Cloud Basic - Project report

Emanuele Ruoppolo - SM3800049

Course of AA 23-24 - DSAI

1 Abstract

This project is dedicated to the development of a cloud-based file storage solution, prioritizing user accessibility and data security. The primary focus lies in implementing a seamless system where users can effortlessly manage their files, including uploads, downloads, and deletions, all within their designated private storage spaces. Throughout the project, we will meticulously assess factors such as scalability, security protocols, and cost-efficiency to ensure the chosen solution meets our requirements effectively.

Utilizing Docker and Docker-compose files, I've customized the Nextcloud platform to suit the project needs. This involved extensive configuration adjustments, including user registration, security protocols and password recovery.

2 Nextcloud

Nextcloud has been chosen for this project due to its extensive feature set tailored specifically for cloud-based file management. Its intuitive user interface and robust security measures make it stand out among other options. Additionally, the availability of a Docker container for Nextcloud optimizes the deployment process, ensuring a quick and straightforward setup.

2.1 Setup

First of all the container has been created by a `.yml` file where two volumes have been specified, in our case two services: Nextcloud and a MariaDB database, with the corresponding **volumes** and networks. In particular for Nextcloud has been specified the localhost port 8080:80, so that the container can be accessible from the host OS. The servers has been linked through a network that enables the communication between the two. Its execution has been made by simply running on the command line inside the directory containing the file the following command: `docker compose up -d` that automatically creates the container based on the requested services, deploying the database and the Nextcloud filesystem, to which it is possible to access from the local machine by

typing `https://localhost:8080` using admin username and password reported in the `.yml` file itself.

2.2 Nextcloud accessibility features

User registration and authentication: Users can easily sign up and log in using the default Nextcloud registration process, and its dedicated interface;

Authorizations management: Nextcloud supports the users to have different roles ad authorizations, is it then possible to differentiate admins from simple users, dividing users in group or assigning different kind of privileges.

Admin privileges: Admins can manage the whole setting of the storage and its security, they can easily add or remove users, modify the interface and benefit of a non-limited amount of space;

Storage management: Nextcloud provides the automatic assignment of a fixed Quota of storage to each regular user, in my case this has been fixed at 3 Gb;

File management: Users can easily manage their files by uploading, downloading and deleting them by their dedicated interface.

2.3 Security

By default, upon successful authentication, Nextcloud issues an access token that clients will use for all future HTTP requests. This access token should not be stored on any system other than the client requesting it. The user password is also stored encrypted in the Nextcloud database. In order to allow users to register by themselves, using the web interface, the Registration has been installed and enabled, it provides the common email based sign-up with verification link. In a production setting we would have to setup a server email account to provide the verification links and other email based services. Besides the base security measures other can be enabled in order to reinforce users' data safety. The most useful are:

Data encryption: Users data can be server-side encrypted, both from the interface or by the command line. From the admin interface the path to follow is

`Administration settings` → `Security` → `Server-side encryption`

Enabling this feature files can not be shared from the remote server, but only from an user through the Nextcloud interface. While this is useful the overall performances of the system are reduced and each files requires higher storage space.

Authentication measures: From the same page some authentication requirements can be applied such as safer passwords for the users, containing special characters, capital letters and a minimum length. Also the system provides to enable a two factor authentication (2FA) to prevent unauthorized access and to enable a password-recovery system by enabling an e-mail system connection.

Other options are offered from Nextcloud to increase the system security. First of all, from the interface admins can access to a `Loggigng` page where to control any suspicious activity done on the server, moreover there are few options of anti-viruses that can be downloaded and configured.

3 Performances evaluation with Locust

In the following section the system performances are evaluated using Locust. Locust stands as an open-source tool rooted in Python, dedicated to performance testing. Its aim is simulating vast user numbers to gauge the scalability and performance of web applications. The mechanism is straightforward: it allows to simulate scenarios where multiple user interactions are done simultaneously, from which a load is generated to rigorously test the application or service in question. In order to evaluate the system performances 90 users has been added using the following command:

```
docker exec -eOC_PASS=2password" --user www-data container_name
              /var/www/html/occ user:add --password-from-env
              --group="users" "$USERNAME"
```

Similarly a method to delete the users and to clean the whole storage have been implemented. They're all reachable in the Git repository.

Then a `tasks.py` file has been created containing all the tasks to perform on the system during the test, these are for each user:

`propfind`**:** sends a **PROPFIND** request to retrieve properties from a resource;

`head`: sends a **HEAD** request to check the availability, size, and last modification date of a resource without downloading it;

`read_file`**:** manages a **GET** request to read a file, in our case a shared `Readme.md` file;

`upload_file`**:** simulates the uploading of a file, in our case these task was repeated three times for different-size files, 1Gb, 1Mb and 1Kb respectively. These tasks sends a **PUT** method to upload the desired file from a local directory;

`upload_file_jpg`: analogously uploads with a **PUT** method an image from a local directory;

`delete_file`: deletes a file by using a **DELETE** method.

A weight is assigned to each task in order to determine the frequency with which it is performed. Various tests has been made changing the simulated number of users simultaneously connected, we report three indicative cases: 10, 30 and 90 users and we can see that all the cases has been well managed from the machine (Mac M1) that seems able to handle also a great amount of users, always keeping the failure rate under the 3% of the total amount of requests. In the charts we present the results of the 30 and 90 users scenarios, we can also highlight some of the errors raised that are:

- `HTTPError('503 Server Error: Service Unavailable)` for all the **GET, PROPFIND, PUT** methods;
- `RemoteDisconnected('Remote end closed connection without response')` for the **PROPFIND** method;
- `HTTPError('507 Server Error: Insufficient Storage)` for the **PUT** method.

Figure 1: 30 users.

These errors seem to be reasonable, considering their low frequency, anyway to seek the reasons behind them more tests would be required, in order to understand if they're related to CPU or IO constraints. About the second test, with 90 uses, we can observe an interesting behavior in the response times variability, particularly during the initial login phase. As users complete the authentication process, such variability seems to mitigate, hinting at an efficient resource allocation of the system once it reaches a stable state.

I then performed another kind of test selecting to measure how the system manages different kin of loads. Two scenarios have been simulated to check how

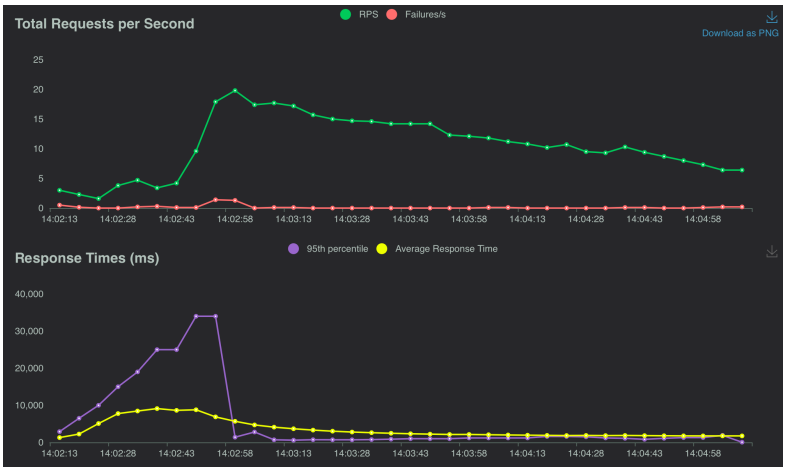

Figure 2: 90 users.

Method	50%ile (ms)	66%ile (ms)	75%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
`HEAD`	18	22	25	30	44	3400	6200	7600
`PROPFIND`	21	26	30	35	55	1800	5800	7100
`DELETE`	31	33	34	35	40	330	6000	7700
`PUT`	36	42	47	51	73	870	4900	6800
`GET`	25	30	35	38	58	670	5500	8000
Aggregated	30	33	37	40	58	670	5800	8000

Table 1: Test performed for small files with 60 users, spawn rate = 10, for $\simeq 2$ min.

the different spawn rate of users and the different file-weight load can impact the behavior of the system:

- Table 1 refers to a scenario in which 60 users are spawned, 10 per second, and each of them can upload a small file ($\sim$KB), delete it, and do the classical head, propfind and get actions.
- Table 2 refers to a scenario in which 20 users are spawned, 5 per second, each of them can upload a bigger file ($\sim$MB) and do the already mentioned operations.

Method	50%ile (ms)	66%ile (ms)	75%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
`HEAD`	28	32	45	36	50	520	1300	1300
`PROPFIND`	33	46	51	53	56	59	1100	1200
`DELETE`	33	35	35	36	42	94	1500	2400
`PUT`	64	77	79	81	84	550	1600	1900
`GET`	39	52	60	61	63	65	1200	1200
Aggregated	37	49	56	61	79	120	1400	2400

Table 2: Test performed for big files with 20 users, spawn rate = 5, for $\simeq$ 2 min.

First of all we see that in both cases few operations have high response time ($\geq$ 95%ile) and these clearly belong to the initial transient phase in which the users are spawned hence the load in increasing. Then when the system reaches an "equilibrium state" the requests themselves are executed in much less time. About the *small files* test we see that the times in the spawning phase are much higher than those of the *big files* and this is reasonable considering the higher spawn rate and the higher number of users. Then in the "equilibrium phase" the situation changes, we see that the times required to manage the operations related to smaller files are lower, even if the number of users is higher, than those related to bigger files. We than ca note that in both the situations the more costly operation is the **PUT** one, especially in big files case, while the others seems to be comparable, except for the **HEAD** operation that seems to be the less expensive.

4 Scalability: On-Cluster Deployment vs. Cloud Services

Two strategic approaches to address the increasing demand on the system are examined here, each with its own set of advantages and drawbacks. On-Cluster Deployment involves the creation of a network of interconnected servers, enhancing scalability and fault tolerance. However, it requires significant setup complexity and maintenance efforts. On the other hand, Cloud Services, like AWS S3, provide seamless scalability, paying for resources as needed, but may raise concerns about data privacy and dependency on the cloud provider's infrastructure.

4.1 On-Cluster

4.1.1 Advantages of On-Cluster Deployment

1. **Scalability:** Offers horizontal scalability, enabling organizations to expand storage needs infinitely by adding more nodes.
2. **Data Redundancy:** Built-in redundancy mechanisms ensure data resilience and high availability.
3. **Security:** Organizations can implement their own privacy policies independently.
4. **Control:** Full control over the cluster infrastructure.

4.1.2 Disadvantages of On-Cluster Deployment

1. **Complexity:** Expertise in distributed systems, networking, and storage technologies is required, leading to higher setup and maintenance complexity.
2. **Costs:** Substantial initial setup costs for hardware and ongoing maintenance expenses contribute to total ownership costs.

4.2 Cloud services

Considering deployment in a production environment, AWS S3 emerges as a compelling choice due to its scalability, reliability, cost-effectiveness, and integration capabilities. AWS provides a scalable infrastructure, high reliability, cost-effectiveness based on usage, and integration with a vast ecosystem of services and tools, making it an ideal cloud provider for deploying systems like Nextcloud.

4.2.1 Advantages of Cloud Services:

1. **Scalability:** On-demand scalability without upfront hardware investments.
2. **Global Accessibility:** Data accessibility from anywhere enhances collaboration and productivity.
3. **Cost Efficiency:** Pay-as-you-go model minimizes upfront costs and optimizes resource utilization.

4.2.2 Disadvantages of Cloud Services:

1. **Data Privacy and Compliance:** Concerns about storing sensitive data in the cloud and compliance with regulatory requirements.

2. **Dependency:** Organizations reliant on cloud services are vulnerable to service outages or disruptions.

Comparing the two solutions, a number of factors must be taken into account when deciding which solution should be chosen, among these there are organizational requirements, budget constraints, scalability needs, and data security issues. In regard to costs, for instance, on-cluster deployment has higher upfront infrastructure costs (capex), while the operative expenses (opex) should probably be assessed taking the account the specifics of any particular situation.

4.3 Conclusions

To summarize, this project delved into the potential implementation of a cloud-based file storage system utilizing Nextcloud, which proved to align seamlessly with the project's needs. Leveraging Nextcloud's intuitive interface, scalable architecture, and Docker container deployment significantly streamlines both development and deployment processes.

Key features selected from Nextcloud, including robust user authentication, role-based access control, private storage allocation, and comprehensive admin management capabilities, form a sturdy foundation for constructing a secure and user-centric file storage platform. Incorporating security measures like server-side file encryption and multi-factor authentication ensures the safeguarding of sensitive user data.

The flexibility of Nextcloud extends to its support for various database backends and storage solutions, allowing for adaptation to diverse deployment scenarios. This report explores two scalable deployment approaches: on-premise deployment with a shared cluster and Cloud provider deployment with autoscaling, offering versatile production solutions tailored to different needs.

Furthermore, a solution through Locust for monitoring and testing analysis has been discussed and deployed for the system, enhancing understanding of platform behavior and resource requirements. This project underscores the significance of strategic decision-making in balancing features, security, scalability, and cost-effectiveness to attain a comprehensive and efficient solution.