Seminar Report

# Comparison of Object Storage Systems

Maximilian Schlensog

MatrNr: 21567922

Supervisor: Michael B. Khani

Georg-August-Universität Göttingen
Institute of Computer Science

March 28, 2025

# Abstract

With the increasing prevalence and speed of networks - especially the internet - whilst the amount and size of data increases as well, Object Storage Systems (OSS), a solution for storing large amounts of data while being robust and safe, has come into the spotlight more and more. OSS, a distributed form of architecture for data storage, was proposed to increase safety and efficiency of exactly the aforementioned situations, cold(er) data of exorbitant size and amount. In this report I will look at OSS-Architecture in general to give an overlook of the topic. But since there are a lot of different OSS solutions, making the product landscape difficult to navigate, some solutions will be presented and compared. The OSS solutions that shall be compared, will be presented on their own. After which the results of the benchmarking and testing will be presented and compared between solutions, with a conclusion/recommendation for specific cases.

## Declaration on the use of ChatGPT and comparable tools in the context of examinations

In this work I have used ChatGPT or another AI as follows:

- ☐ Not at all
- ☑ During brainstorming
- ☐ When creating the outline
- ☐ To write individual passages, altogether to the extent of 0% of the entire text
- ☑ For the development of software source texts
- ☑ For optimizing or restructuring software source texts
- ☐ For proofreading or optimizing
- ☐ Further, namely: -

I hereby declare that I have stated all uses completely.
Missing or incorrect information will be considered as an attempt to cheat.

# Contents

# List of Tables

# List of Figures

# List of Listings

# List of Abbreviations

| | |
|---|---|
| **AWS** | *Amazon Web Services* |
| **ECMWF** | *European Centre for Medium-Range Weather Forecast* |
| **FDB** | *Fields Database* |
| **OSS** | *Object Storage System* |
| **S3** | *Simple Storage Solution* |

# 1 Introduction

Object Storage Systems have been proposed in the 1990s, when it was becoming increasingly clear that sizes of data and their amounts will expand and result in tera- or more byte magnitudes [Fac+05]. While the (then) prevalent architectures of block and file storage are useful to this day, the situation makes the difference. For a brief comparison of the architectures, see 1.
The hardware-close architecture of the **block storage** structures its space into equally large chunks, so-called *blocks*. Files and data get at least a whole block dedicated whether or not it is over or under the size of a block. There is no further information on stored data except for where a coherent part of data/file starts and ends. Being able to access a Storage thusly can have serious safety implications as there are no encapsulations present to manage security more detailed other than either having full or no access. Operating systems usually access their hardware that way usually as the efficiency of such access is fast and low effort, and security can be handled by the OS.
**File storage** is one layer more abstracted from block storage and is used when structuring data hierarchically for example in most form of representation for human readability. In this architecture, data is encapsulated in files, which contain information about format, block storage addresses (the OS still saves data block-wise), permissions, etc. and of course the data itself. Files can be kept in hierarchical trees of folders and they themselves can have certain user-specific permissions. This is usually what is used for most GUI-based storage inspection.
The **object storage** architecture is the most abstract but most convenient for web-based, scalable and large size storage. Object storage saves data as a coherent so-called object, which contain both the actual data and metadata needed to comprehend and use the data. While the service handles the storage in its network, saving it decentralized and with redundancies for robustness, that makes the performance slower than other architectures, but still good for large coherent and cold(ish) data. Objects are stored within buckets which have designated access policies. Usually OSS can be accessed both via API and GUI.

# 2 Services

In this section I will give a brief overview of all of the services I intended to test. These are *Amazon Web Services* (AWS) *Simple Storage Solution* (S3), Google Cloud, CEPH, MinIO and the *European Centre for Medium-Range Weather Forecast* (ECMWF)'s *Fields Database* (FDB).

| | **Provider** | **Target audience** | **Availability** | **Self-Deployment** |
|---|---:|---:|---:|---:|
| AWS S3 | Amazon | Companies | Some free activity | None |
| GoogleCloud | Google | Companies | 300$ free for 90 days | None |
| MinIO | MinIO Inc | Companies, Any Devs | Free/part of AIStor | Yes |
| CEPH | Open-Source | Companies, Any Devs | Free | Yes |
| FDB | ECMWF | In-house Researchers | Free | Theoretically |

Table 1: Overview of different OSS services, red ones are without detailed results.

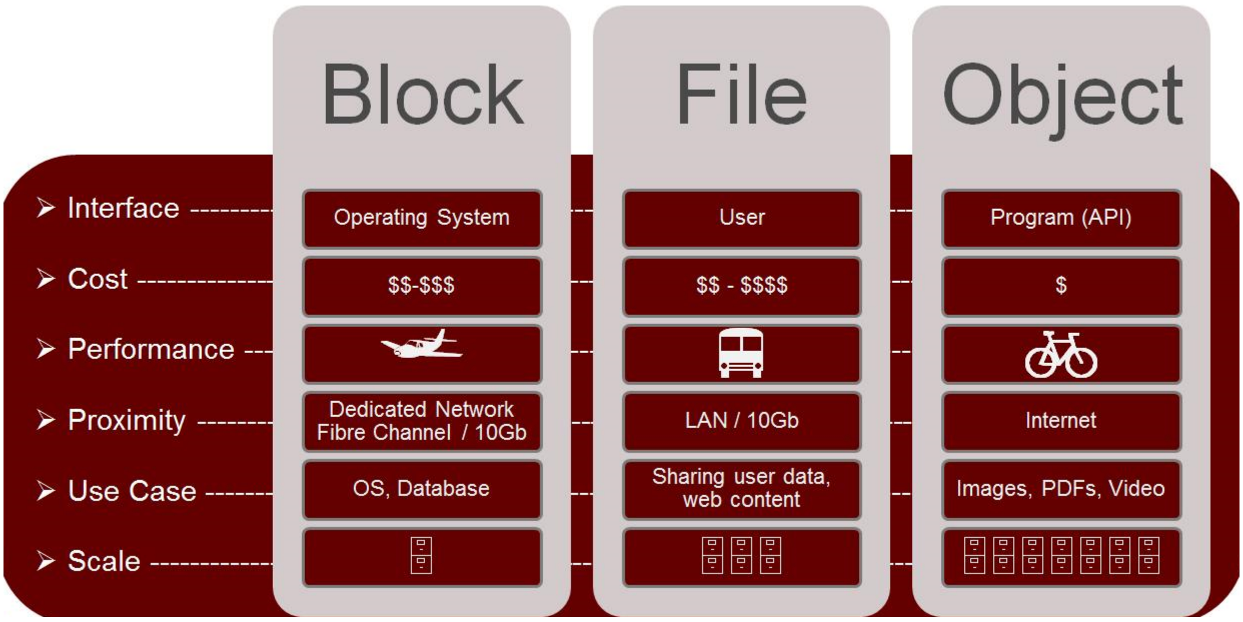

Figure 1: Comparison of the three Storage architectures: Block, File and Object Storage (taken from `https://thecustomizewindows.com/2017/09/object-storage-vs-block-storage/`, last accessed 24.03.2025, 13:00)

**Unfortunately, only MinIO and S3 resulted in fully formed experiments, while the other Services had problems that made it increasingly difficult to deploy/use them.**

**Google Cloud**, though having a possibility for free usage via a 300$ free contingent in the beginning, still needs a creditcard for their data, which I did not own, thus ending the research there prematurely, resulting in no data, and therefore being absent from the results section. Google Cloud has an s3-compatibility (`https://cloud.google.com/storage/docs/interoperability?hl=de`, last accessed 27.03.2025, 13:05)

**CEPH** on the other hand has a documentation about deploying a cluster yourself. Even though it is rather complicated and seems to expect quite some knowledge about clusters, there were some clear cut goals and functioning parts. Unfortunately, self-deployment worked only so far as to build a cluster with a single node, regularly failing to get the cluster into a healthy state. Additionally, any other services using CEPH were also part of products, having no form of free content. CEPH object storage is mostly compatible with s3-interfaces (`https://docs.ceph.com/en/latest/radosgw/`, last accessed, 27.03.2025, 13:03)

**FDB** unfortunately seems to have never been practically planned to be in any way be used by the public. Even though promoted thusly and having all necessary dependencies on github, the installation process did not work out on a Ubuntu device, especially if considering that it was mostly intended to be used on Apple devices. After contacting the ECMWF adn asking for help, which unfortunately did not work out either, there are no useful results from this endeavor.

## 2.1 S3

The AWS Simple Storage Solution is an *Object Storage System* (OSS) meant for company usage, with notable customers being The BMW Group or Netflix (`AWSCustomers`).S3 belongs to the group of AWS. After registering a root user, additional users can be added to the same contingent. The root user can regulate access to buckets via either policies/groups, which other users can be added to. Buckets can be made available via various access points, which themselves have either certain access rights or general rules like either being available to the internet or not, defining a world-region and more.

## 2.2 CEPH

Trying to deploy a CEPH cluster was difficult when considering the official documentation. While bootstrapping a running cluster was indeed possible, the status never was health, as adding in other nodes from other devices I own did not work. Neither troubleshooting nor ChatGPT helped in that regard. Next, I tried deploying the cluster in three virtual Ubuntu server machines on my windows PC, but came no further. Though they had established communication, no more than one node was ever part of the cluster, though the architecture needs three(/four) roles, monitor, manager(two to run smoothly) and object storage demon. The starting node only ever served the monitor and a manager role, making the cluster unhealthy.

## 2.3 MinIO

MinIO is available through different kinds of access. It is part of the product AIStor by MinIO Inc., aimed at companies wanting to use and have access to their own OSS. Additionally, there is the MinIO playground, an instance hosted by MinIO to test out the product. Then there is the possibility to host it oneself by installing and managing the product. The setup is rather simple and requires no extensive knowledge or much overhead.

## 2.4 Fields Database

Developed for in-house use at the ECMWF, the Fields database is used in conjunction with the MARS-Database, while being explicitly optimized for meteorological data in GRIB file format [SQR17]. It is a self-contained project in the sense that it does not have any compatability with other services.

# 3 Results

I tested the available solutions for latency, up and download speeds, user and security management, ease of deployment/use, customization of the service. The speeds were tested using two versions of a hubble space telescope picture of the andromeda galaxy, one 114 megabytes, the other one 1.7 gigabytes large. Originally when considering the FDB for testing as well, there was a GRIB-File to be tested for speed as well, because of the specialization of the FDB for that format. It is a file displaying volumetric soil layers in different depths of soil in an area in north Namibia and south Angola with an extent of

44x36 km, it is 1.5 megabytes large. All runs of the experiment have been run from my private internet access in Göttingen, (Speeds at download: 104 mpbs, upload: 41 mpbs) and the local instance of MinIO I hosted was in that same LAN.

## 3.1 MinIO

The documentation of MinIO, especially for self-deployment on different platforms is well made and rather short, with little prior knowledge or many resources needed. Installing and running the server was easy and light-weight. The GUI of both the MinIO playground and the self-hosted Server are identical, with easy graphical management of users, policies, buckets, up- and downloads. Downloads unfortunately did not work for the playground via API, as even one request threw an error of too many requests.
MinIO allows for extensive user and policy management similar to AWS. Buckets can be access controlled to both follow policy and user restrictions/access. Users can be generated and grouped into policies which decide their accesses in larger quantities of users. Buckets can also be managed to be only accessed by certain users/policies. After inspecting the local storage of the self-deployed instance, I found that the objects were stored in a file/folder hierarchy alike the bucket structure established in the system. Data and Metadata are stored as files, data as several parts and neither of them are human-readable.

## 3.2 FDB

After deciding to incorporate this system into this experiment after reading [SQR17], I tried using the system myself. Finding several repositories that are open to public, including all the mentioned dependencies needed for the FDB. Since most of the installations went smoothly and errors could either be read and solved well or even with the help of ChatGPT, the FDB was installed on my personal Ubuntu machine, but still it did not work out. There were both commands missing as well as a missing support for GRIB-files, which I explicitly downloaded from the ERA5-Product (ECMWF Reanalysis Version 5) series for this experiment. After opening a ticket for the missing and broken GRIB-File support, the ECMWF provided me with a slightly adapted installation guide, different from the original one, which did not solve the problems either. Since then, after spending a handful working days on that topic, I made no further efforts running the FDB. For some more details, see (`https://jira.ecmwf.int/servicedesk/customer/portal/4/SD-104679?sda_source=notification-email`, account needed to access, last accessed 25.03.2025 15:38)

## 3.3 S3

After registering an account, one has access to many AWS solutions, one of them the S3. In the AWS console, any regular bucket and access-management operations are possible. Creating buckets, up- and downloading objects as well as managing users, accesses and policies. Buckets are not accessible after creation, access points have to be created with explicit access rules, even not being able to be reached via internet, adding further to security. Interestingly enough, any AWS-account can be invited to join the "company" established by the root AWS user, making collaboration easier. API-wise, the documentation of boto3 - the python library supporting s3-access - is acceptable. Classes and objects needed for initial access are not immediately clear, the given examples usually drive successful applications. The code itself also is light-weight when considering lines, making it more easy to understand.

# 4 Comparison

| **Region** | **Product** | **Operation** | **Time** |
|---|---|---|---|
| eu-central-1 | MinIO | Create Bucket | 1.111 |
| eu-central-1 | MinIO | upload 1.7gb | 504.018 |
| eu-central-1 | MinIO | upload 114mb | 47.066 |
| sa-east-1 | MinIO | Create Bucket | 1.096 |
| sa-east-1 | MinIO | upload 1.7gb | 483.46 |
| sa-east-1 | MinIO | upload 114mb | 47.857 |
| local | MinIO | Create Bucket | 1.096 |
| local | MinIO | downl 1.7gb | 63.967 |
| local | MinIO | downl 114mb | 4.384 |
| local | MinIO | upload 1.7gb | 85.107 |
| local | MinIO | upload 114mb | 6.056 |
| eu-central-1 | AWS S3 | downl 1.7gb | 149.360 |
| eu-central-1 | AWS S3 | downl 114mb | 10.311 |
| eu-central-1 | AWS S3 | upload 1.7gb | 350.062 |
| eu-central-1 | AWS S3 | upload 114mb | 24.777 |
| sa-east-1 | AWS S3 | downl 1.7gb | 160.198 |
| sa-east-1 | AWS S3 | downl 114mb | 16.933 |
| sa-east-1 | AWS S3 | upload 1.7gb | 648.254 |
| sa-east-1 | AWS S3 | upload 114mb | 62.099 |

Table 2: Results of testing services via boto3. Operations tested were creating a bucket, up- and download of two files (mentioned in beginning of section 3). Time in seconds, averaged over different amounts of runs. Operations not displayed usually failed for some or other reason.

*Interestingly enough, though finding a difference in latency in the s3 regions, there seemed to be none when switching the region in the open playground of MinIO, which suggests that it made no difference (avg. latencies in seconds: play.min.io : 0.572; AWS eu-central-1: 0.588; AWS sa-east-1: 1.164)*

| MinIO | | | AWS S3 |
|---|---|---|---|
| + | - | + | - |
| Easy to use | not as fast as s3 | good speed | complicated GUI |
| Easy to deploy | not quite as many functions as S3 | many, many functions... | ...maybe too many in the beginning |
| Self-Configurable | | Free Trial, but... | no chance of not paying afterwards |
| There is both a free version as well as a supported version as part of product | | Huge Community / Support | |
| S3-compatible | | Established big player (all others look for S3-Compatibility) | |
| Good security controls | | Even better security controls with e.g. access points | |

Table 3: Comparing the two in-depth analyzed solutions, MinIO & AWS S3

MinIO and S3 have different target audiences and it shows. MinIO supports self-deployment as a free source for any sort of developer or company, while S3 is definitely suited for larger companies with a well-made GUI for the AWS-console and way more control over user-management, access control, further services, etc.

## 5 Conclusion

It seems that both MinIO and S3 fill their niches. S3 is optimized for large companies and fulfills that role well, it has both API and GUI access that are well-made, well-maintained and well documented. It seems to be the gold standard when it comes to OSS-services. Other storages usually advertise with the fact that they are compatible with S3-API-libraries, which does indeed speak for itself. S3 has the best speed and way more functions than other services, as well as having other AWS-services attached to it. So for companies that do not necessarily have to do with a lot of technical expertise or wants to have the best solution on the market with good speed and resilience, S3 seems to be the best choice. MinIO falls in between product and free software as it offers both. While only the free version was tested, it seems as the use case looks more for companies that are not quite as large but still need petabyte-scale storages. The fact that MinIO can be deployed by oneself speaks for usage in academical and other non-economically-focused environment, where a team of people can handle such a cluster well. Same goes for the CEPH product, but unfortunately, as it did not work out for me, unknown in quality. The distinction seems clear: S3 for large companies, stable, fast, many functions, good security
MinIO for smaller companies/academic environments.

# References

[Fac+05] M. Factor et al. “Object storage: the future building block for storage systems”. In: *2005 IEEE International Symposium on Mass Storage Systems and Technology*. 2005, pp. 119–123. DOI: 10.1109/LGDI.2005.1612479.

[Mon+24] Anindita Sarkar Mondal et al. *Demystifying Object-based Big Data Storage Systems*. 2024. arXiv: 2406.00550 [cs.DB]. URL: https://arxiv.org/abs/2406.00550.

[SQR17] Simon D. Smart, Tiago Quintino, and Baudouin Raoult. “A Scalable Object Store for Meteorological and Climate Data”. In: *Proceedings of the Platform for Advanced Scientific Computing Conference*. PASC ’17. Lugano, Switzerland: Association for Computing Machinery, 2017. ISBN: 9781450350624. DOI: 10.1145/3093172.3093238. URL: https://doi.org/10.1145/3093172.3093238.

[SQR19] Simon D. Smart, Tiago Quintino, and Baudouin Raoult. “A High-Performance Distributed Object-Store for Exascale Numerical Weather Prediction and Climate”. In: *Proceedings of the Platform for Advanced Scientific Computing Conference*. PASC ’19. Zurich, Switzerland: Association for Computing Machinery, 2019. ISBN: 9781450367707. DOI: 10.1145/3324989.3325726. URL: https://doi.org/10.1145/3324989.3325726.

# A Code samples

```python
import boto3, time, requests
from botocore.config import Config
# MinIO Playground credentials
access_key = "CHANGE"
secret_key = "CHANGE"
endpoint_url = "https://play.min.io"
config = Config(retries={'max_attempts': 1})
# Initialize S3 client
s3 = boto3.client(
    's3',
    aws_access_key_id=access_key, aws_secret_access_key=secret_key,
    endpoint_url=endpoint_url, region_name='eu-central-1',
    config=config)
bucketNames = []
bucketNames.append((bucket_name_eu, s3))
bucketNames.append((aws_bucket_eu, aws))
bucketNames.append((bucket_name_local, localMin))
for bucketName, client in bucketNames:
    #Create#==============================
    start_time = time.time()
    client.create_bucket(Bucket=bucketName)
    end_time = time.time()
    #UPLOAD#============================
    start_time = time.time()
    client.upload_file(
        'andromeda 10k.tif', bucketName, 'andromeda 10k.tif')
    end_time = time.time()
    #DOWNLOAD#==========================
    start_time = time.time()
    client.download_file(
        bucketName, 'andromeda 10k.tif', 'andromeda 10k.tif')
    end_time = time.time()
```

Listing 1: Boto3 python-pseudo-code, shortened for understanding