Data Science and Artificial Intelligence

May 2024

University of Trieste

Cloud Project

Cloud-Based Storage System

Andrea Gottardi
SM380003

Contents

1 Introduction

This project aims to address the need for a cloud-based file storage system that enables to seamlessly upload, download and delete files, while ensuring the privacy of individual storage spaces. We selected Nextcloud for its ease of deployment using Docker containers and useful features. In fact, the availability of a Docker container for Nextcloud simplifies the deployment.

2 User Authentication and Authorization

This section of the report is designed to provide an overview over the authentication and authorization aspects of the user system. Nextcloud facilitates user authentication and authorization with its features:

- **Registration**: Nextcloud provides built-it features that enables seamless user's registration, login and logout. Administrators can abilitate the registration application from the Applications page.
- **Roles Management**: Nextcloud supports differentiation between regular users and admins. Through the available role-based access control we can define different suitable levels of privileges.
- **Private Storage Space**: Nextcloud assigns, by default, a limited private storage space. Administrators can adjust the private storage quotas via the **Users** page.

File Operations Nextcloud provides users with the most common file management operations, such as:

- file upload
- file download
- file deletion

from private storage space.

3 Security

In this part of the report we address user security. The **security** and **authentication** features by Nextcloud are varied. One of the most basic security features is implementing a **password policy**: the admin can define it through the security page of the administration settings. In our simplified scenario, we employed a very lenient policy, allowing for trivial passwords, in order to facilitate the testing phase.

Additional measures In a more complex environment, it would be useful to take into consideration the security features offered by Nextcloud. A vital step is to enable server-side encryption, which ensures that the data stored within files remains inaccessible to unauthorized parties. Furthermore, strengthening user authentication with two-factor authentication can increase security and minimize the risk of unauthorized access.

HTTPS protocol An industry standard is the use of the HTTPS protocol instead of the HTTP for client-server communication. This precaution has a crucial role in preventing man in the middle and other type of attacks. This solution was not implemented in the project, but it can be realized by obtaining a certificate and exploiting the 443 port instead of the 80.

4 Deployment

The deployment is managed through Docker and docker-compose.

4.1 Docker compose file

The relative docker-compose.yml file defines a series of containers/services and the network that connects them. The contaneir present in the file are:

- **nextcloud-db**: This is the MySQL database container that stores the Nextcloud data. It contains the root username and password. It has a volume mounted to store the database data, placed in the relative host directory.
- **redis**: The Redis container is used to create a caching mechanism for Nextcloud.

- **nextcloud instances**: This containers are the two identical Nextcloud instances. They operate on the same data volume but are indepedendent and have different configuration files. They depend on the nextcloud-db database and he redis containers.
- **locust**: The Locust container is used to test performance under stress and will be used to generate load on the Nextcloud instances.
- **nginx**: It is primarily used to route requests to the Nextcloud instances and for load balancing. It depends on the nextcloud_instance and nextcloud_instance1 containers, for which it provides the access interface and the load balancing.

At the end of the docker-compose file is defined the **network** that allows comunication between containers. All the containers are configured to be part of the nextcloud-network.

4.2 Execution

To run the docker containers, it is sufficient to navigate to the folder in which the file docker-compose.yml is stored and run

```
docker compose up -d
```

while to stop them we run

```
docker compose down.
```

The root user can then access the cloud system as an administrator through the URLs **http://localhost:8080** or **http://localhost:8081**. To test user interactions we created thirty fake users using the powershell script **CreateTestUsers.ps1**. Then it is possible to run the locust stress test and observe the performance metrics.

5 Testing with locust

To conduct load testing, we generated 30 users (with the CreateTestUsers.ps1 script) and deployed a locust container that utilizes a Python script stored in a specific directory on the host machine. This script defines tasks for the locust instance, including user authentication and file operations. The testing involves uploading and subsequently deleting files of various sizes.

Users can upload files of different sizes: 1 KB, 1 MB, and 1 GB. To assess the application's performance under different stress levels, we conducted tests initially allowing uploads of files of any size, followed by sessions restricted to exclusively small, medium, and large files. We evaluate these four scenarios and analyze the resulting graphs produced by Locust. These visualizations offer insights into metrics such as requests per second, failure rates, response times, and user counts.

5.1 Testing with different file sizes

Various sizes With files of varying sizes, we observe the following trends from the plot in Figure 1:

- An initial spike in failures occurs with an increase in the request rate, which later stabilizes at 1.5 requests per second, accompanied by a 25% failure rate across all requests.
- Response time averages around 55000ms and fluctuates with changes in requests per second.
- A performance threshold appears evident in terms of requests per second.

Small size The system efficiently handles 1KB files as evidenced by a 0% failure rate and drastically improved performance relative to larger files, as illustrated in Figure 2:

- The system processes an average of 15 requests per second.
- Response time remains stable and progressively decreases to about 450ms.

Medium size The behavior observed with 1MB files largely mirrors that seen with 1KB files, though with marginally reduced performance. The failure rate remains negligible, and details are shown in Figure 3:

Figure 1: All files

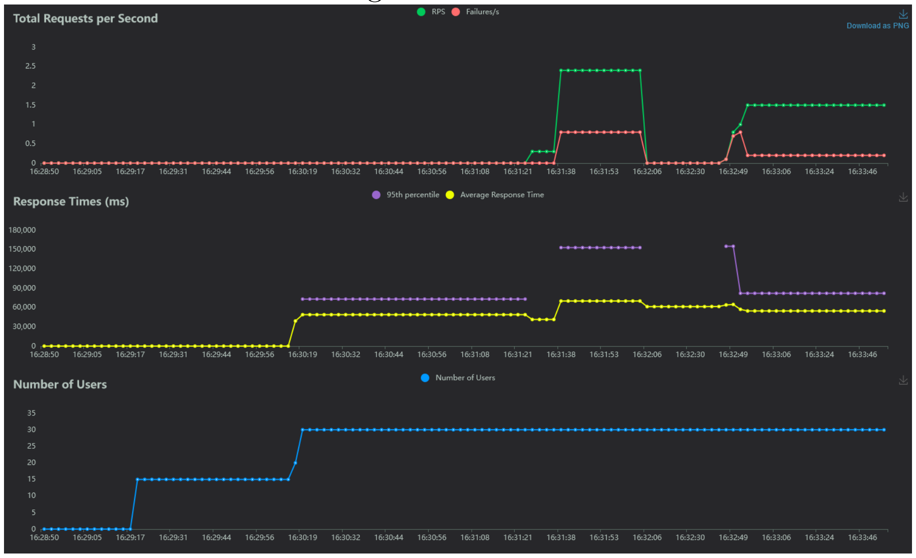
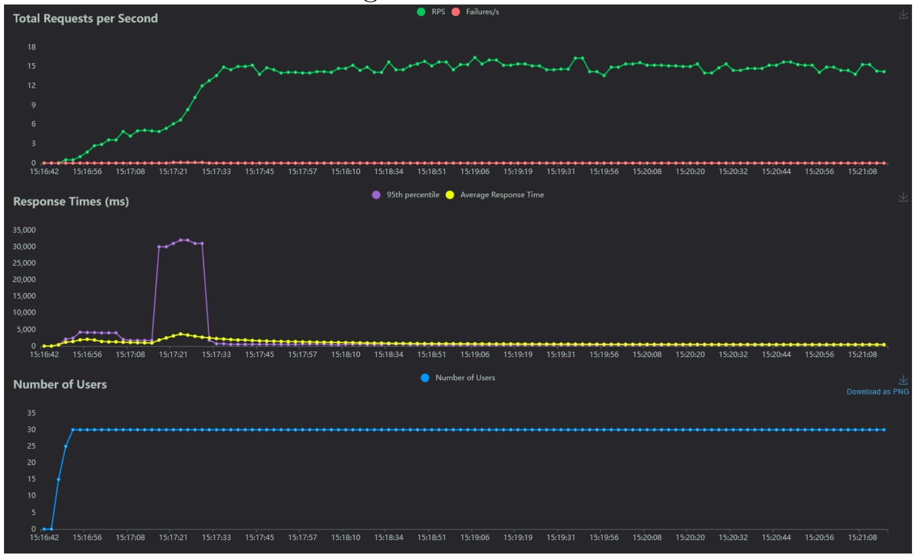


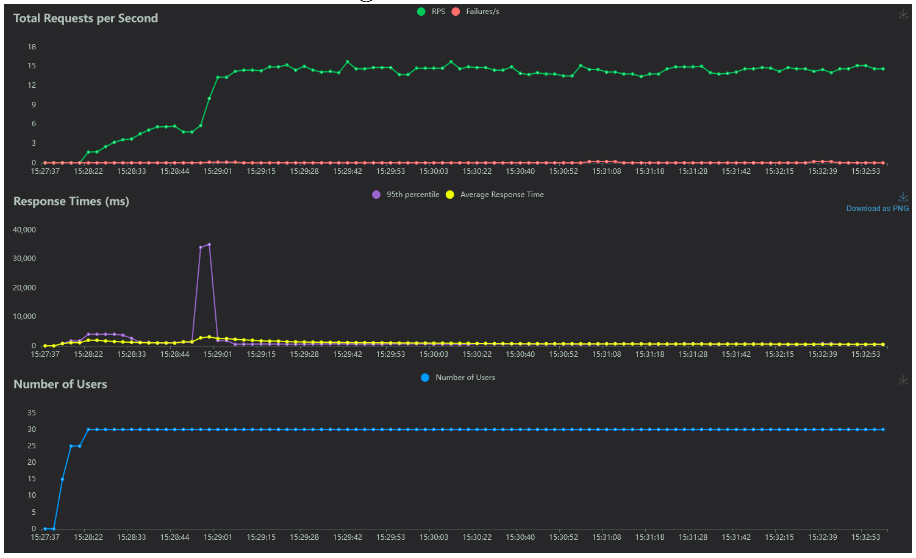
Figure 2: 1kb files



- The system processes an average of 14.5 requests per second, closely matching the rate for smaller files.
- Response time remains relatively stable, with a slight increase to about

550ms.

Figure 3: 1mb files



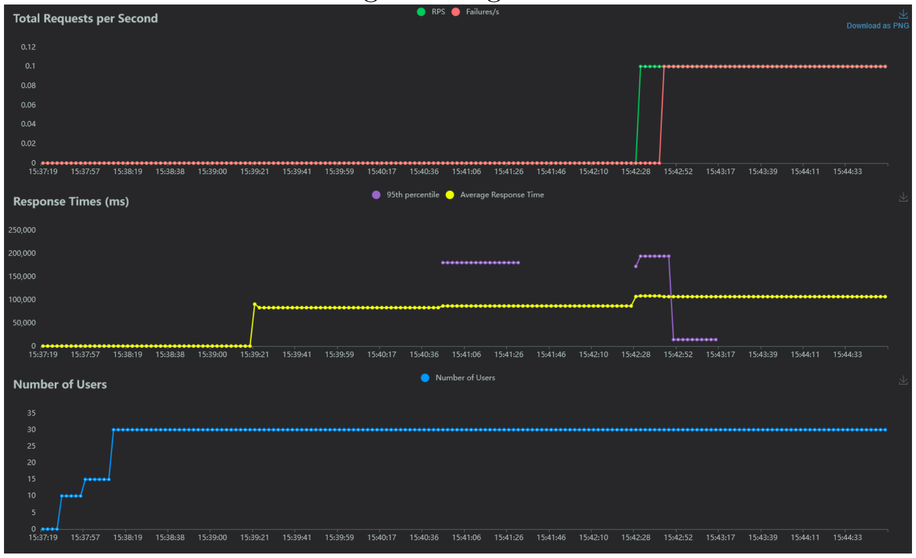
Large size Stability deteriorates significantly when restricting uploads to 1GB files, with a significant increase in failures and response times, as depicted in Figure 4:

- Approximately 87% of requests fail.
- The rate of requests per second drops to 0.1.
- Average response time raise to 106660ms.

6 Scalability

The use of **redis** and **nginx** allows for a greater scalability of the system.

Figure 4: 1gb files



6.1 Caching with Redis

Redis enhances Nextcloud's capabilities by implementing a caching mechanism. This mechanism involves temporarily storing frequently accessed data in a high-speed storage system. The most recently or frequently used data is kept in a readily accessible location to expedite future requests. Redis introduces two significant advantages:

1. **Improved System Responsiveness:** By reducing the time taken to retrieve frequently accessed data, Redis speeds up the overall response time of the system.
2. **Rapid Data Access for High Demand:** Even under heavy loads or when serving numerous users, Redis ensures quick data retrieval.

These features significantly enhance the user experience by providing faster access and reduced latency, thereby improving the scalability and efficiency of the system.

6.2 Load balancing and request routing

Nginx employs a load balancing mechanism that distributes incoming queries between the two applications: its configuration is managed through the **ng-**

inx.conf file and determines the load balancing algorithm. This is achieved by setting up an upstream module named nextcloud_backend, which includes two server instances:

- nextcloud_app1:80
- nextcloud_app2:80.

Nginx is configured to use the least_conn directive, which routes traffic to the server currently handling the fewest active connections. This strategy not only enhances the scalability of the system by optimizing resource utilization and balancing the load more evenly but also improves response times and system reliability by avoiding overloading any single server. The configuration ensures a more effective distribution of network traffic, making it highly suitable for environments that demand high availability and performance.

6.3 Alternatives

To enhance our system's capacity to manage a higher volume of requests, we can leverage Nginx and Redis alongside additional strategies. Nginx and Docker enable us to seamlessly integrate more applications into our system. By increasing the number of Nextcloud instances and implementing an auto-scaling mechanism, we can dynamically adjust the number of active server instances in response to fluctuating loads.

7 Conclusion

In this project, we developed and implemented a cloud-based storage system leveraging Docker and Nextcloud. Nextcloud equipped us with a robust suite of tools and features to efficiently manage common storage operations. Simultaneously, Docker facilitated the underlying infrastructure, enabling seamless connectivity and integration of the necessary components. This combination enhanced the scalability and flexibility of our storage solution. Following the system's deployment, we conducted performance tests using Locust and observed that the system handles smaller and medium-sized files effectively but encounters challenges when processing larger files. The results achieved are satisfactory; however, there is space for improvement to handle a larger volume of requests and more substantial files more efficiently.