# Cloud-Based File Storage System

Barrasso Marco

September 24, 2025

## 1 Introduction

This project centers around the creation and implementation of a cloud-based file storage system, with a specific focus on utilizing Nextcloud as the selected platform for the system. The primary objectives include user-friendly features such as file upload, download, and deletion, while ensuring each user has a dedicated private storage space. Critical considerations involve user authentication, file operations, scalability, security measures, and cost-efficiency.

To make the setup repeatable, we run everything with Docker Compose on a local machine (MacBook Air M1). The documents show how the parts fit together, what each one does, and what we did for security and future growth. All Docker files and code are in the GitHub repository, along with a short guide so you can deploy and try it yourself.

## 2 Creating and Deploying the System

### 2.1 Architecture Overview

We package the service as a small set of containers orchestrated with Docker Compose. The core components are:

- **Nextcloud:** the web application and WebDAV API that handle user authentication, file upload/download/delete, sharing, and the browser UI.
- **PostgreSQL:** the relational database that stores metadata (users, groups, shares, file index, app state). File contents remain on the filesystem, so the database stays comparatively small.
- **Redis:** an in-memory data structure server that is used primarily as an application cache or quick-response database.

User files are persisted on a dedicated data volume mounted into the Nextcloud container; database files live on a separate DB volume. The services communicate over a private bridge network; only the Nextcloud HTTP port is published. In our setup, the application is reachable at:

`http://localhost:8080`

which maps host port 8080 to the container's internal web server. To deploy this file, it must be saved as docker-compose.yml and it is enough to simply execute the following command in the terminal within the directory containing the file:

```
docker-compose up -d
```

This command will pull the required images, create containers based on the defined services, and deploy the Nextcloud file system with its associated database.

## 3 Key Features

In this section, the features required by the exercise are discussed as implemented by the Nextcloud platform.

- **User Authentication:** Nextcloud provides a secure user authentication system, allowing users to sign up, log in, and log out. Different measures can be taken to improve the security of the file system, increase password length and require special characters, enable two-factor authentication, as well as encryption on stored data. This option comes with a performance cost, and should therefore only be used when it is really needed.
- **User Authorization:** Users can be easily added and deleted, together with their data. Different user roles, such as regular users and admins, are supported for effective user management. Admins have the ability to manage and delete users, and are the only ones who can modify the overall settings of the system.
- **File Upload:** Users can easily upload files to their private storage space.
- **File Download:** Users have the possibility to download files from their private storage.
- **File Deletion:** Users can also delete files from their private storage space.
- **Private Storage Space:** Regular users are allocated their private storage space to ensure data privacy. Different storage limits can be imposed to avoid any particular user from exploiting the service and preserve storage.

## 4 Monitoring and Testing

In this section I will discuss how i monitored the performance of the system using Grafana and tested it with Curl and Locust.

### 4.1 Monitoring with Grafana

To monitor performance, we run three extra containers: Grafana (a visualization tool), Prometheus (which acts as Grafana's data source), and Nextcloud Exporter (which works as a bridge between Nextcloud and Prometheus). After bringing them up with Docker Compose, open Grafana at `http://localhost:3000`, log in, and add a Prometheus data source. Set the Prometheus server to `http://localhost:9090` and make sure everything is up. Once the data source is saved and healthy, create your dashboard in the Grafana UI or import an existing `.json` dashboard file.

I found a pretty nice dashboard on the Grafana official web site and I decided to import that one. You can find it in the `grafana_dashboard.json` file in the GitHub repository. When the dashboard is connected you can see various metrics, this is how the one I imported looks:

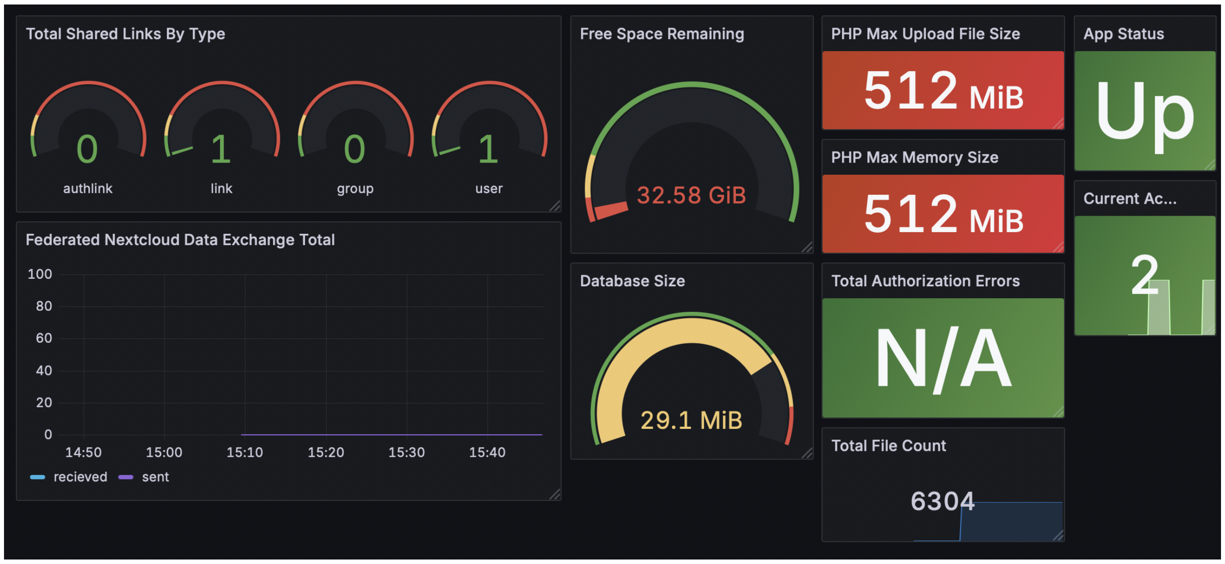

Figure 1: Grafana Dashboard

## 5 Performance Evaluation with Curl

`curl` ("Client URL") is a command-line tool for transferring data with URLs. It supports many protocols and options, making it a versatile choice for downloading/uploading files, sending requests to web services, and testing API endpoints.

To assess the performance of the file storage system, we created files of different sizes using the `dd` command. We then uploaded these files to, and downloaded them from, the system while measuring the time to complete each operation. The script used for these tests is provided in `upload_and_download.sh`.

The following plots report the elapsed time for download and upload as a function of the file size.

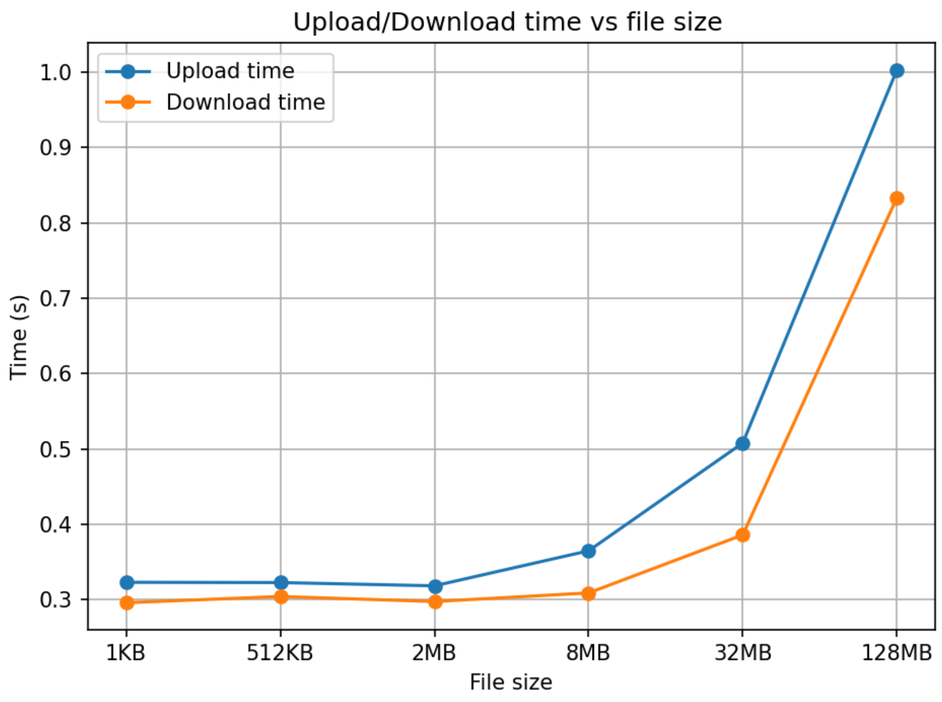

Figure 2: Time vs File Size

## 6 Load Testing with Locust

`Locust` is an open-source, Python-based load testing tool for simulating large numbers of concurrent users and measuring the scalability and performance of web applications. Test scenarios are defined in Python and emulate realistic user interactions with the target system.

For my experiments on Nextcloud, I created 30 test users by running the `add_nextcloud_users.sh` script. To permit traffic generated by the Locust client to reach Nextcloud, we firstly need to add the locust container hostname to the `trusted domains`.

I implemented a set of simple tasks in `tasks.py` to exercise common operations:

- create files of varying sizes;
- read the contents of an existing user file;
- upload a text file;
- list all files owned by the user.

I include two charts from runs on my MacBook Air M1, spawning 10 and 30 concurrent users, respectively. For each task we record request latency and requests per second.

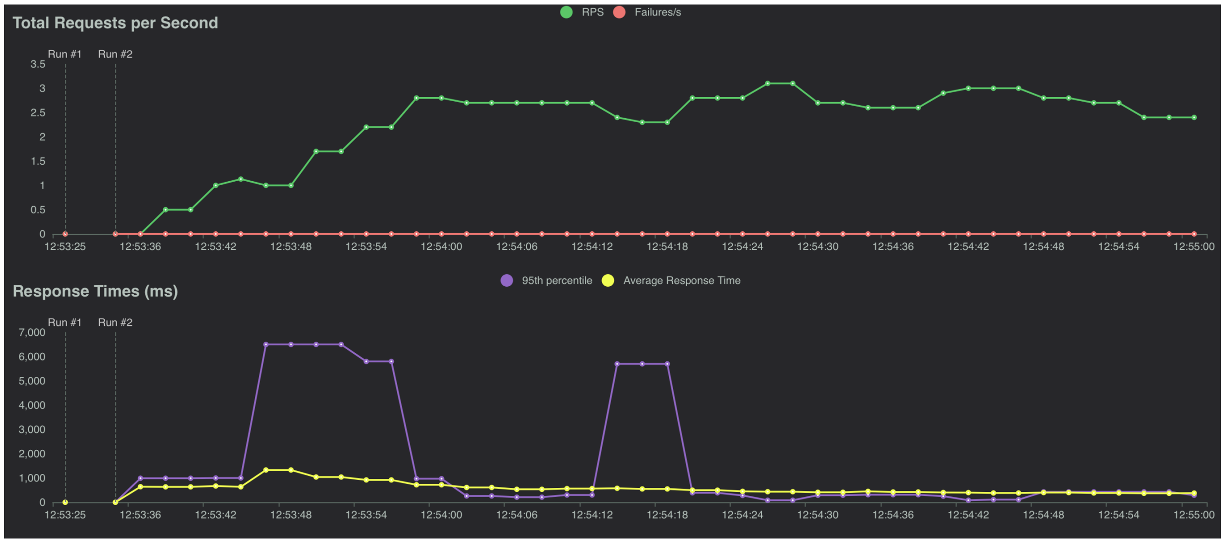

Figure 3: Locust test with 10 users

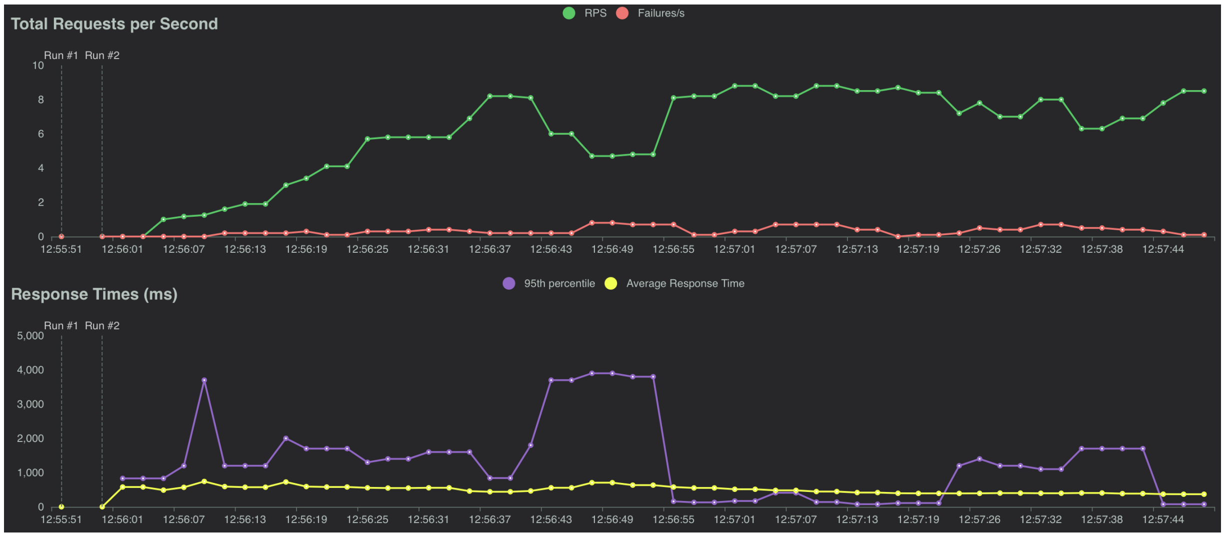

Figure 4: Locust test with 30 users

While the response time does not show significant differences between the tests performed with 10 and 30 users, one thing to be noticed is that some requests fail in the last scenario. More tests would be needed to pinpoint the reason behind this phenomenon.

# 7 Scalability

This section looks at two ways to handle more load: (1) running on your own cluster, and (2) using cloud services. We list the main pros and cons and when each makes sense.

## 7.1 On-cluster Deployment

Cluster deployment involves creating a network of interconnected servers to distribute and manage the file system. This approach strengthens scalability and fault tolerance, but it demands significant setup complexity and ongoing maintenance efforts. While it provides high availability and promotes independence, it may be more suitable for enterprises with substantial infrastructure investments who already possess a cluster.

### 7.1.1 Advantages

- **Scales by adding nodes:** Add servers as you need more space or speed.
- **Built-in redundancy:** Data is copied across nodes, so it is more resilient.
- **Security and control:** You decide where data lives and how it is protected.
- **Full ownership:** You control hardware, network, and software.

### 7.1.2 Disadvantages

- **Complex to run:** Needs skills in distributed systems and monitoring.
- **Costs:** Up-front hardware costs and ongoing maintenance.
- **Slower to change:** Adding capacity takes time (buying and installing gear).

## 7.2 Cloud Services

Leveraging cloud services like AWS S3 allows organizations to scale storage seamlessly, paying only for the resources they consume (pay as you go services). One key feature of such services is autoscaling, the ability to adjust capacity automatically to match demand.

### 7.2.1 Advantages

- **Elastic scaling:** Increase or decrease capacity instantly.
- **Global access:** Easy access from many places.
- **Pay-as-you-go:** Lower up-front costs; pay for actual usage.
- **High durability:** Built-in data protection and lifecycle tools.

### 7.2.2 Disadvantages

- **Data rules:** Laws may limit where data can be stored.
- **Costs can add up:** Network egress and per-request fees can be high.
- **Vendor lock-in:** You depend on one provider's tools and prices.
- **Variable performance:** Shared systems can add latency unless you pay more.
- **Data Privacy:** Storing sensitive data in the cloud may raise concerns related to data privacy.

## 7.3 Considerations

When choosing between the two, consider organizational requirements, budget, scalability needs, and data security. On-cluster deployments have higher up-front infrastructure costs, while operating expenses depend on the specifics of the environment. So, as we can see, there is no single best choice.