

**2017**级

《物联网数据存储与管理》课程

**实 验 报 告**

**姓 名 金修旭**

**学 号 U201714739**

**班 号 物联网1701班**

**日 期 2020.05.26**

目 录

[1 实验目的 1](#_Toc41506624)

[2 实验背景 1](#_Toc41506625)

[2.1 Account Server 1](#_Toc41506626)

[2.2 Container 1](#_Toc41506627)

[2.3 Object Server 2](#_Toc41506628)

[3 实验环境 3](#_Toc41506629)

[3.1 系统环境 3](#_Toc41506630)

[3.2 Minio-Server 3](#_Toc41506631)

[3.3 Minio-python API 5](#_Toc41506632)

[3.4 osm 7](#_Toc41506633)

[3.5 s3bench 8](#_Toc41506634)

[3.6 OpenStack Swift CLI 10](#_Toc41506635)

[4 实验内容 11](#_Toc41506636)

[4.1 对象存储技术实践 11](#_Toc41506637)

[4.2 对象存储性能分析 11](#_Toc41506638)

[5 实验过程 11](#_Toc41506639)

[5.1 环境配置 11](#_Toc41506640)

[6 实验总结 13](#_Toc41506641)

[参考文献 14](#_Toc41506642)

# 实验目的

1. 熟悉对象存储技术，代表性系统及其特性；
2. 实践对象存储系统，部署实验环境，进行初步测试；
3. 基于对象存储系统，架设实际应用，示范主要功能。

# 实验背景

OpenStack Object Storage是一个高度可用，分布式，最终一致的对象blob存储。可以使用Object Storage API创建，修改和获取对象和元数据，该API是作为一组Representational State Transfer（REST）Web服务实现的。

## Account Server

表示层次结构的顶级

服务提供商会创建帐户，并拥有该帐户中的所有资源。该帐户定义容器的命名空间。容器在两个不同的帐户中可能具有相同的名称。

在OpenStack环境中，帐户与项目或租户同义。

## Container

定义对象的命名空间。两个不同容器中具有相同名称的对象表示两个不同的对象。可以在帐户中创建任意数量的容器。

除了包含对象之外，还可以使用容器通过使用访问控制列表（ACL）来控制对对象的访问。不能使用单个对象存储ACL。

在容器级别配置和控制许多其他功能，例如对象版本控制。

可以在一个请求中批量删除多达10,000个容器。

可以在具有云提供商的预定义名称和定义的容器上设置存储策略。

## Object Server

存储数据内容，例如文档，图像等。还可以使用对象存储自定义元数据。

使用Object Storage API，

存储无限数量的对象。每个对象可以大到5 GB，这是默认值。可以配置最大对象大小。

1. 使用大对象创建上载和存储任何大小的对象。
2. 使用跨源资源共享来管理对象安全性。
3. 使用内容编码元数据压缩文件。
4. 使用内容处置元数据覆盖对象的浏览器行为。
5. 安排删除对象。
6. 在单个请求中批量删除最多10,000个对象。
7. 自动提取存档文件。
8. 生成一个URL，该URL提供对对象的时间限制GET访问。
9. 使用表单POST中间件从浏览器直接将对象上传到Object Storage系统。
10. 创建指向其他对象的符号链接。

# 实验环境

## 系统环境



## Minio-Server

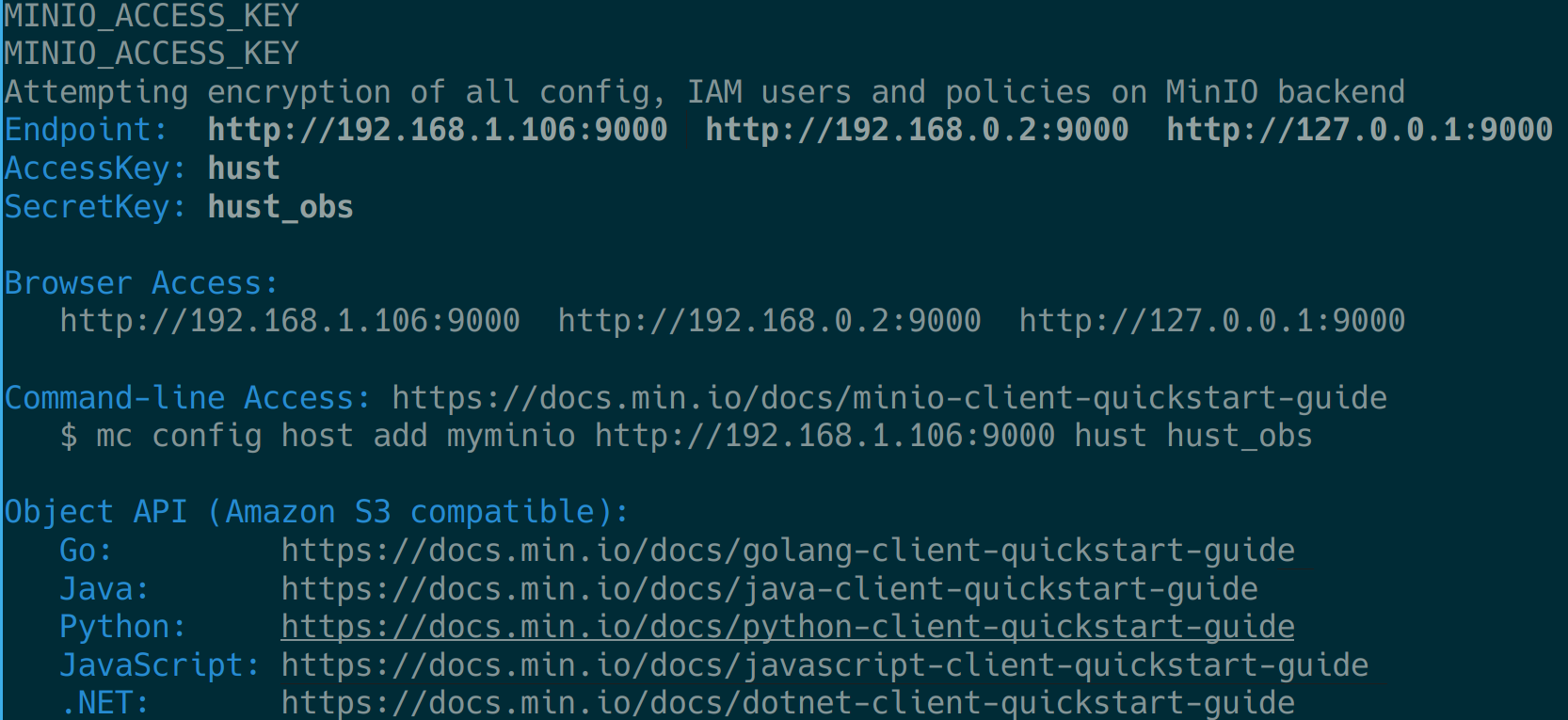


图3.2 Minio-Server启动验证

此时在浏览器打开127.0.0.1:9000(endpoint)，在打开的minio browser，输入自己的AccessKey和SerectKey登陆。

此时可以选择右下角红色标记随意添加bucket和上传文件，方便地实现类似网盘的功能。

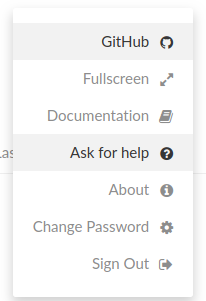


图3.4 账户管理相关选项

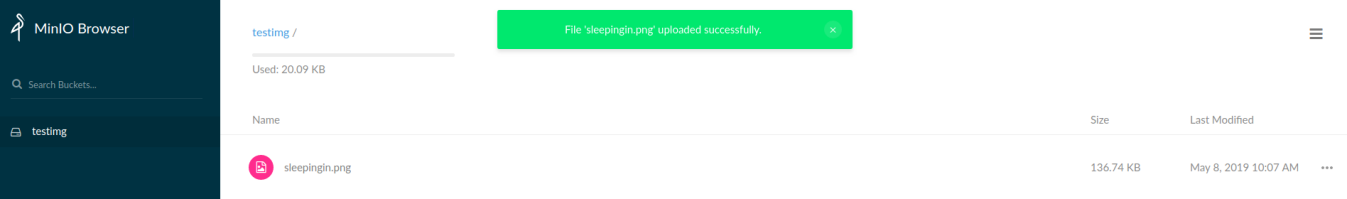


图3.5 Bucket/file测试图

注意，bucket有相应的命名限制，只能采用小写字母，而不允许下划线和相应大写字母的出现。

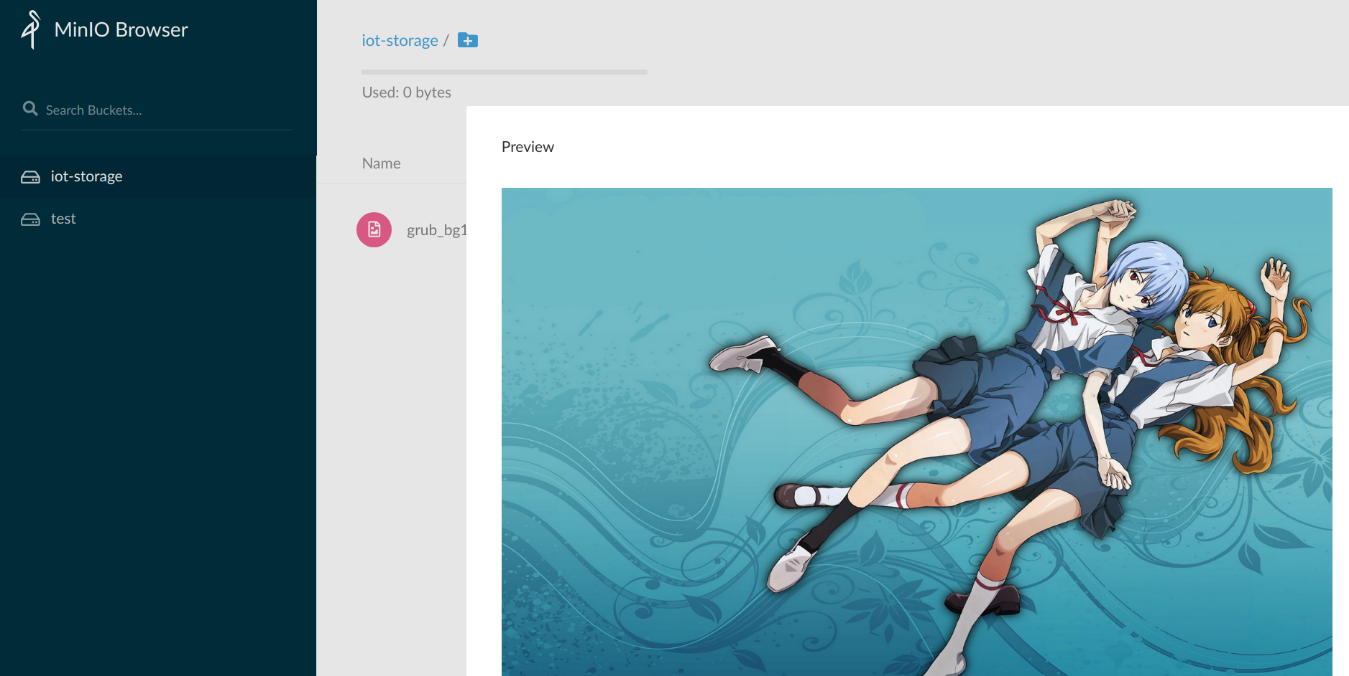


图3.6 本地文件验证

## Minio-python API

使用python-tutorial 进行API编程测试，使用官方测试仓库，代码如下

代码3.1 Minio-python API

from minio import Minio

from minio.error import ResponseError

from minio.error import (ResponseError, BucketAlreadyOwnedByYou,

                         BucketAlreadyExists)

minioClient = Minio('play.min.io:9000',

                  access\_key='Q3AM3UQ867SPQQA43P2F',

                  secret\_key='zuf+tfteSlswRu7BJ86wekitnifILbZam1KYY3TG',

                  secure=True)

# Make a bucket with the make\_bucket API call.

**try**:

       minioClient.make\_bucket("maylogs", location="us-east-1")

except BucketAlreadyOwnedByYou as err:

       pass

except BucketAlreadyExists as err:

       pass

except ResponseError as err:

**raise**

**else**:

        # Put an object 'pumaserver\_debug.log' with contents from 'pumaserver\_debug.log'.

**try**:

               minioClient.fput\_object('maylogs', 'pumaserver\_debug.log', '/tmp/pumaserver\_debug.log')

        except ResponseError as err:

               print(err)

运行结果如下

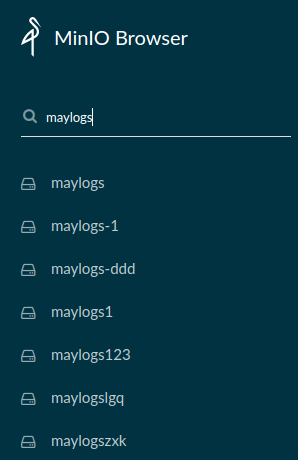


图3.8 Minio-python 测试结果

## osm

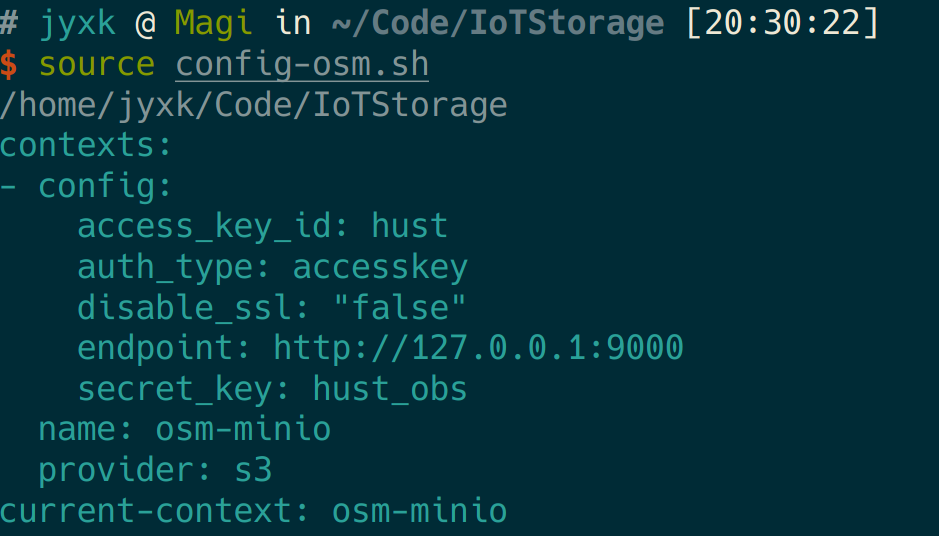


图.9 配置osm

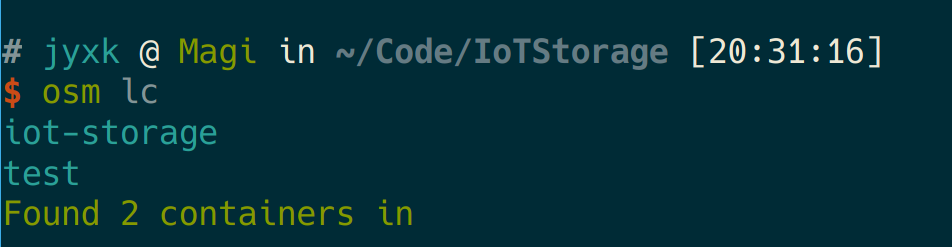


图3.10 此存储下的bucket

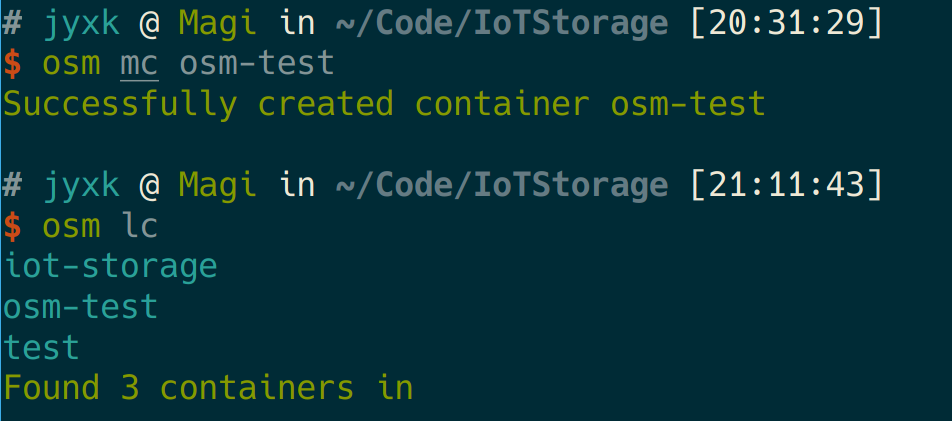


图3.11 创建bucket

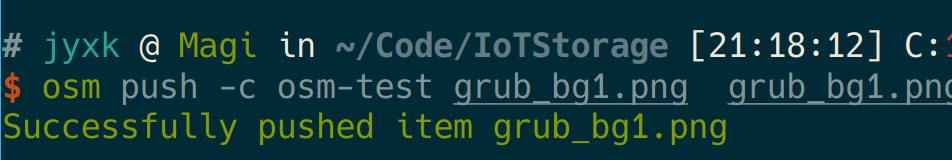


图3.12 上载IMG测试

## s3bench

使用脚本进行测试，其中参数可自行调节

# jyxk @ Magi in ~/Code/IoTStorage [21:26:31]   
$ ./run-s3bench.sh   
Test parameters   
endpoint(s): [http://127.0.0.1:9000]   
bucket: loadgen   
objectNamePrefix: loadgen   
objectSize: 0.0312 MB   
numClients: 8   
numSamples: 256   
verbose: %!d(bool=false)   
  
  
Generating in-memory sample data... Done (577.435µs)   
  
Running Write test...   
  
Running Read test...   
  
Test parameters   
endpoint(s): [http://127.0.0.1:9000]   
bucket: loadgen   
objectNamePrefix: loadgen   
objectSize: 0.0312 MB   
numClients: 8   
numSamples: 256   
verbose: %!d(bool=false)   
  
  
Results Summary for Write Operation(s)   
Total Transferred: 0.000 MB   
Total Throughput: 0.00 MB/s   
Total Duration: 0.092 s   
Number of Errors: 256   
  
  
Results Summary for Read Operation(s)   
Total Transferred: 0.000 MB   
Total Throughput: 0.00 MB/s   
Total Duration: 0.047 s   
Number of Errors: 256   
  
  
Cleaning up 256 objects...   
Deleting a batch of 256 objects in range {0, 255}... Succeeded   
Successfully deleted 256/256 objects in 46.355513ms

根据原始数据分析，读写最高速度均出现在客户端数量为32，样本数为1024，大小为512\*32时的情况。或许在我的机器上，针对较小的文件大小和较大的文件数，会发挥较好的表现。

## OpenStack Swift CLI

这里利用老师提供的open stack swift docker 的dockerfile进行安装，按照readme说明运行以下命令。

Build docker image

docker build -t openstack-swift-docker .

Prepare datavolume

docker run -v /srv --name SWIFT\_DATA busybox

Run container

docker run -d --name openstack-swift -p 12345:8080 --volumes-from SWIFT\_DATA -t openstack-swift-docker

Check container

docker logs openstack-swift

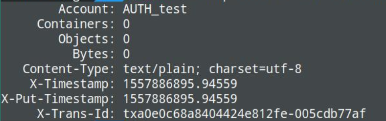
docker ps

Verify functionality

swift -A http://127.0.0.1:12345/auth/v1.0 -U test:tester -K testing stat

swift -A http://127.0.0.1:12345/auth/v1.0 -U test:tester -K testing list

然后使用类似上述命令，测试Swift Docker启动



# 实验内容

## [对象存储技术实践](#_Toc509412099)

1. 采用Docker进行环境相关配置
2. 相关API测试以及测试程序编写
3. 测试数据下载分析

## 对[象存储性能分析](#_Toc509412100)

1. 对于不同情况下的测试结果进行绘图
2. 测试结果分析

# 实验过程

## 环境配置

1. 安装软件包

由于基于Arch系统优良的包管理和AUR，所需要的包和依赖可以一键安装。

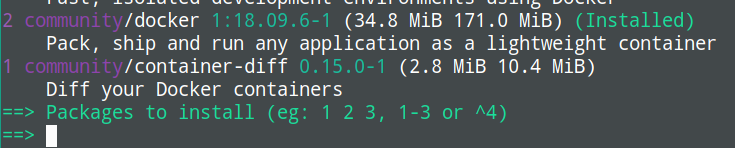


图5.1 yay安装相关包

2）使用s3bench脚本测试当前环境。

脚本：

s3bench=~/go/bin/s3bench

if [ -n "$GOPATH" ]; then

s3bench=$GOPATH/bin/s3bench

fi

# -accessKey Access Key

# -accessSecret Secret Key

# -bucket=loadgen Bucket for holding all test objects.

# -endpoint=http://127.0.0.1:9000 Endpoint URL of object storage service being tested.

# -numClients=8 Simulate 8 clients running concurrently.

# -numSamples=256 Test with 256 objects.

# -objectNamePrefix=loadgen Name prefix of test objects.

# -objectSize=1024 Size of test objects.

for i in {1..15..2}

do

for j in {16..512..16}

do

$s3bench \

-accessKey=hust \

-accessSecret=hust\_obs \

-bucket=loadgen \

-endpoint=http://127.0.0.1:9000 \

-numClients=$i \

-numSamples=$j \

-objectNamePrefix=loadgen \

-objectSize=$(( 1024\*32 ))

done

done

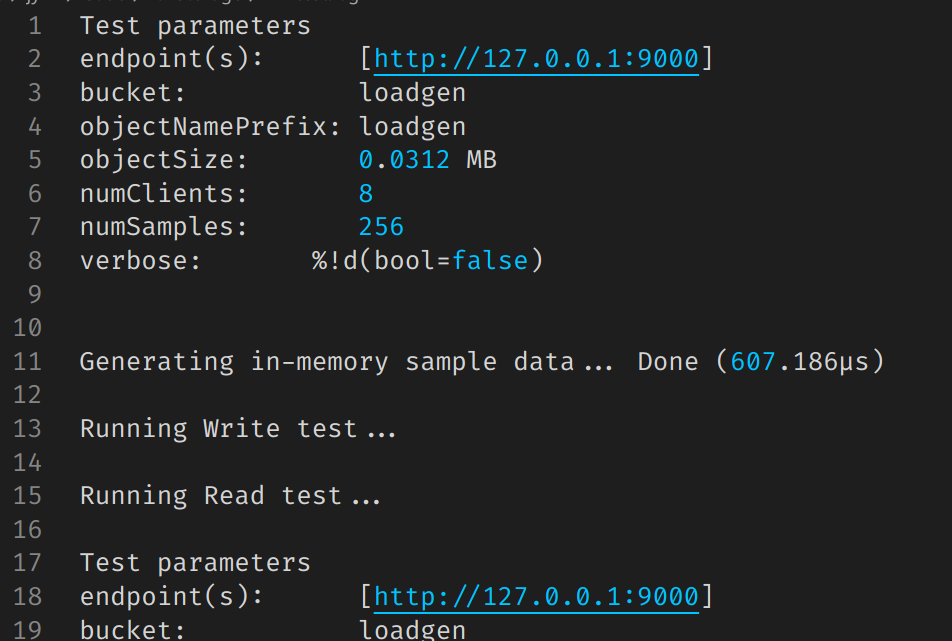
# build your own test script with designated '-numClients', '-numSamples' and '-objectSize'

# 1. Use loop structure to generate test batch (E.g.: to re-evaluate multiple s3 servers under the same configuration, or to gather data from a range of parameters);

# 2. Use redirection (the '>' operator) for storing program output to text files;

# 3. Observe and analyse the underlying relation between configuration parameters and performance metrics.

结果如下：



具体数据见test.log。

3.7

# 实验总结

在实验过程中，环境配置可以说相对来说比较简单，但是某些玄学BUG可能无明显报错信息，转而转向Docker，由于所有的包括服务端和测试端均在Docker中启动，存在部分Automated Dcoker入口写死，需要自己手动更改Docker文件进行相应的编译，在此过程中提高了Docker的熟练程度。

使用minio与osm搭配让我体会了基本的对象存储系统的工作模式，体会到了对象存储系统与其他类别存储系统的不同，感受到了在部署以及使用两方面，对象存储系统的优势。

最后，感谢实验过程中提供帮助的老师和同学以及所参阅资料的提供者。

# 参考文献

1. ARNOLD J. OpenStack Swift[M]. O’Reilly Media, 2014.
2. ZHENG Q, CHEN H, WANG Y等. COSBench: A Benchmark Tool for Cloud Object Storage Services[C]//2012 IEEE Fifth International Conference on Cloud Computing. 2012: 998–999.
3. WEIL S A, BRANDT S A, MILLER E L等. Ceph: A Scalable, High-performance Distributed File System[C]//Proceedings of the 7th Symposium on Operating Systems Design and Implementation. Berkeley, CA, USA: USENIX Association, 2006: 307–320.
4. Swift Document.URL:[https://docs.openstack.org/swift](https://docs.openstack.org/swift/latest/container.html)