Cloud-Based File Storage System Deployment Report

Roshanak Behrouz
Data Science and Artificial Intelligence

Introduction

This report outlines the deployment and testing of a cloud-based file storage system using Nextcloud, a popular open-source solution. The system is designed to manage user authentication and authorization, handle file operations, ensure scalability and security, and optimize cost-efficiency.

System Requirements and Implementation

1. User Authentication and Authorization:

- **Sign Up, Log In, and Log Out:** Users can easily create an account, access their private space after logging in, and securely log out. To address the requirements of the project I enabled registration of users from apps in Nextcloud so that user can sign-up.

- **User Roles:** I create one role of Users for regular users and assign each user to defined role. Regular users have access to their files and cannot view or edit other users' data. Admins can manage user accounts and access control.

- **Private Storage Space:** I manage to dedicate every user private storage space. Admins have the ability to manage the users.

2. File Operations:

- **Uploading Files:** Users can upload files to their designated private storage space through an intuitive interface.

- **Downloading Files:** Users can download their stored files at any time, provided they have adequate permissions and authentication.

- **Deleting Files:** Users can manage their storage by deleting files, with changes immediately reflecting in their available storage quota.

3. Scalability:

- The dual-instance architecture of Nextcloud with Nginx load balancing ensures the system can scale horizontally to accommodate increased load.

- MariaDB's separate instances reduce bottlenecks and allow for distribution of database queries.

- Load balancing was implemented using Nginx to distribute traffic evenly across multiple Nextcloud instances, ensuring optimal resource utilization and user experience.

4. Security:

- **Secure File Storage and Transmission:** Nextcloud comes with a set of secure defaults to address the security of the system. In order to implement secure file storage, if the user data is sensible, Nextcloud allows to enable file encryption on the server side. This will reduce the performance of the system but will prevent an intruder that gains access to the data to read it. When file encryption is enabled, all files can still be shared by a user using the Nextcloud interface but won't be sharable directly from the remote server. Note that enabling encryption also increases the amount of storage space required by each file. To enable encryption from the web interface simply login as admin, search for the Default Encryption module app and enable it. Then to enable file encryption: Administration Settings -> Administration Security -> Server-side encryption.

- **User Authentication Security:** Robust authentication mechanisms, including strong password policies like using numbers and symbols and require to change password periodically and possible two-factor authentication, were recommended to enhance security. I would do the Nextcloud Security Scan https://scan.nextcloud.com to check for unknown vulnerabilities of my system. I would regularly look at the Activity and Logging sections of the admin page for any suspicious or unauthorized access patterns.

- **Monitoring**: A basic tool is the Usage Survey report of Nextlcoud. First of all, I would regularly keep track of the users, the data storage and the activities. It is important to note that, because of the pay-as-I-go policy, the more resources I allocate, the larger would be the fee. A way to manage the system could be setting a maximum storage space for every user, and creating various groups of users like base-premium to differentiate the storage space availability.

- **System Updates**: Scheduled updates and backups are critical for maintaining system security and data integrity, ensuring the platform remains resilient against emerging threats and data loss scenarios.

5. Cost-Efficiency:

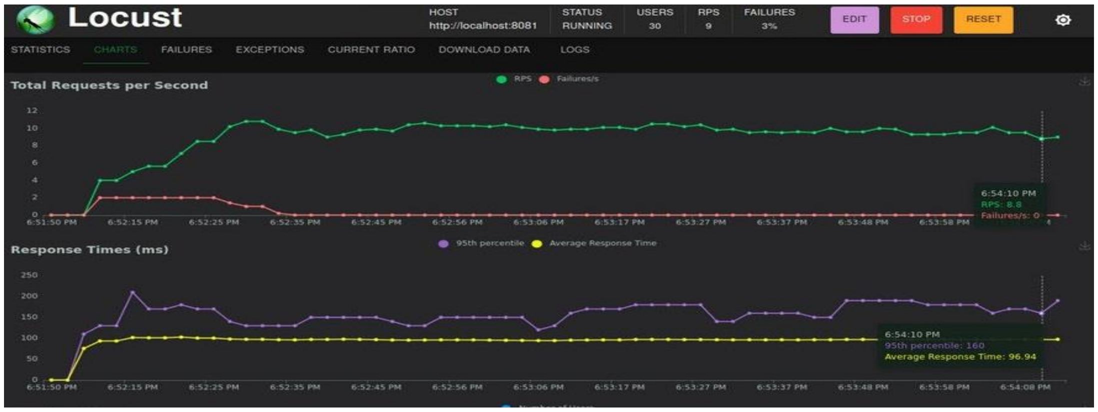
- The deployment utilized open-source technologies, significantly reducing software licensing costs.

- Recommendations for future deployments include choosing cost-effective cloud storage and compute resources, leveraging reserved instances, and implementing auto-scaling to optimize resource utilization and costs.

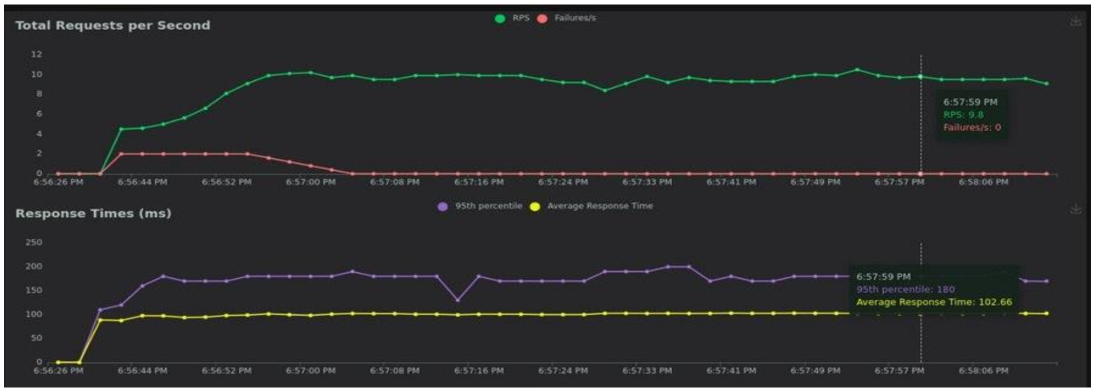
6. Testing Infrastructure:

- Load testing was performed using Locust to simulate multiple users interacting with the system, providing insights into the system's handling of various operations under stress. The test highlighted the system's robustness in managing concurrent file downloads, showcasing its readiness for real-world usage. Results from locust for load testing, as mentioned before, this test is done for three different file sizes. Here are the results:

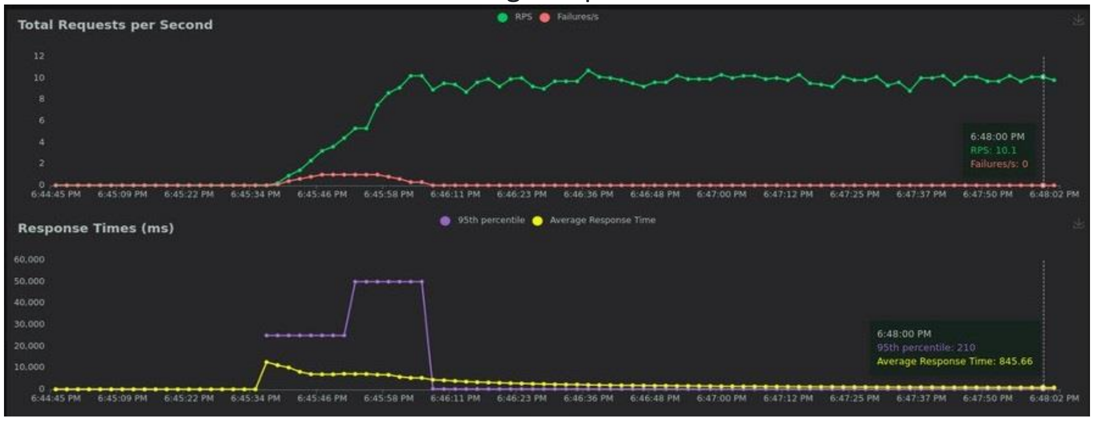
- 10KB file:



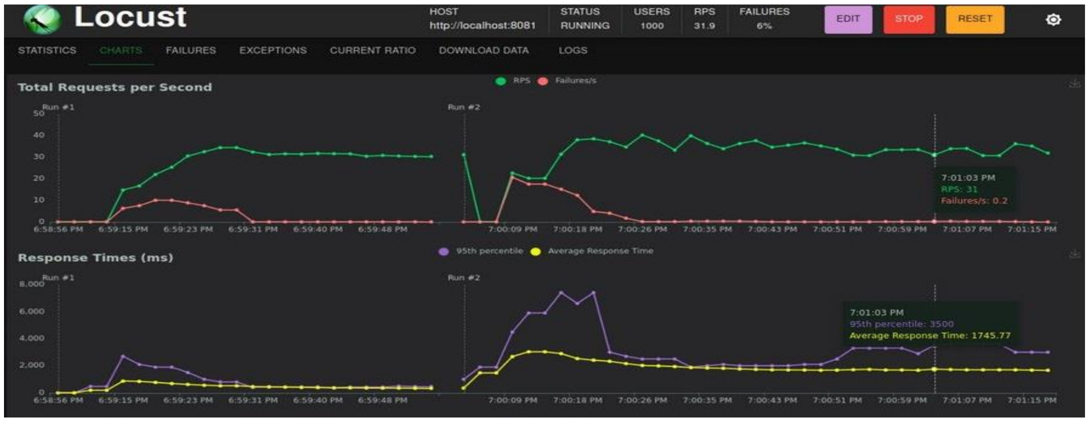
- 1MB file:



- 1GB file: As the size of file increased average response increases.



- Comparing: I increased the number of users and as you see Average response time increased



7. Deployment:

The deployment, executed on a laptop environment (My Laptop is Asus with 12 GB of Ram and a Corei5 CPU with Ubuntu OS (dual boot)).

Reason of choices:

- Nextcloud is a collaboration platform focused on file sharing, synchronization, and productivity, while MinIO is an object storage server optimized for scalable storage of unstructured data. Locust is Python based, open source, developer friendly and easy to use. Nginx is Low resource usage, active community users in case of trouble shooting.

Docker and Docker Compose:

Docker: I used Docker to containerize individual components of my file storage system. This approach isolated dependencies and ensured consistent environments across different stages of development and deployment.

Docker Compose: With Docker Compose, I defined and ran multi-container Docker applications. I created a docker-compose file to configure my application's services, networks, and volumes, which included the Nextcloud instances, database servers, and the Nginx reverse proxy. I made a directory named Nextcloud and saved my docker-compose file and other necessary files there. I used Docker Compose to define and run multiple Docker containers for my Nextcloud and database instances. I edited docker compose file to address my needs to set up a Nginx load balancer. In order to setup a load balancer I need two or more instances of Nextcloud. I created one more instance of Nextcloud to do the load balancing. Then I need to define Database configuration. Two database instances were defined in the Docker Compose file (db1 and db2) to provide separate databases for each Nextcloud instance. Environment variables like root password, database name, user, and user password were configured for each database service. Two Nextcloud instances (app1 and app2) were configured, each connected to its respective database. Environment variables were set up to ensure each Nextcloud instance connects to the correct database. Unique ports were exposed for each Nextcloud instance for direct access and testing (8081 and 8082). A NGINX service was defined in the docker-compose file to act as a load balancer. The NGINX configuration file (nginx.conf) was customized to distribute incoming requests between the two Nextcloud instances using the upstream directive. The load balancer was configured to listen on a common port (80) and proxy requests to the two Nextcloud instances. NGINX was set to depend on the Nextcloud instances to ensure it starts after they are up and running. I used "docker-compose up" to start all the services defined in the Docker Compose file. This command initializes the databases, Nextcloud instances, and the NGINX load balancer. After the services were up, I accessed the Nextcloud instances directly via their exposed ports (8081 and 8082) to verify their functionality. The NGINX load balancer was accessed using the host's IP address which then distributed the traffic between the two Nextcloud instances.

MariaDB: Used as the database backend for Nextcloud, storing user data, metadata, and application settings. Separate database instances support each Nextcloud instance, ensuring data isolation and improved performance.

Nginx: Nginx was set up as a reverse proxy to distribute incoming user requests efficiently across the two Nextcloud instances, enabling load balancing and providing better resource utilization and fault tolerance.

Configuration: I created a Nginx configuration file specifying the upstream servers (Nextcloud instances, I have only 2 instances) and the routing logic. Nginx listened on port 80 and proxied the requests to the appropriate Nextcloud instance based on the defined load-balancing strategy. I used 8081,8082 for nextcloud instance1 and 2 respectively (I started docker-compose with only one Nextcloud instance to check if it works, then expanded to two next cloud instances.)

Locust for Load Testing:

Load Testing: Locust was used to simulate user behavior and test the scalability and performance of my Nextcloud deployment. I defined user tasks in a Python script to mimic file download operations from Nextcloud. Locust is an open-source load testing tool that allows us to simulate millions of simultaneous users to test the performance and scalability of our web applications. In order to see how my system behaves under heavy load, which is crucial for ensuring that my cloud-based file storage system can handle the expected user traffic without performance degradation. First, I defined 30 users with bash and saved them into a file. http://localhost:8089 address defined to use Locust.

Testing Strategy: I configured Locust to generate a specified number of virtual users to perform downloads of special size files I created, testing the system's response to concurrent user actions and monitoring performance metrics like request failure rates and response times. Files created and saved in order to test the performance:

```
dd if=/dev/zero of=10kB_file.txt bs=1024 count=10
dd if=/dev/urandom of=1MB_file.txt bs=1M count=1
dd if=/dev/urandom of=1GB_file.txt bs=1M count=1024
```

User Creation and Credential Management:

User Generation Script: I created a Bash script to automate the creation of 30 Nextcloud users. This script utilized Docker's exec command to interact with the Nextcloud instance and add users programmatically.

```
nano create_users.sh
 chmod +x create_users.sh
./create_users.sh
```

Credential Storage: After generating the users, I stored their credentials (username and password pairs) in a text file located at ~/nextcloud/credentials.txt. This file acted as a centralized repository for all generated user credentials.

Credential Usage in Locust: In my Locust script, I implemented logic to read these credentials from the text file. Each simulated user in my Locust tests was assigned a unique set of credentials from this file, allowing for more realistic and varied simulation scenarios.

8. Cloud Provider Choice in production:

- For a production deployment, a cloud provider AWS (Amazon Web Services) is recommended for deploying the cloud-based file storage system for several reasons:

- **Scalability:** AWS provides a highly scalable environment that can automatically adjust according to the application's demands. This is crucial for a file storage system as the number of users and the amount of data can grow significantly. Services like Amazon EC2 allow I to scale up or down easily based on demand.

- **Reliability and Availability:** AWS offers a robust infrastructure with a network of data centers located across various geographical regions and Availability Zones. This ensures high availability and reliability of the services, minimizing the risk of downtime which is essential for any storage solution.

- **Security:** AWS provides comprehensive security features that adhere to industry standards, including data encryption at rest and in transit, network firewalls, identity access management, and detailed auditing capabilities. These features can help secure sensitive data and ensure compliance with various regulatory requirements.

- **Cost-Effectiveness:** AWS offers a pay-as-I-go model that allows I to pay only for the resources I use. This can significantly reduce the costs compared to maintaining an on-premise infrastructure. Additionally, AWS offers various pricing models and services that can help optimize costs based on my specific usage patterns.

- **Wide Range of Services:** AWS offers an extensive range of services that can be integrated with file storage system, including computing, networking, database, analytics, and machine learning services. This can enhance the functionality of system and provide opportunities for innovation and improvement.

- **Ecosystem and Support:** AWS has a large ecosystem of partners, developers, and third-party tools that can extend the capabilities of my file storage system. Additionally, AWS provides various support plans and resources to help I with deployment, management, and troubleshooting.

- **Experience and Market Leadership:** AWS is a market leader in the cloud computing space and has extensive experience hosting and managing large-scale cloud applications. By choosing AWS, we can leverage their expertise and best practices to ensure the success of my deployment.

challenges:

Throughout the process of setting up my Nextcloud instances with Docker and Nginx for load balancing, I encountered various errors and challenges. Here's a summary:

Container Identification Issues: Errors like "No such container" were encountered when I tried to interact with containers that were either not running or incorrectly referenced in my commands.

Solution: To address the "No such container" errors, I ensured that my Docker containers were up and running using docker-compose up and verified their status with docker ps. I corrected wrong references in my commands.

```
docker ps
```

Database Connection Issues: I experienced database connection errors, particularly "Failed to connect to the database" and "Temporary failure in name resolution," indicating issues with the database container's network configuration or the Nextcloud application's database access settings. While I was writing docker compose file I decided to use Mysql 5.7 and successfully run nextcloud on localhost but when I was checking Security & setup warnings, I noticed that I should use Mysql 8.0 or Mariadb so I changed it but then I started to get errors.

Solution: After getting lots of errors I uninstalled docker-compose and install it again and start the project from scratch to solve the issues.

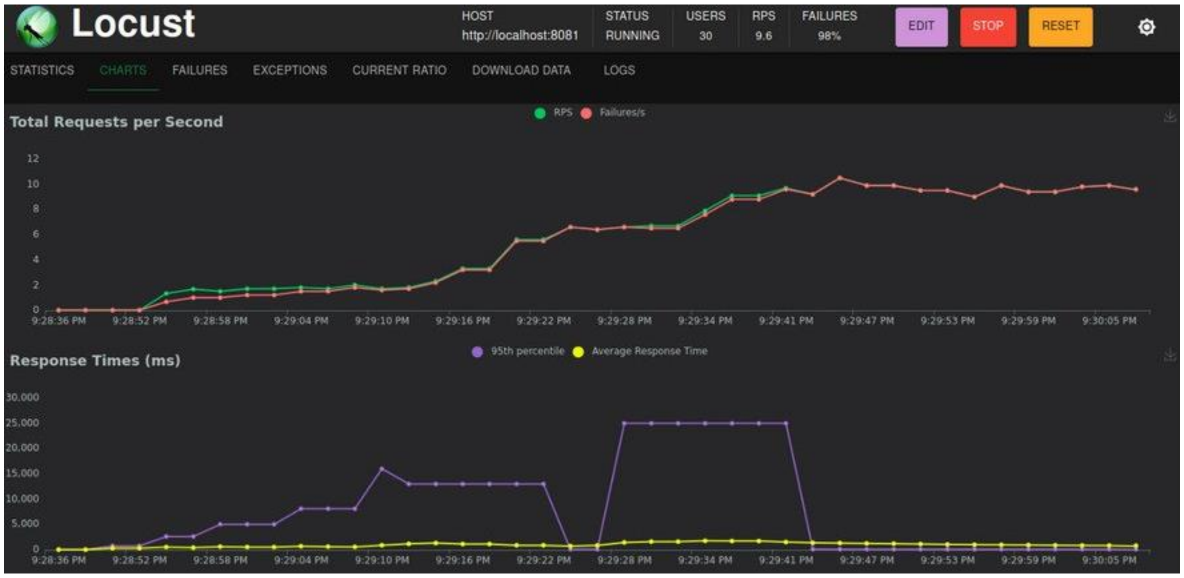
Storage Issues: I experienced storage issues while I try to run docker compose up after making changes to the docker-compose.

Solution: When I faced challenges related to storage from previous unsuccessful attempts, using docker system prune helped by cleaning up the environment. This action could have resolved issues like: Running out of disk space due to accumulated unused Docker images and containers.

```
docker system prune
```

Locust Testing Challenges: During load testing with Locust, I encountered script errors related to user credential unpacking and incorrect file path references. Additionally, there were performance issues with a high failure rate for file downloads.

Solution: I addressed the script errors in Locust by correcting the user credentials file format and ensuring it was properly parsed. I also fixed file path references to ensure the Locust tasks could accurately request the target files. At first, I get errors on unsuccessful login attempts as I didn’t create users to test. Then I write a bash file to create users then remember that in order to be able to download a file from nextcloud first I need to upload some file there. I uploaded the files I created with in admin account and then I gave permission of access to users and fixed the problem. (First, I test only with one user and I get failures all the time, then I understand that I need to make users).



File Operation Limitations: When attempting to download files among multiple users, I ran into limitations or were unsure how to achieve this with Nextcloud and my Locust testing setup.

Solution: I addressed the script errors in Locust by correcting the user credentials file format and ensuring it was properly parsed. I also fixed file path references to ensure the Locust tasks could accurately request the target files.

Future improvements:

- Use tags for testing different kind of files instead of editing the locust file each time.
- Explore options for seamless integration with other cloud services and platforms to support evolving business requirements and technological advancements
- Expand monitoring strategy to include additional metrics and performance indicators beyond what is provided by Nextcloud's built-in tools. Consider integrating with dedicated monitoring platforms like Prometheus, Grafana, or ELK Stack for more advanced monitoring and visualization capabilities.