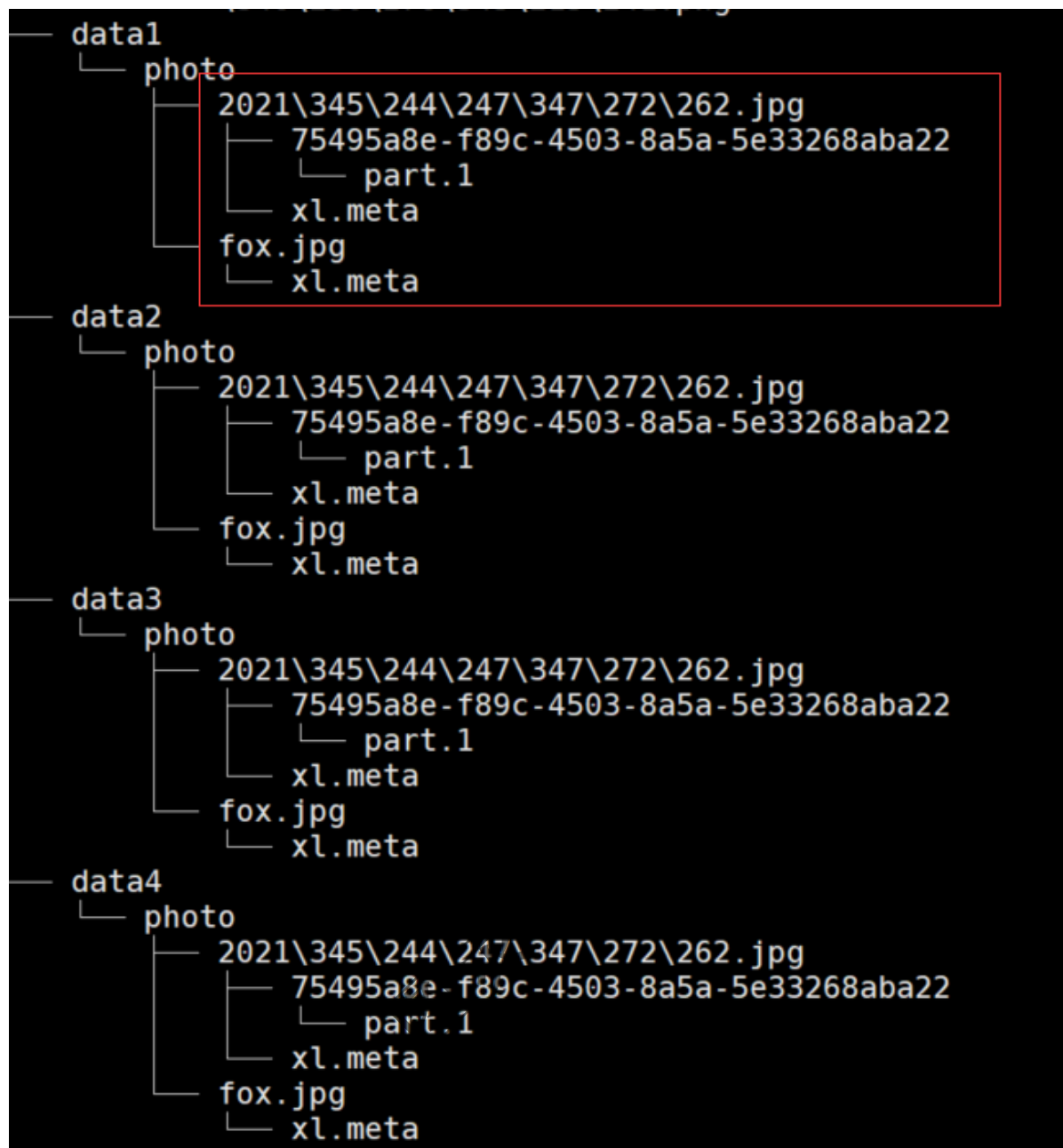
- **Object**: 存储到 Minio 的基本对象，如文件、字节流，Anything...
- **Bucket**: 用来存储 Object 的逻辑空间。每个 Bucket 之间的数据是相互隔离的。对于客户端而言，就相当于一个存放文件的顶层文件夹。
- **Drive**: 即存储数据的磁盘，在 MinIO 启动时，以参数的方式传入。Minio 中所有的对象数据都会存储在 Drive 里。
- **Set**: 即一组 Drive 的集合，分布式部署根据集群规模自动划分一个或多个 Set，每个 Set 中的 Drive 分布在不同位置。一个对象存储在一个 Set 上。(For example: {1...64} is divided into 4 sets each of size 16.)
 - 一个对象存储在一个Set上
 - 一个集群划分为多个Set
 - 一个Set包含的Drive数量是固定的，默认由系统根据集群规模自动计算得出
 - 一个SET中的Drive尽可能分布在不同的节点上

1.3 纠删码EC (Erasure Code)

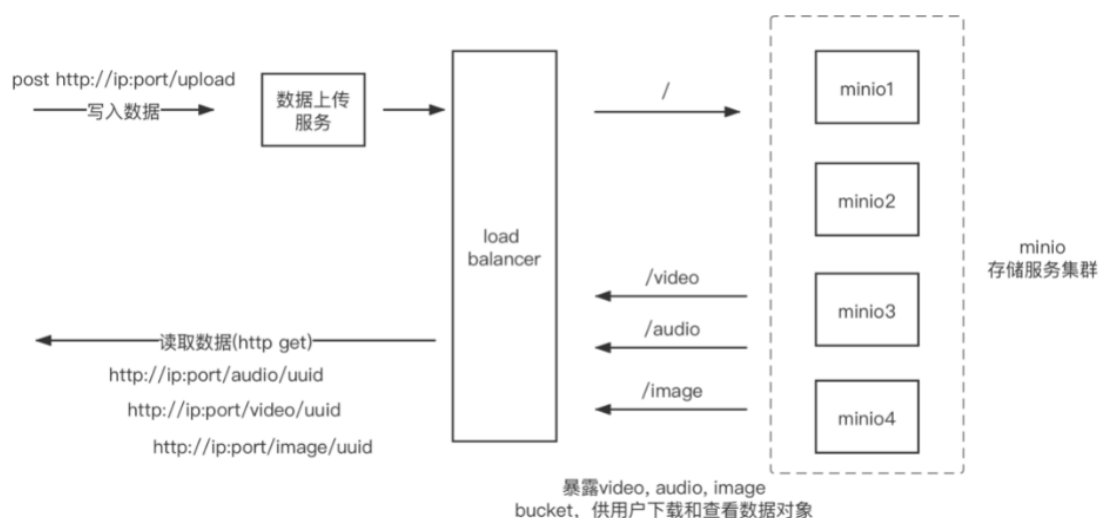
MinIO 使用纠删码机制来保证高可靠性，使用 highwayhash 来处理数据损坏 (Bit Rot Protection)。关于纠删码，简单来说就是可以通过数学计算，把丢失的数据进行还原，它可以将n份原始数据，增加m份数据，并能通过n+m份中的任意n份数据，还原为原始数据。即如果有任意小于等于m份的数据失效，仍然能通过剩下的数据还原出来。

1.4 存储形式

文件对象上传到 MinIO，会在对应的数据存储磁盘中，以 Bucket 名称为目录，文件名称为下一级目录，文件名下是 part.1 和 xl.meta(老版本，最新版本如下图)，前者是编码数据块及检验块，后者是元数据文件。



1.5 存储方案

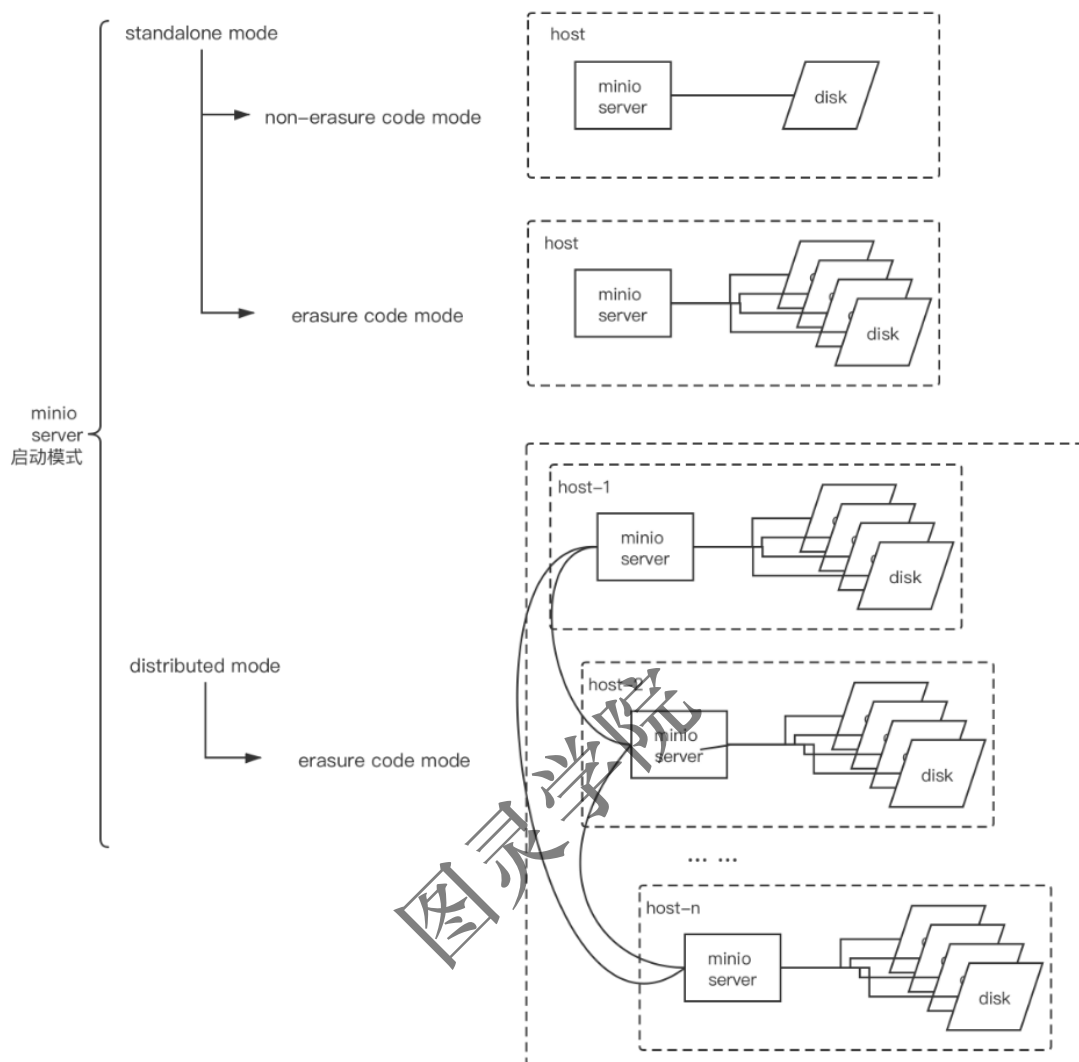


2. Minio环境搭建

官方文档: <https://docs.min.io/docs/>

中文文档: <http://docs.minio.org.cn/docs/> (没有及时更新, 容易被坑)

minio支持多种server启动模式:



2.1 单机部署

minio server的standalone模式, 即要管理的磁盘都在host本地。该启动模式一般仅用于实验环境、测试环境的验证和学习使用。在standalone模式下, 还可以分为**non-erasure code mode**和**erasure code mode**。

non-erasure code mode

在此启动模式下, 对于每一份对象数据, minio直接在data下面存储这份数据, 不会建立副本, 也不会启用纠删码机制。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。一旦遇到多于一个磁盘参数, minio server会自动启用**erasure code mode**。**erasure code**对磁盘的个数是有要求的, 如不满足要求, 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)至少为4个。

基于centos7

操作系统	CPU架构	地址
GNU/Linux	64-bit Intel	http://dl.minio.org.cn/server/minio/release/linux-amd64/minio

```
wget -q http://dl.minio.org.cn/server/minio/release/linux-amd64/minio
chmod +x minio
#启动minio server服务, 指定数据存储目录/mnt/data
./minio server /mnt/data
```

```
[root@k8s soft]# chmod +x minio
[root@k8s soft]# ./minio server /mnt/data

You are running an older version of MinIO released 1 week ago
Update: Run `mc admin update`

API: http://192.168.3.14:9000 http://172.17.0.1:9000 http://172.18.0.1:9000 http://172.19.0.1:9000 http://127.0.0.1:9000
RootUser: minioadmin
RootPass: minioadmin

Console: http://192.168.3.14:37954 http://172.17.0.1:37954 http://172.18.0.1:37954 http://172.19.0.1:37954 http://127.0.0.1:37954
RootUser: minioadmin
RootPass: minioadmin

Command-line: https://docs.min.io/docs/minio-client-quickstart-guide
$ mc alias set myminio http://192.168.3.14:9000 minioadmin minioadmin

Documentation: https://docs.min.io

WARNING: Console endpoint is listening on a dynamic port (37954), please use --console-address ":PORT" to choose a static port.
WARNING: Detected default credentials 'minioadmin:minioadmin', we recommend that you change these values with 'MINIO_ROOT_USER' and 'MINIO_ROOT_PASSWORD' environment variables
```

默认用户名密码minioadmin:minioadmin, 修改默认用户名密码可以使用:

```
export MINIO_ROOT_USER=admin
export MINIO_ROOT_PASSWORD=12345678
```

默认的配置目录是\${HOME}/.minio, 可以通过--config-dir命令自定义配置目录:

```
./minio server --config-dir /mnt/config /mnt/data
```

控制台监听端口是动态生成的, 可以通过--console-address ":port"指定静态端口

```
./minio server --console-address ":50000" /mnt/data
```

```
[root@k8s soft]# ./minio server --console-address :50000 /mnt/data

You are running an older version of MinIO released 1 week ago
Update: Run 'mc admin update'

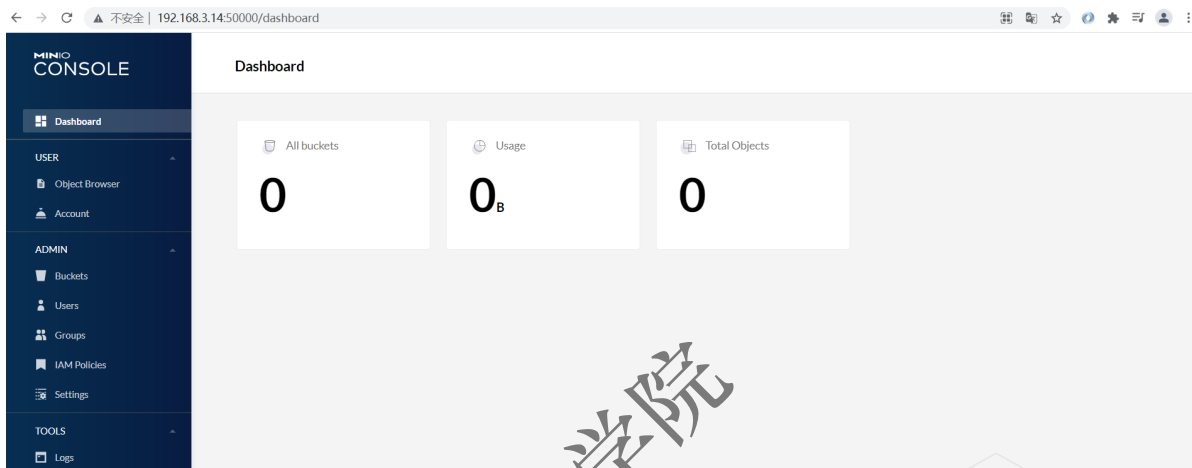
API: http://192.168.3.14:9000 http://172.17.0.1:9000 http://172.18.0.1:9000 http://172.19.0.1:9000 http://127.0.0.1:9000
RootUser: minioadmin
RootPass: minioadmin

Console: http://192.168.3.14:50000 http://172.17.0.1:50000 http://172.18.0.1:50000 http://172.19.0.1:50000 http://127.0.0.1:50000
RootUser: minioadmin
RootPass: minioadmin

Command-line: https://docs.min.io/docs/minio-client-quickstart-guide
$ mc alias set myminio http://192.168.3.14:9000 minioadmin minioadmin

Documentation: https://docs.min.io
WARNING: Detected default credentials 'minioadmin:minioadmin', we recommend that you change these values with 'MINIO_ROOT_USER' and 'MINIO_ROOT_PASSWORD' environment variables
```

访问minio控制台: <http://192.168.3.14:50000/dashboard>



基于docker

```
docker run -p 9000:9000 --name minio \
-v /mnt/data:/data \
-v /mnt/config:/root/.minio \
minio/minio server /data
```

存在问题： 浏览器无法访问minio控制台，因为没有对外暴露控制台端口

对外暴露minio控制台的端口，通过--console-address ":50000"指定控制台端口为静态端口

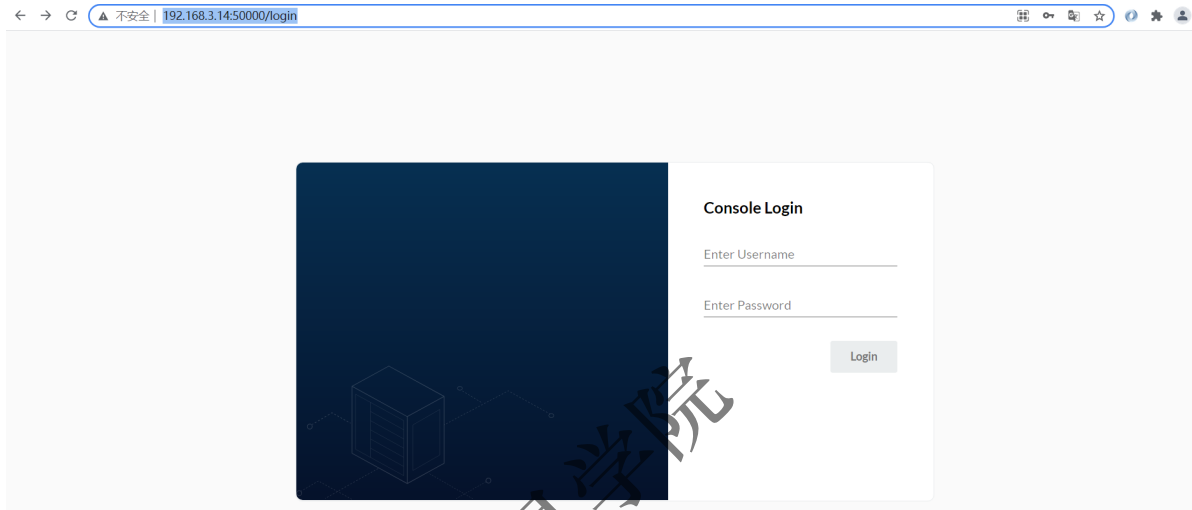
```
docker run -p 9000:9000 -p 50000:50000 --name minio \
-v /mnt/data:/data \
-v /mnt/config:/root/.minio \
minio/minio server --console-address ":50000" /data
```

```
[root@fox ~]# docker run -p 9000:9000,50000:50000 --name minio \
> -v /mnt/data:/data \
> -v /mnt/config:/root/.minio \
> minio/minio server --console-address ":50000" /data
docker: invalid publish opts format (should be name=value but got '9000:9000').
See 'docker run --help'.
[root@fox ~]# docker run -p 9000:9000 -p 50000:50000 --name minio \
> -v /mnt/data:/data \
> -v /mnt/config:/root/.minio \
> minio/minio server --console-address ":50000" /data
API: http://172.17.0.2:9000 http://127.0.0.1:9000

Console: http://172.17.0.2:50000 http://127.0.0.1:50000

Documentation: https://docs.min.io
WARNING: Detected default credentials 'minioadmin:minioadmin', we recommend that you change these values with 'MINIO_ROOT_USER' and 'MINIO_ROOT_PASSWORD' environment variables
```

访问minio控制台: <http://192.168.3.14:50000/>



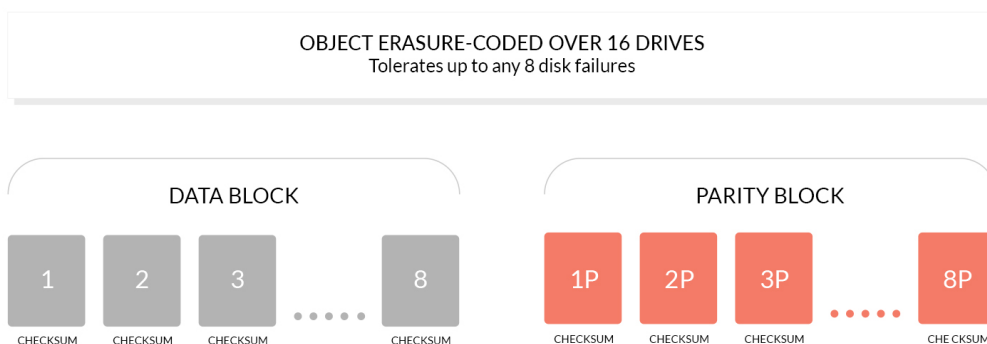
MinIO自定义用户名密码

```
docker run -d -p 9000:9000 -p 50000:50000 --name minio \
-e "MINIO_ROOT_USER=admin" \
-e "MINIO_ROOT_PASSWORD=12345678" \
-v /mnt/data:/data \
-v /mnt/config:/root/.minio \
minio/minio server --console-address ":50000" /data
```

minio纠删码模式

Minio使用纠删码 `erasure code` 和校验和 `checksum` 来保护数据免受硬件故障和无声数据损坏。即便您丢失一半数量 ($N/2$) 的硬盘, 您仍然可以恢复数据。

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。这意味着如果是12块盘, 一个对象会被分成6个数据块、6个奇偶校验块, 你可以丢失任意6块盘 (不管其是存放的数据块还是奇偶校验块), 你仍可以从剩下的盘中的数据中进行恢复。



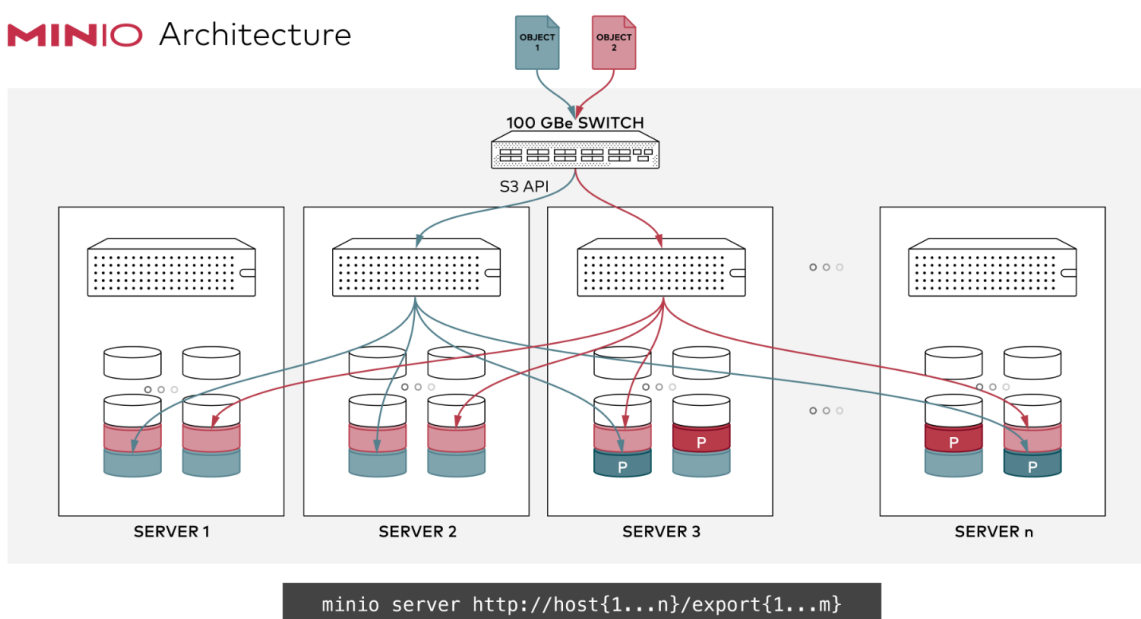
使用Minio Docker镜像，在8块盘中启动Minio服务：

```
docker run -d -p 9000:9000 -p 50000:50000 --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 ":50000"
```

2.2 分布式集群部署

分布式Minio可以让你将多块硬盘（甚至在不同的机器上）组成一个对象存储服务。由于硬盘分布在不同的节点上，分布式Minio避免了单点故障。

MINIO Architecture



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。而对于在存储领域，一般对于保证数据可靠性的方法主要有两类，一类是冗余法，一类是校验法。

冗余

冗余法最简单直接，即对存储的数据进行副本备份，当数据出现丢失，损坏，即可使用备份内容进行恢复，而副本 备份的多少，决定了数据可靠性的高低。这其中会有成本的考量，副本数据越多，数据越可靠，但需要的设备就越多，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是采用此种方式实现，如 Hadoop 的文件系统（3个副本），Redis 的集群，MySQL 的主备模式等。

校验

校验法即通过校验码的数学计算的方式，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，在数据传输和保存时经常用到，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优势

数据保护

分布式Minio采用 纠删码来防范多个节点宕机和位衰减 bit rot。

分布式Minio至少需要4个硬盘，使用分布式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，你的数据就是安全的。不过你需要至少有N/2+1个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，不过你需要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

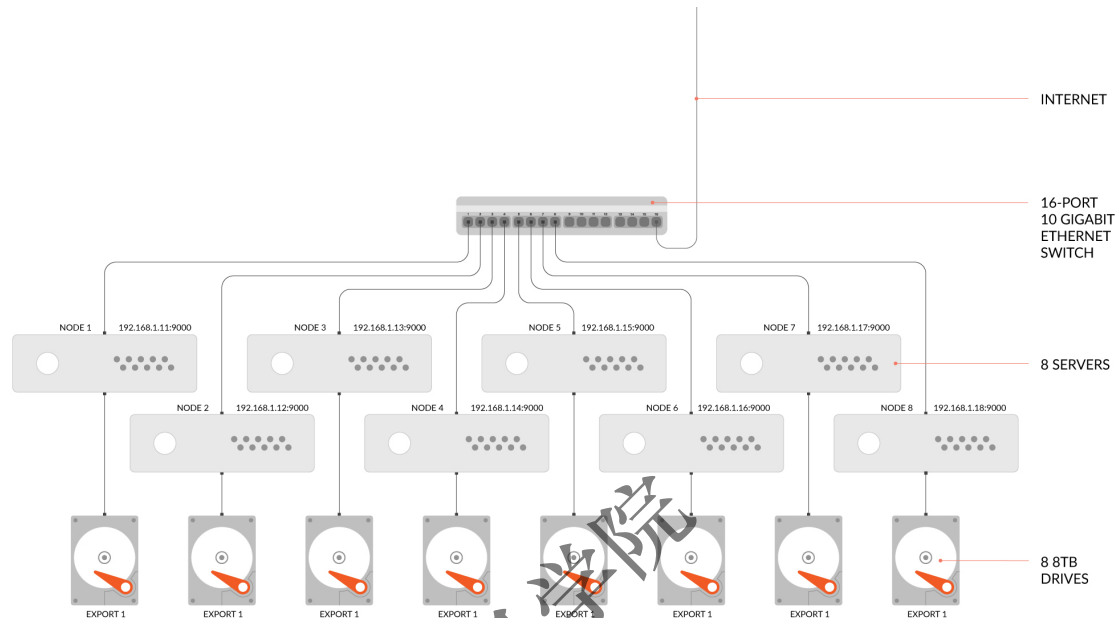
启动一个分布式Minio实例，你只需要把硬盘位置做为参数传给minio server命令即可，然后，你需要在所有其它节点运行同样的命令。

- 分布式Minio里所有的节点需要有同样的access密钥和secret密钥，这样这些节点才能建立联接。为了实现这个，你需要在执行minio server命令之前，先将access密钥和secret密钥export成环境变量。新版本使用MINIO_ROOT_USER&MINIO_ROOT_PASSWORD。
- 分布式Minio使用的磁盘里必须是干净的，里面没有数据。
- 下面示例里的IP仅供示例参考，你需要改成你真实用到的IP和文件夹路径。
- 分布式Minio里的节点时间差不能超过3秒，你可以使用NTP 来保证时间一致。
- 在Windows下运行分布式Minio处于实验阶段，请悠着点使用。

8个节点，每节点1块盘

启动分布式Minio实例，8个节点，每节点1块盘，需要在8个节点上都运行下面的命令：

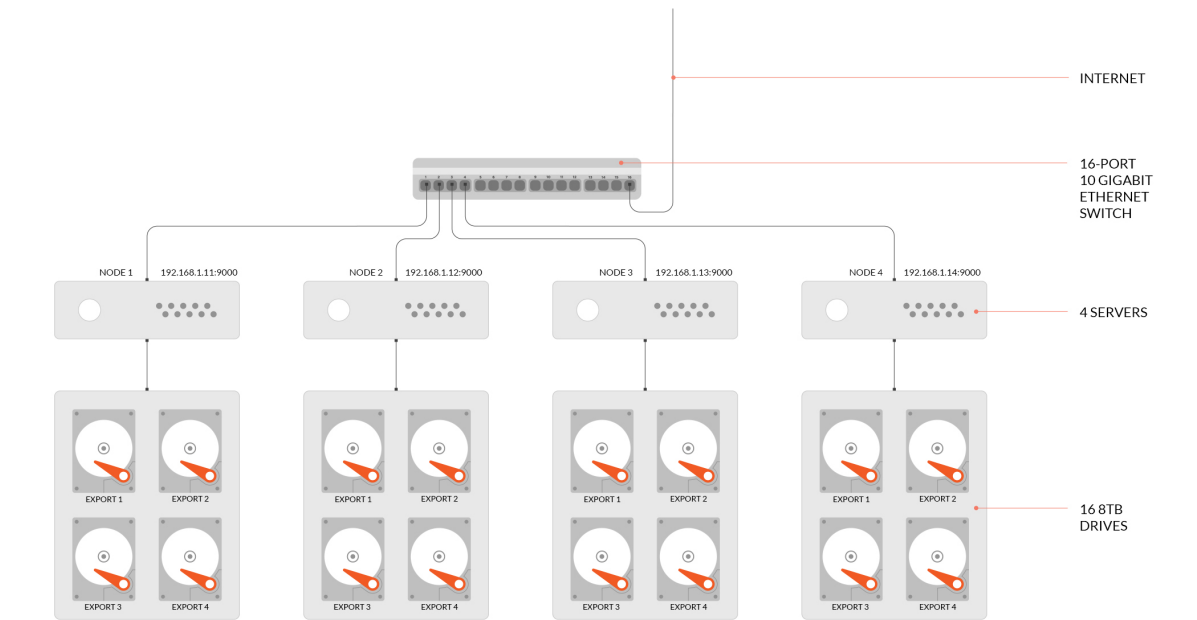
```
export MINIO_ROOT_USER=admin
export MINIO_ROOT_PASSWORD=12345678
minio server http://192.168.1.11/export1 http://192.168.1.12/export2 \
    http://192.168.1.13/export3 http://192.168.1.14/export4 \
    http://192.168.1.15/export5 http://192.168.1.16/export6 \
    http://192.168.1.17/export7 http://192.168.1.18/export8
```



4节点，每节点4块盘

启动分布式Minio实例，4节点，每节点4块盘，需要在4个节点上都运行下面的命令

```
export MINIO_ROOT_USER=admin
export MINIO_ROOT_PASSWORD=12345678
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://192.168.1.12/export1 http://192.168.1.12/export2 \
    http://192.168.1.12/export3 http://192.168.1.12/export4 \
    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://192.168.1.14/export1 http://192.168.1.14/export2 \
    http://192.168.1.14/export3 http://192.168.1.14/export4
```



通过脚本方式启动

```
export MINIO_ROOT_USER=admin
export MINIO_ROOT_PASSWORD=12345678
MINIO_HOME=/usr/local/soft
MINIO_HOST=192.168.3.14

for i in {01..04}; do
    nohup ${MINIO_HOME}/minio server --address ":90${i}" --console-address ":500${i}" http://${MINIO_HOST}:9001/mnt/data01 http://${MINIO_HOST}:9002/
    mnt/data02 http://${MINIO_HOST}:9003/mnt/data03 http://${MINIO_HOST}:9004/mnt/data04 > ${MINIO_HOME}/minio_90${i}.log 2>&1 &
done
```

执行脚本后查看日志和进程

```
[root@fox soft]# tail -f minio-9002.log
Waiting for the first server to format the disks.
Waiting for all MinIO sub-systems to be initialized.. lock acquired
All MinIO sub-systems initialized successfully
Waiting for all MinIO IAM sub-system to be initialized.. lock acquired
Status:      4 Online, 0 Offline.
API: http://192.168.3.14:9002 http://172.17.0.1:9002 http://172.18.0.1:9002 http://172.19.0.1:9002 http://127.0.0.1:9002
Console: http://192.168.3.14:50002 http://172.17.0.1:50002 http://172.18.0.1:50002 http://172.19.0.1:50002 http://127.0.0.1:50002
Documentation: https://docs.min.io

[root@fox soft]# ps -ef|grep minio
root      3282      1   1 21:03 pts/2    00:00:04 /usr/local/soft/minio server --address :9001 --console-address :50001 http://192.168.3.14:9001/mnt/data
01 http://192.168.3.14:9002/mnt/data02 http://192.168.3.14:9003/mnt/data03 http://192.168.3.14:9004/mnt/data04
root      3283      1   2 21:03 pts/2    00:00:09 /usr/local/soft/minio server --address :9002 --console-address :50002 http://192.168.3.14:9001/mnt/data
01 http://192.168.3.14:9002/mnt/data02 http://192.168.3.14:9003/mnt/data03 http://192.168.3.14:9004/mnt/data04
root      3284      1   1 21:03 pts/2    00:00:05 /usr/local/soft/minio server --address :9003 --console-address :50003 http://192.168.3.14:9001/mnt/data
01 http://192.168.3.14:9002/mnt/data02 http://192.168.3.14:9003/mnt/data03 http://192.168.3.14:9004/mnt/data04
root      3285      1   1 21:03 pts/2    00:00:05 /usr/local/soft/minio server --address :9004 --console-address :50004 http://192.168.3.14:9001/mnt/data
01 http://192.168.3.14:9002/mnt/data02 http://192.168.3.14:9003/mnt/data03 http://192.168.3.14:9004/mnt/data04
root      3382    2331   0 21:10 pts/2    00:00:00 grep --color=auto minio
```

测试上传:

springcloud

All Buckets / springcloud

Search Objects

🔄

🕒

🗑️ Delete Selected

+ Create Folder

📄 File

Select	Name	Last Modified	Size	Options
<input type="checkbox"/>	12 Sentinel规则持久化实战及其源码分析.pdf	Sun Aug 08 2021 21:12:33 GMT+0800	4.2 MiB	📄 🗑️
<input type="checkbox"/>	2021大纲.jpg	Sun Aug 08 2021 21:05:14 GMT+0800	7.3 MiB	📄 🗑️
<input type="checkbox"/>	apache-skywalking-apm-es7-8.4.0.tar.gz	Sun Aug 08 2021 21:14:26 GMT+0800	168 MiB	📄 🗑️
<input type="checkbox"/>	fox.jpg	Sun Aug 08 2021 21:05:24 GMT+0800	17 MiB	📄 🗑️

存储结构

```
data01
└─ springcloud
   └─ 12\ Sentinel\350\247\204\345\210\231\346\214\201\344\271\205\345\214\226\345\256\236\346\210\230\345\217\212\345\205\266\346\272\220\347\240\
      201\345\210\206\346\236\220.pdf
      └─ 07217ec5-1754-4606-9ce6-05e824ec8b87
         └─ part.1
            └─ xl.meta
      └─ 2021\345\244\247\347\272\262.jpg
         └─ 3a458779-c508-43a1-888a-15123db414d0
            └─ part.1
               └─ xl.meta
      └─ apache-skywalking-apm-es7-8.4.0.tar.gz
         └─ 156d6f44-d0b9-4cf5-9504-2f8b47c22406
            └─ part.1
               └─ part.10
                  └─ part.11
                     └─ part.2
                        └─ part.3
                           └─ part.4
                              └─ part.5
                                 └─ part.6
                                    └─ part.7
                                       └─ part.8
                                          └─ part.9
                                             └─ xl.meta
      └─ fox.jpg
         └─ 08156756-afe0-4af8-8c8d-alc8e112a28c
            └─ part.1
               └─ part.2
                  └─ xl.meta

data02
└─ springcloud
   └─ 12\ Sentinel\350\247\204\345\210\231\346\214\201\344\271\205\345\214\226\345\256\236\346\210\230\345\217\212\345\205\266\346\272\220\347\240\
      201\345\210\206\346\236\220.pdf
      └─ 07217ec5-1754-4606-9ce6-05e824ec8b87
         └─ part.1
            └─ xl.meta
      └─ 2021\345\244\247\347\272\262.jpg
         └─ 3a458779-c508-43a1-888a-15123db414d0
            └─ part.1
               └─ xl.meta
      └─ apache-skywalking-apm-es7-8.4.0.tar.gz
         └─ 156d6f44-d0b9-4cf5-9504-2f8b47c22406
            └─ part.1
               └─ part.10
                  └─ part.11
                     └─ part.2
                        └─ part.3
                           └─ part.4
                              └─ part.5
                                 └─ part.6
                                    └─ part.7
                                       └─ part.8
                                          └─ part.9
                                             └─ xl.meta
      └─ fox.jpg
         └─ 08156756-afe0-4af8-8c8d-alc8e112a28c
            └─ part.1
               └─ part.2
                  └─ xl.meta

data03
└─ springcloud
   └─ 12\ Sentinel\350\247\204\345\210\231\346\214\201\344\271\205\345\214\226\345\256\236\346\210\230\345\217\212\345\205\266\346\272\220\347\240\
      201\345\210\206\346\236\220.pdf
      └─ 07217ec5-1754-4606-9ce6-05e824ec8b87
         └─ part.1
            └─ xl.meta
      └─ 2021\345\244\247\347\272\262.jpg
         └─ 3a458779-c508-43a1-888a-15123db414d0
            └─ part.1
               └─ xl.meta
      └─ apache-skywalking-apm-es7-8.4.0.tar.gz
         └─ 156d6f44-d0b9-4cf5-9504-2f8b47c22406
            └─ part.1
               └─ part.10
                  └─ part.11
                     └─ part.2
                        └─ part.3
                           └─ part.4
                              └─ part.5
                                 └─ part.6
                                    └─ part.7
                                       └─ part.8
                                          └─ part.9
                                             └─ xl.meta
      └─ fox.jpg
         └─ 08156756-afe0-4af8-8c8d-alc8e112a28c
            └─ part.1
               └─ part.2
                  └─ xl.meta

data04
└─ springcloud
   └─ 12\ Sentinel\350\247\204\345\210\231\346\214\201\344\271\205\345\214\226\345\256\236\346\210\230\345\217\212\345\205\266\346\272\220\347\240\
      201\345\210\206\346\236\220.pdf
      └─ 07217ec5-1754-4606-9ce6-05e824ec8b87
         └─ part.1
            └─ xl.meta
      └─ 2021\345\244\247\347\272\262.jpg
         └─ 3a458779-c508-43a1-888a-15123db414d0
            └─ part.1
               └─ xl.meta
      └─ apache-skywalking-apm-es7-8.4.0.tar.gz
         └─ 156d6f44-d0b9-4cf5-9504-2f8b47c22406
            └─ part.1
               └─ part.10
                  └─ part.11
                     └─ part.2
                        └─ part.3
                           └─ part.4
                              └─ part.5
                                 └─ part.6
                                    └─ part.7
                                       └─ part.8
                                          └─ part.9
                                             └─ xl.meta
      └─ fox.jpg
         └─ 08156756-afe0-4af8-8c8d-alc8e112a28c
            └─ part.1
               └─ part.2
                  └─ xl.meta

data1
└─ photo
   └─ 2021\345\244\247\347\272\262.jpg
      └─ 75495a8e-f89c-4503-8a5a-5e33268aba22
         └─ part.1
```

使用Docker Compose部署MinIO

<https://docs.min.io/docs/deploy-minio-on-docker-compose.html>

要在Docker Compose上部署分布式MinIO，请下载[docker-compose.yaml](#)和[nginx.conf](#)到你当前的工作目录。

```
docker-compose pull
docker-compose up
```

扩展现有的分布式集群

例如我们是通过区的方式启动MinIO集群，命令行如下：

```
export MINIO_ROOT_USER=admin
export MINIO_ROOT_PASSWORD=12345678
minio server http://host{1...32}/export{1...32}
```

MinIO支持通过命令，指定新的集群来扩展现有集群（纠删码模式），命令行如下：

```
export MINIO_ROOT_USER=admin
export MINIO_ROOT_PASSWORD=12345678
minio server http://host{1...32}/export{1...32}
http://host{33...64}/export{1...32}
```

现在整个集群就扩展了1024个磁盘，总磁盘变为2048个，新的对象上传请求会自动分配到最少使用的集群上。通过以上扩展策略，您就可以按需扩展您的集群。重新配置后重启集群，会立即在集群中生效，并对现有集群无影响。如上命令中，我们可以把原来的集群看做一个区，新增集群看做另一个区，新对象按每个区域中的可用空间比例放置在区域中。在每个区域内，基于确定性哈希算法确定位置。

说明：您添加的每个区域必须具有与原始区域相同的磁盘数量（纠删码集）大小，以便维持相同的数据冗余SLA。例如，第一个区有8个磁盘，您可以将集群扩展为16个、32个或1024个磁盘的区域，您只需确保部署的SLA是原始区域的倍数即可。

基于nginx实现loadbalancer

```
upstream minio {
    server 192.168.3.14:9001;
    server 192.168.3.14:9002;
    server 192.168.3.14:9003;
    server 192.168.3.14:9004;
}

upstream console {
    ip_hash;
    server 192.168.3.14:50001;
    server 192.168.3.14:50002;
    server 192.168.3.14:50003;
    server 192.168.3.14:50004;
}

server {
    listen      9000;
    listen  [::]:9000;
```

```

server_name localhost;

# To allow special characters in headers
ignore_invalid_headers off;
# Allow any size file to be uploaded.
# Set to a value such as 1000m; to restrict file size to a specific
value
client_max_body_size 0;
# To disable buffering
proxy_buffering off;

location / {
    proxy_set_header Host $http_host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;

    proxy_connect_timeout 300;
    # Default is HTTP/1, keepalive is only enabled in HTTP/1.1
    proxy_http_version 1.1;
    proxy_set_header Connection "";
    chunked_transfer_encoding off;

    proxy_pass http://minio;
}
}
server {
    listen 50000;
    listen [::]:50000;
    server_name localhost;

    # To allow special characters in headers
    ignore_invalid_headers off;
    # Allow any size file to be uploaded.
    # Set to a value such as 1000m; to restrict file size to a specific
value
    client_max_body_size 0;
    # To disable buffering
    proxy_buffering off;

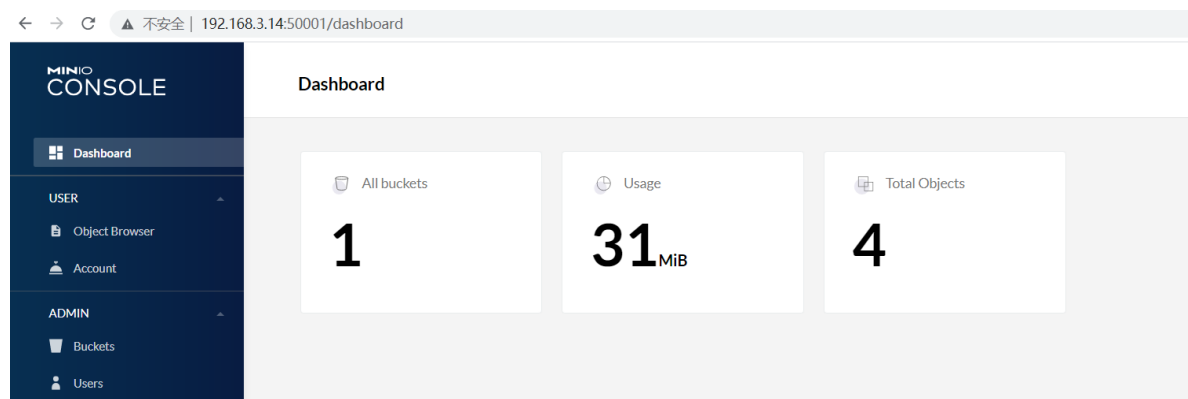
    location / {
        proxy_set_header Host $http_host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header X-NginX-Proxy true;

        proxy_connect_timeout 300;
        # Default is HTTP/1, keepalive is only enabled in HTTP/1.1
        proxy_http_version 1.1;
        proxy_set_header Connection "";
        chunked_transfer_encoding off;

        proxy_pass http://console;
    }
}

```

测试: <http://192.168.3.14:50001/dashboard>



2.3 Minio客户端使用

MinIO Client (mc)为ls, cat, cp, mirror, diff, find等UNIX命令提供了一种替代方案。它支持文件系统和兼容Amazon S3的云存储服务 (AWS Signature v2和v4)。

ls	列出文件和文件夹。
mb	创建一个存储桶或一个文件夹。
cat	显示文件和对象内容。
pipe	将一个STDIN重定向到一个对象或者文件或者STDOUT。
share	生成用于共享的URL。
cp	拷贝文件和对象。
mirror	给存储桶和文件夹做镜像。
find	基于参数查找文件。
diff	对两个文件夹或者存储桶比较差异。
rm	删除文件和对象。
events	管理对象通知。
watch	监视文件和对象的事件。
policy	管理访问策略。
config	管理mc配置文件。
update	检查软件更新。
version	输出版本信息。

部署客户端mc

平台	CPU架构	URL
GNU/Linux	64-bit Intel	http://dl.minio.org.cn/client/mc/release/linux-amd64/mc

```
wget http://dl.minio.org.cn/client/mc/release/linux-amd64/mc
chmod +x mc
./mc --help
mv mc /usr/local/sbin/
```

平台	CPU架构	URL
Microsoft Windows	64-bit Intel	http://dl.minio.org.cn/client/mc/release/windows-amd64/mc.exe

配置mc

mc 将所有的配置信息都存储在 ~/.mc/config.json 文件中

```
# 查询mc host配置
mc config host ls
# 添加minio服务
mc config host add minio-server http://192.168.3.14:9000 admin 12345678
# 删除host
mc config host remove minio-server
```

mc命令使用

ls - 列出存储桶和对象	mb - 创建存储桶	cat - 合并对象
cp - 拷贝对象	rm - 删除对象	pipe - Pipe到一个对象
share - 共享	mirror - 存储桶镜像	find - 查找文件和对象
diff - 比较存储桶差异	policy - 给存储桶或前缀设置访问策略	
config - 管理配置文件	watch - 事件监听	events - 管理存储桶事件
update - 管理软件更新	version - 显示版本信息	

上传下载

```
# 查询minio服务上的所有buckets(文件和文件夹)
mc ls minio-server

# 下载文件
mc cp minio-server/tulingmall/fox/fox.jpg /tmp/
#删除文件
mc rm minio-server/tulingmall/fox/fox.jpg
#上传文件
mc cp zookeeper.out minio-server/tulingmall/
```

```
[root@fox ~]# mc ls minio-server
[2021-08-05 15:46:48 CST] 0B tulingmall/
[root@fox ~]# mc cp minio-server/tulingmall/fox/fox.jpg /tmp/
...fox/fox.jpg: 17.02 MiB / 17.02 MiB |████████████████████| 503.40 MiB/s 0s [root@fox ~]#
[root@fox ~]# ls /tmp/
bmac_b2b002d09fbd9aea13329943374d1137 fox.jpg panelExec.log zookeeper
bt_install.pl hsuperfdata_root vmware-root
[root@fox ~]# mc rm minio-server/tulingmall/fox/fox.jpg
Removing `minio-server/tulingmall/fox/fox.jpg`.
[root@fox ~]# mc ls minio-server/tulingmall/fox/
[2021-08-05 15:49:27 CST] 504KiB 2.jpg
[root@fox ~]# mc cp zookeeper.out minio-server/tulingmall/
zookeeper.out: 3.45 KiB / 3.45 KiB |████████████████████| 73.24 KiB/s 0s [root@fox ~]#
[root@fox ~]# mc ls minio-server/tulingmall/
[2021-08-05 16:40:11 CST] 3.5KiB zookeeper.out
[2021-08-05 16:40:20 CST] 0B fox/
```


Bucket管理

```
# 创建bucket
mc mb minio-server/bucket01
# 删除bucket
mc rb minio-server/bucket02
# bucket不为空，可以强制删除 慎用
mc rb --force minio-server/bucket01
```

```
[root@fox ~]# mc mb minio-server/bucket01
Bucket created successfully `minio-server/bucket01`.
[root@fox ~]# mc mb minio-server/bucket02
Bucket created successfully `minio-server/bucket02`.
[root@fox ~]# mc rb minio-server/bucket02
Removed `minio-server/bucket02` successfully.
```

```
[root@fox ~]# mc rb minio-server/bucket01
mc: <ERROR> `minio-server/bucket01` is not empty. Retry this command with `--force` flag
if you want to remove `minio-server/bucket01` and all its contents
[root@fox ~]# mc rb --force minio-server/bucket01
Removed `minio-server/bucket01` successfully.
```

```
#查询bucket03磁盘使用情况
mc du minio-server/bucket03
```

```
[root@fox ~]# mc du minio-server
64MiB
[root@fox ~]# mc du minio-server/bucket03
31MiB bucket03
```

mc admin使用

MinIO Client (mc) 提供了“admin”子命令来对您的MinIO部署执行管理任务。

service	服务重启并停止所有MinIO服务器
update	更新更新所有MinIO服务器
info	信息显示MinIO服务器信息
user	用户管理用户
group	小组管理小组
policy	MinIO服务器中定义的策略管理策略
config	配置管理MinIO服务器配置
heal	修复MinIO服务器上的磁盘，存储桶和对象
profile	概要文件生成概要文件数据以进行调试
top	顶部提供MinIO的顶部统计信息
trace	跟踪显示MinIO服务器的http跟踪
console	控制台显示MinIO服务器的控制台日志
prometheus	Prometheus管理Prometheus配置
kms	kms执行KMS管理操作

用户管理

```
mc admin user --help
#新建用户
mc admin user add minio-server fox
mc admin user add minio-server fox02 12345678
#查看用户
mc admin user list minio-server
#禁用用户
mc admin user disable minio-server fox02
#启用用户
mc admin user disable minio-server fox02
#查看用户信息
mc admin user info minio-server fox
#删除用户
mc admin user remove minio-server fox02
```

```
[root@fox ~]# mc admin user add minio-server fox
Enter Secret Key:
Added user `fox` successfully.
[root@fox ~]# mc admin user list minio-server
enabled    fox
[root@fox ~]# mc admin user add minio-server fox02 12345678
Added user `fox02` successfully.
[root@fox ~]# mc admin user list minio-server
enabled    fox
enabled    fox02
[root@fox ~]# mc admin user disable minio-server fox02
Disabled user `fox02` successfully.
[root@fox ~]# mc admin user list minio-server
enabled    fox
disabled   fox02
[root@fox ~]# mc admin user disable minio-server fox02
Disabled user `fox02` successfully.
[root@fox ~]# mc admin user enable minio-server fox02
Enabled user `fox02` successfully.
[root@fox ~]# mc admin user list minio-server
enabled    fox
enabled    fox02
[root@fox ~]# mc admin user info minio-server fox
AccessKey: fox
Status: enabled
PolicyName:
MemberOf:
```

策略管理

policy命令，用于添加，删除，列出策略，获取有关策略的信息并为MinIO服务器上的用户设置策略。

```
mc admin policy --help
#列出MinIO上的所有固定策略
mc admin policy list minio-server
# 查看policy信息
mc admin policy info minio-server readwrite
```

```
[root@fox ~]# mc admin policy list minio-server
diagnostics
readonly
readwrite
writeonly
consoleAdmin
[root@fox ~]# mc admin policy list minio-server/bucket03
consoleAdmin
diagnostics
readonly
readwrite
writeonly
```

```
[root@fox ~]# mc admin policy info minio-server readwrite
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:*"
      ],
      "Resource": [
        "arn:aws:s3:::*"
      ]
    }
  ]
}
[root@fox ~]# mc admin policy info minio-server readonly
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetBucketLocation",
        "s3:GetObject"
      ],
      "Resource": [
        "arn:aws:s3:::*"
      ]
    }
  ]
}
```

添加新的策略

编写策略文件: /root/tulingmall.json

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
```

```

    "Action": [
      "s3:GetBucketLocation",
      "s3:GetObject"
    ],
    "Resource": [
      "arn:aws:s3:::tulingmall"
    ]
  }, {
    "Effect": "Allow",
    "Action": [
      "s3:*"
    ],
    "Resource": [
      "arn:aws:s3:::tulingmall/*"
    ]
  }
]
}

```

```

"Action": [
  "s3:GetBucketLocation",
  "s3:ListBucket",
  "s3:GetObject",
  "s3:PutObject",
  "s3:DeleteObject"
]

```

将tulingmall.json添加到策略数据库

```

# 添加新的策略
mc admin policy add minio-server tulingmall-admin /root/tulingmall.json
mc admin policy list minio-server

```

```

[root@fox ~]# vim tulingmall.json
[root@fox ~]# mc admin policy add minio-server tulingmall-admin /root/tulingmall.json
Added policy `tulingmall-admin` successfully.
[root@fox ~]# mc admin policy list minio-server
writeonly
consoleAdmin
diagnostics
readonly
readwrite
tulingmall-admin

```

```

mc admin user add minio-server fox03 12345678
# 设置用户的访问策略
mc admin policy set minio-server tulingmall-admin user=fox03

```

```

[root@fox ~]# mc admin user info minio-server fox03
AccessKey: fox03
Status: enabled
PolicyName: 
MemberOf: 
[root@fox ~]# mc admin policy set minio-server tulingmall-admin user=fox03
Policy `tulingmall-admin` is set on user `fox03`
[root@fox ~]# mc admin user info minio-server fox03
AccessKey: fox03
Status: enabled
PolicyName: tulingmall-admin
MemberOf: 

```

测试: fox03/12345678 登录minio控制台<http://192.168.3.14:50000/>, 只能操作tulingmall的bucket

3. Minio Java Client使用

MinIO Java Client SDK提供简单的API来访问任何与Amazon S3兼容的对象存储服务。

官方demo: <https://github.com/minio/minio-java>

官方文档: <https://docs.min.io/docs/java-client-api-reference.html>

引入依赖

```
<dependency>
  <groupId>io.minio</groupId>
  <artifactId>minio</artifactId>
  <version>8.3.0</version>
</dependency>

<dependency>
  <groupId>me.tongfei</groupId>
  <artifactId>progressbar</artifactId>
  <version>0.5.3</version>
</dependency>

<dependency>
  <groupId>com.squareup.okhttp3</groupId>
  <artifactId>okhttp</artifactId>
  <version>4.8.1</version>
</dependency>
```

3.1 文件上传

```
public class Fileuploader {
    public static void main(String[] args)
        throws IOException, NoSuchAlgorithmException, InvalidKeyException {
        try {
            // Create a minioClient with the MinIO server playground, its access key
            and secret key.
            MinioClient minioClient =
                MinioClient.builder()
                    .endpoint("http://192.168.3.14:9000")
                    .credentials("admin", "12345678")
                    .build();

            // 创建bucket
            String bucketName = "tulingmall";
            boolean exists =
                minioClient.bucketExists(BucketExistsArgs.builder().bucket(bucketName).build());
            if (!exists) {
                // 不存在, 创建bucket

                minioClient.makeBucket(MakeBucketArgs.builder().bucket(bucketName).build());
            }
        }
    }
}
```

```

// 上传文件
minioClient.uploadObject(
    UploadObjectArgs.builder()
        .bucket(bucketName)
        .object("tuling-mall-master.zip")
        .filename("F:\\mall\\tuling-mall-master.zip")
        .build());
System.out.println("上传文件成功");
} catch (MinioException e) {
    System.out.println("Error occurred: " + e);
    System.out.println("HTTP trace: " + e.httpTrace());
}
}
}

```

3.2 文件下载

```

public class DownloadDemo {

    public static void main(String[] args) {

        // Create a minioClient with the Minio Server playground, its access key
        and secret key.
        MinioClient minioClient =
            MinioClient.builder()
                .endpoint("http://192.168.3.14:9000")
                .credentials("admin", "12345678")
                .build();

        // Download object given the bucket, object name and output file name
        try {
            minioClient.downloadObject(
                DownloadObjectArgs.builder()
                    .bucket("tulingmall")
                    .object("fox/fox.jpg")
                    .filename("fox.jpg")
                    .build());

        } catch (Exception e) {
            e.printStackTrace();
        }

    }

}

```

3.3 Spring boot整合minio

构建MinioClient对象，并交给spring管理

```

@Data
@Component
@ConfigurationProperties(prefix = "minio")
public class MinioProperties {

```

```

        private String endpoint;
        private String accessKey;
        private String secretKey;
    }

    //yml
    minio:
        endpoint: http://192.168.3.14:9000
        accesskey: admin
        secretKey: 12345678

@Configuration
public class MinioConfig {

    @Autowired
    private MinioProperties minioProperties;

    @Bean
    public MinioClient minioClient(){
        MinioClient minioClient =
            MinioClient.builder()
                .endpoint(minioProperties.getEndpoint())
                .credentials(minioProperties.getAccessKey(),
minioProperties.getSecretKey())
                .build();
        return minioClient;
    }
}

```

实现文件上传，下载，删除操作

```

@RestController
@Slf4j
public class MinioController {

    @Autowired
    private MinioClient minioClient;

    @Value("${minio.bucketName}")
    private String bucketName;

    @GetMapping("/list")
    public List<Object> list() throws Exception {
        //获取bucket列表
        Iterable<Result<Item>> myObjects = minioClient.listObjects(
            ListObjectsArgs.builder().bucket(bucketName).build());
        Iterator<Result<Item>> iterator = myObjects.iterator();
        List<Object> items = new ArrayList<>();
        String format = "{ 'fileName': '%s', 'fileSize': '%s' }";
        while (iterator.hasNext()) {
            Item item = iterator.next().get();

```

```

        items.add(JSON.parse(String.format(format, item.objectName(),
formatFileSize(item.size()))));
    }
    return items;
}

@PostMapping("/upload")
public Res upload(@RequestParam(name = "file", required = false)
MultipartFile[] file) {

    if (file == null || file.length == 0) {
        return Res.error("上传文件不能为空");
    }

    List<String> orgfileNameList = new ArrayList<>(file.length);

    for (MultipartFile multipartFile : file) {
        String orgfileName = multipartFile.getOriginalFilename();
        orgfileNameList.add(orgfileName);
        try {
            //文件上传
            InputStream in = multipartFile.getInputStream();
            minioClient.putObject(
PutObjectArgs.builder().bucket(bucketName).object(orgfileName).stream(
                in, multipartFile.getSize(), -1)
                .contentType(multipartFile.getContentType())
                .build());

            in.close();
        } catch (Exception e) {
            log.error(e.getMessage());
            return Res.error("上传失败");
        }
    }

    Map<String, Object> data = new HashMap<String, Object>();
    data.put("bucketName", bucketName);
    data.put("fileName", orgfileNameList);
    return Res.ok("上传成功", data);
}

@RequestMapping("/download/{fileName}")
public void download(HttpServletResponse response, @PathVariable("fileName")
String fileName) {

    InputStream in = null;
    try {
        // 获取对象信息
        StatObjectResponse stat = minioClient.statObject(
StatObjectArgs.builder().bucket(bucketName).object(fileName).build());
        response.setContentType(stat.contentType());
        response.setHeader("Content-Disposition", "attachment;filename=" +
URLEncoder.encode(fileName, "UTF-8"));
        //文件下载
        in = minioClient.getObject(

```



```

        GetObjectArgs.builder()
            .bucket(bucketName)
            .object(fileName)
            .build());
        IOUtils.copy(in, response.getOutputStream());
    } catch (Exception e) {
        log.error(e.getMessage());
    } finally {
        if (in != null) {
            try {
                in.close();
            } catch (IOException e) {
                log.error(e.getMessage());
            }
        }
    }
}

}

@DeleteMapping("/delete/{fileName}")
public Res delete(@PathVariable("fileName") String fileName) {
    try {
        minioClient.removeObject(

RemoveObjectArgs.builder().bucket(bucketName).object(fileName).build());
    } catch (Exception e) {
        log.error(e.getMessage());
        return Res.error("删除失败");
    }
    return Res.ok("删除成功", null);
}

private static String formatFileSize(long files) {
    DecimalFormat df = new DecimalFormat("#.00");
    String fileSizeString = "";
    String wrongSize = "0B";
    if (files == 0) {
        return wrongSize;
    }
    if (files < 1024) {
        fileSizeString = df.format((double) files) + " B";
    } else if (files < 1048576) {
        fileSizeString = df.format((double) files / 1024) + " KB";
    } else if (files < 1073741824) {
        fileSizeString = df.format((double) files / 1048576) + " MB";
    } else {
        fileSizeString = df.format((double) files / 1073741824) + " GB";
    }
    return fileSizeString;
}

}

```

图灵学院