Cloud Computing Basic
Cloud-Based File Storage System

Luis Fernando Palacios Flores [*]

University of Trieste

1 Introduction

The purpose of this report is to describe the design, implementation, and deployment of a cloud-based file storage system. Nextcloud is a popular open-source cloud storage solution that provides a platform that allows users to store, sync, and share files, and much more [1]. Here Nextcloud was utilized to leverage its features. Additionally, it includes built-in features to manage user authentication and authorization, manage file operations, and add security measurements to the system.

Regarding user authentication and authorization management, Nextcloud offers robust user management capabilities, allowing users to sign up, log in, and log out securely. Users can have different roles, such as regular users and administrators, which aligns with the requirement for differentiating user roles. Regular users have their private storage space, ensuring data segregation and privacy. Admins have access to management functionalities, enabling them to manage users effectively. For instance, users can be grouped into categories depending on their use of the platform, which might enhance the system's performance.

Concerning file operations management, Nextcloud enables users to upload, download, delete, and share files from their private storage space. These file operations can be performed securely, with encryption options available to ensure data protection during storage and transmission.

Nextcloud prioritizes security, offering features such as end-to-end encryption and secure user authentication mechanisms [2]. Encryption options ensure that data is protected both during storage and transmission. This platform includes access control mechanisms that allow administrators to define granular permissions, reducing the risk of unauthorized access. On the servers's side data encryption at rest, protects data on the disks against unauthorized access. On the users's side, Nextcloud provides robust authentication mechanisms such as multi-factor authentication, minimum password size, and strong passwords with numeric and special characters. Further security measures can be implemented using third-party tools and deployment options will be discussed later.

[*]luisfernando.palaciosflores@studenti.units.it

2 Implemented File Storage System

2.1 Design

Here Docker Compose was used to implement the File Storage System and orchestrate the containers needed. The nextcloud container image [3] served as the basic storage platform. PostgreSQL was utilized for the database server used by Nextcloud to store its data [4]. For faster data access, Redis, an in-memory data structure store, was considered as a caching mechanism [5] and a basic form of performance optimization and scalability.

This is a basic design that leverages the Nextcloud features and that aims to provide some performance improvement to the base platform. On one hand, PostgreSQL offers scalability through features like replication, partitioning, and clustering. PostgreSQL's ability to handle complex queries and data types may make it a preferred choice for larger and more complex Nextcloud installations.

2.2 Monitoring

Prometheus and Grafana are commonly used with Nextcloud for monitoring and visualization of various metrics and performance data. Prometheus can collect and store metrics [6], while Grafana provides a user-friendly interface to analyze and visualize these metrics [7].

Prometheus was utilized to monitor Nextcloud because of its scalability in handling large volumes of metrics data for deployments of any size [6, 8, 9]. It features a robust alerting system for defining rules based on metrics, enabling timely issue identification and response. AlertManager was used for alerting purposes [8]. Additionally, Prometheus worked in conjunction with Node Exporter [10–12], a tool exporting hardware and OS metrics to monitor the host system [8].

2.3 Load Testing

Locust was employed to assess system performance under increasing load. This open-source load-testing tool enables the simulation of multiple concurrent users accessing applications [13]. Load testing on the Nextcloud container was conducted by defining test scenarios that replicate real user behavior. Executing Locust with defined test scenarios using Python scripts allows for stress testing the Nextcloud container and evaluating its performance under different conditions. The tool features a web-based interface for real-time monitoring and analysis of performance metrics.

2.4 Cost-Efficiency

All the components - Nextcloud, PostgreSQL, Redis, Prometheus, Node Exporter, AlertManager, Grafana, and Locust - used in this project were carefully chosen because they offer a lightweight, user-friendly system that is efficient and can be easily scaled, monitored, and tested.

Nextcloud features are user-friendly, such as enabling end-to-end encryption for data transmission, and it offers seamless collaboration with other tools. While security measures may add overhead, they significantly bolster the system's robustness. Redis enhances data access speed and reduces the load on backend servers. Prometheus, Node Exporter, AlertManager, and Grafana collaborate to monitor the system and ensure its proper functioning, making them indispensable components. Locust aids in load testing to pinpoint performance bottlenecks and optimize resource allocation, ultimately promoting cost efficiency by ensuring optimal resource utilization. Unlike the previous tools, this one might be used only in the development phase.

To enhance cost efficiency, the focal point lies in the Nextcloud and PostgreSQL instances operating within the system. Optimizing these tools may necessitate a thorough analysis of the system's usage. Determining the system's size in advance can help in making informed decisions and implementing appropriate measures. Additionally, the monitoring tools can be fine-tuned based on the timing of data collection and operations, taking into account user behavior.

2.5 Launching the system

To utilize the file storage system created for this project along with the files located in the root directory of the project, these steps can be followed:

1. Create network: `./create_network.sh`
2. Launch Nextcloud: `docker-compose -f docker-compose.yml up -d`
3. Sign-in and install Nextcloud in the browser with the url `localhost:8080`
4. Enable basic security measurements and `nextcloud` trusted domain (without this local domain all the load testing functions fail): `./security_setup.sh`
5. Configure Grafana: `localhost:3000`
   - Log in with the default credentials: `admin` for user and password.
   - `Navigate to Connections > Data Sources > Add data source`.
     `Select Prometheus > Settings > Connection`.
     Introduce URL `http://localhost:9090`.
   - `Settings > Build a dashboard > Import a dashboard`
     (there are many dashboard templates, for instance [14]).
   - Add AlertManager as a Data Source with URL `http://alertmanager:9093`.
6. Create users: `./create_users.sh`
7. Generate test data for load-testing: `./generate_data.sh`
8. Launch Locust: `docker-compose -f docker-compose-locust.yml up -d`
9. Perform load testing with Locust: `localhost:8089`
10. Delete users: `./delete_users.sh`

3 Results and Discussion

In local deployment, all Docker Compose containers were successfully created, initiated, and run, with their designated ports accessible. However, despite numerous attempts and research efforts, setting up the Prometheus and Node Exporter containers proved challenging, as they failed to retrieve metrics from the Nextcloud container. While the connection with Grafana was established successfully, the lack of data retrieval resulted in a message indicating the absence of visualizable data, thus leaving the file storage system monitoring incomplete. It appears that a domain name may have been necessary to address this issue, as suggested in [10–12]. Monitoring the file storage system and implementing further enhancements, as discussed in section 4, exceed the scope of this Cloud Computing course. Nevertheless, I designed the system to include these features, discussed their significance, and made an attempt to implement them. These aspects proved to be the most challenging to integrate due to the limited skills acquired during the course and could be considered as future work.

3.1 Load Testing Results

The Nextcloud instance and all the containers were successfully launched following steps 1-8 as defined in section 2.5. Load testing was conducted on the browser using the Locust web application. The HTTP request methods specified in Locust, namely `request`, `get`, and `put` [15], were employed for load testing. These methods were used to simulate actions like fetching metadata about files or directories, logging in as a user and reading files from the server, and uploading a file to the server, respectively.

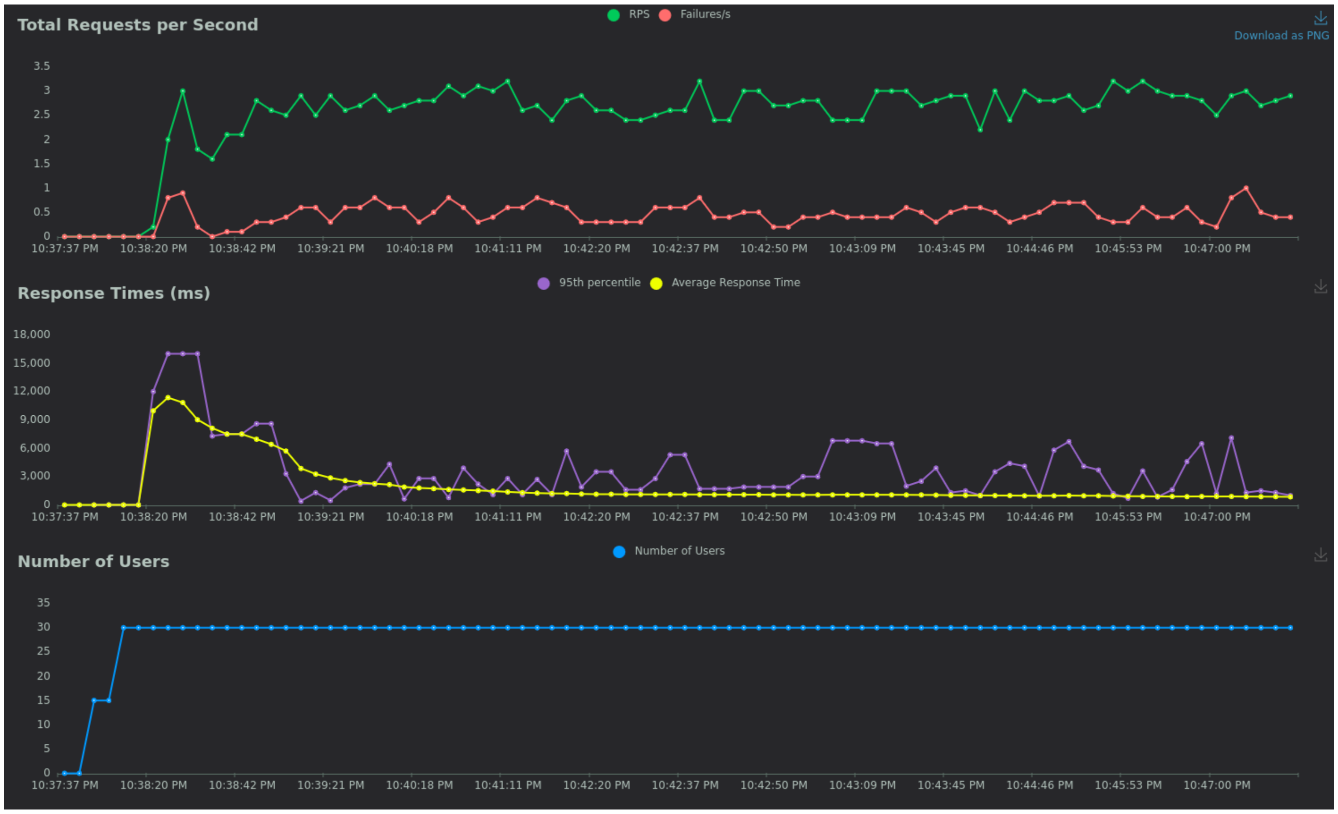

Figure 1: Locust load testing results for files of 1MB with a maximum of 30 users over a duration of 10 minutes, and a frequency of 10 for the task of uploading documents.

I ran my file storage system on my Dell G5 5587 laptop partitioned with Ubuntu OS and equipped with an Intel Core i7-8750H processor, which is a 6-core, 12-thread CPU with a base clock speed of 2.20GHz and a maximum turbo frequency of 4.10GHz. It is important to note that the tests conducted in this project offer insights into the system's performance and scalability. For a comprehensive understanding of these metrics, the system should be deployed on a server.

Therefore, in order to estimate the system's performance under increasing workload, three tests were conducted. In these tests, users were able to upload files of sizes 1KB, 1MB, and 1GB, respectively. All tests were carried out within a time interval of 10 minutes with a ramp-up of

5 users per second. The first two tests were conducted with a maximum of 30 users each, while the third test had 10 users, as my computer was not able to handle numerous requests to upload 1GB of data. Additionally, in the Python script used for load testing, the frequency of the task of uploading documents to the system was changed from 10 to 1 for the final test.

Figures 1 and 2 display the results of load testing. Repetitions of the tests consistently yielded similar outcomes. The results of load testing involving files of 1KB and 1MB exhibited similarities, with only the latter being presented here. In these tests, approximately 3 requests were handled per second, with an average response time of 0.9 seconds.

Under different conditions, tests using 1GB files showed an average of 1 request per second and an average response time of 6.7 seconds. Another test involving 1MB files, conducted under the same conditions as the 1GB test, showed a similar number of requests per second, but with the notable difference of a reduced response time of 0.7 seconds. This represents a significant decrease compared to the results for 1GB files by almost 10 times. These findings suggest that only the response time is impacted by an increasing workload, which aligns with expectations.

The task failures are related to upload and read operations that the HTTP request was not able to resolve because of a client error.

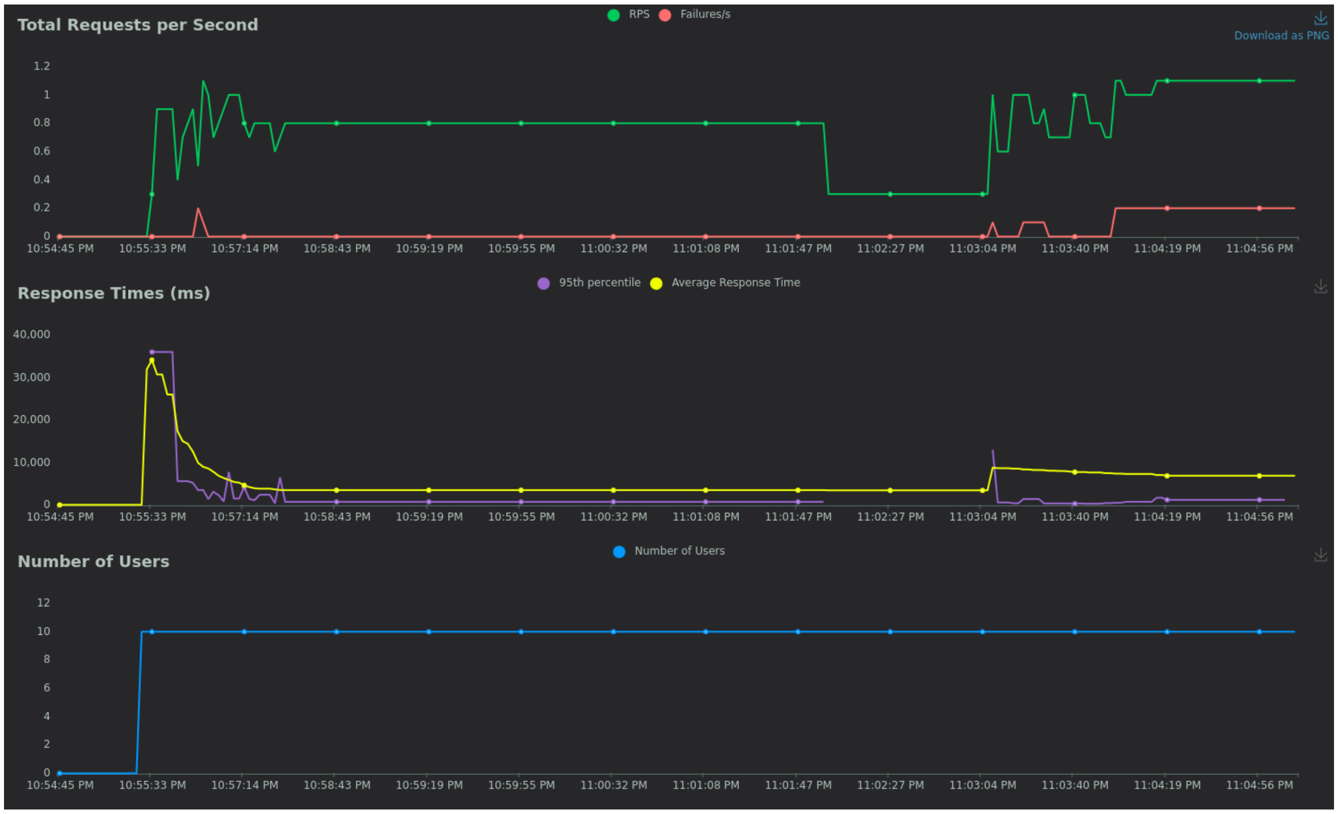

Figure 2: Locust load testing results for files of 1GB with a maximum of 30 users over a duration of 10 minutes, and a frequency of 1 for the task of uploading documents.

4 Theoretical Design Improvements

In this project, I studied and attempted different ways to enhance the file storage system's security, management, and deployment to comply with the project assignment instructions. However, these techniques were far beyond the scope of the course content, and in this section, I'll provide a theoretical discussion aligned with the course coverage.

4.1 Improved Security Measures

The basic security features aforementioned are integrated into Nextcloud and are straightforward to enable according to the administrator's preferences -either using the console or in the browser GUI [16].

Further data transmission security measurements besides the end-to-end encryption allowed by Nextcloud can be implemented using protocols such as SSL or TLS, ensuring data integrity and confidentiality during transit. However, to use these security measurements a domain name was required. Additional configurations regarding the Domain Name System to translate domain names into IP addresses are needed. This solution is feasible and optimal with a domain name and a server running in a cluster for a cloud-based deployment. There are multiple domain providers with accessible prices [17]. Here I didn't understand how to make the DNS configurations.

Network security components like reverse proxies and firewalls could be added to the system to protect and manage the server. A firewall is a network security device or software that monitors and controls incoming and outgoing network traffic based on predefined security rules. It acts as a barrier between a trusted internal network and untrusted external networks (such as the internet), filtering traffic to prevent unauthorized access and protect against various threats. Including a firewall in the internal network of the docker containers can potentially enhance the server's security, particularly for scalable systems.

A reverse proxy is a server that sits between clients (such as web browsers) and backend servers (such as web servers or application servers). It forwards client requests to the appropriate backend server and then returns the server's response to the client. Reverse proxies can provide an additional layer of security by filtering and inspecting incoming requests, blocking malicious traffic, and masking backend server details to reduce the risk of attacks. Additionally, reverse proxies can handle SSL/TLS encryption and decryption, offloading this processing from backend servers to improve performance and simplify management. Nextcloud can be seamlessly run on a reverse proxy [18]. Here I attempted -and failed- to include the Traefik container in my system because of its modern features, designed for cloud-native environments, such as built-in support for dynamically configuring routes and SSL certificates, making it suitable for managing scalable file storage systems [19].

4.2 Scalability

There are several options available to scale the system effectively. Nextcloud can be horizontally scaled by deploying multiple instances and utilizing load balancers to evenly distribute traffic. Depending on the specific requirements of clients, the server's maximum storage capacity, and the maximum per-user capacity, this approach could be ideal for a system experiencing high data traffic. Alternatively, the infrastructure can be vertically scaled by allocating additional computing resources to the existing server and employing more robust hardware to manage increased loads. Moreover, Traefik, as discussed in the previous section, can also be employed for load balancing in addition to serving as a reverse proxy.

In this project, I chose Nextcloud as the platform for the file storage system due to its user-friendly architecture and integrated features. Alternatively, options like MinIO could offer improved performance, given its high-performance object storage capabilities and the ability to auto-scale and dynamically adjust storage capacity based on demand [20].

Another way to manage the scalability of the system is through controlled user management by organizing them into groups to simplify permission management. Implementing strict resource allocation through setting resource quotas for users can prevent resource abuse. Utilizing monitoring tools such as Prometheus and Grafana to track system performance can help identify potential bottlenecks and enhance the system's scalability.

Regular updates to the containers that define the system bring performance enhancements and bug fixes that can improve security and potentially scalability. Automation of container updates can be achieved through the Watchtover tool [2]. Implementing all the mentioned measures can effectively enable the system to manage a growing number of users and files.

4.3 Cloud-Based Deployment

For the cloud-based deployment of the file storage system, Amazon Web Services (AWS) would be a suitable choice due to its dominant position in the market, offering competitive pricing and a wide array of services tailored to meet diverse needs. AWS seamlessly integrates with numerous third-party services, tools, and solutions, ensuring interoperability and compatibility within the ecosystem. Leveraging AWS's infrastructure, Amazon EC2 serves as the compute engine, while Amazon S3 provides scalable and reliable object storage, enabling the system to dynamically adapt to fluctuating workloads and user demands while maintaining optimal performance and resource utilization [21]. Moreover, AWS provides a Domain Name System (DNS) service called Amazon Route 53. This service allows users to register domain names, route traffic to resources such as EC2 instances or S3 buckets, and manage DNS records dynamically.

In addition to its robust infrastructure, AWS provides a comprehensive suite of security features for safeguarding sensitive data in the file storage system. AWS Identity and Access Management (IAM) allows further fine-grained control over user permissions, ensuring only authorized access to resources. Virtual Private Cloud (VPC) enables the creation of isolated network environments, enhancing data privacy and network security.

4.4 Further Cost-Efficiency

Unlike the discussion provided in section 2.4, the additional considerations proposed afterward might be appropriately balanced. The same considerations remain valid, and an update is presented here. The most expensive option in this scenario is the selection of the cloud server, which is AWS S3 in this case. Deciding where to deploy the system and how to manage scalability is crucial, as improper management could lead to significant costs. Subsequently, obtaining a domain name and SSL certificates is a significant decision to consider. AWS tools can be utilized for these purposes. However, it is essential to carefully evaluate the purpose of the file storage system and its clients usage and demand to maximize and optimize the utilization of this cloud solution. The design outlined here may probably need to be adjusted to create a high-performance and resource-efficient system, possibly by substituting some components with native AWS tools or similar components.

5 Conclusions

A file storage system was implemented using Nextcloud as the foundational platform because of its convenient built-in features, including account creation, file management at various user levels, and security measures such as encryption. Docker Compose was utilized to orchestrate different containers of the system's design.

The design incorporated PostgreSQL for data storage, Redis for caching, Prometheus, AlertManager, Node Exporter, and Grafana for monitoring and alerting purposes, along with Locust for load testing. During local deployment, all containers ran successfully, with most of them interacting seamlessly. An exception was noted with Prometheus, as it failed to retrieve data from the specified containers, indicating a need for future improvements in the current implementation. Local load testing results revealed that the system's performance remained consistent with an increasing workload from 1KB to 1MB files but decreased significantly by about 10 times when processing 1GB files.

To further optimize the system, deploying it on a cloud service like AWS is recommended. AWS's expertise in the ecosystem, seamless integration with third-party services, and native data management structures such as object storage enable scalability and dynamic workload handling based on user demand. Moreover, tasks like acquiring a domain name, obtaining SSL certificates for secure data transmission, and implementing additional security measures can be efficiently managed within AWS. It is essential to analyze any system optimization based on potential user demand to achieve a balanced and cost-effective solution.

References

[1] Introduction — nextcloud latest administration manual latest documentation. https://docs.nextcloud.com/server/latest/admin_manual/index.html, 2024. Accessed: 2024-03-16.

[2] User management — nextcloud latest administration manual latest documentation. https://docs.nextcloud.com/server/latest/admin_manual/configuration_user/index.html, 2024. Accessed: 2024-03-16.

[3] nextcloud/docker: Docker image of nextcloud. https://github.com/nextcloud/docker, 2024. Accessed: 2024-03-16.

[4] postgres - official image — docker hub. https://hub.docker.com/_/postgres, 2024. Accessed: 2024-03-16.

[5] redis - official image — docker hub. https://hub.docker.com/_/redis, 2024. Accessed: 2024-03-16.

[6] Collect docker metrics with prometheus — docker docs. https://docs.docker.com/config/daemon/prometheus/, 2024. Accessed: 2024-03-17.

[7] Configure a grafana docker image — grafana documentation. https://grafana.com/docs/grafana/latest/setup-grafana/configure-docker/, 2024. Accessed: 2024-03-17.

[8] Ulises Martinez. Simple prometheus setup on docker compose — by ulises martinez — medium. https://mxulises.medium.com/simple-prometheus-setup-on-docker-compose-f702d5f98579, 2024. Accessed: 2024-03-17.

[9] Prometheus. Installation — prometheus. https://prometheus.io/docs/prometheus/latest/installation/, 2024. Accessed: 2024-03-17.

[10] dotwee/prometheus-nextcloud-exporter: Prometheus exporter for nextcloud servers. https://github.com/dotWee/prometheus-nextcloud-exporter, 2024. Accessed: 2024-03-18.

[11] thecalcaholic/ncp-monitoring-dashboard: A grafana dashboard for nextcloudpi. https://github.com/theCalcaholic/ncp-monitoring-dashboard, 2024. Accessed: 2024-03-18.

[12] nextcloud-exporter/readme.md at master · xperimental/nextcloud-exporter. https://github.com/xperimental/nextcloud-exporter/blob/master/README.md, 2024. Accessed: 2024-03-18.

[13] What is locust? — locust 0.1.dev202 documentation. https://docs.locust.io/en/stable/what-is-locust.html, 2024. Accessed: 2024-03-17.

[14] Nextcloud exporter prometheus dashboard — grafana labs. https://grafana.com/grafana/dashboards/11033-nextcloud/, 2024. Accessed: 2024-03-17.

[15] Api — locust 0.1.dev202 documentation. https://docs.locust.io/en/stable/api.html, 2024. Accessed: 2024-03-18.

[16] Using the occ command — nextcloud latest administration manual latest documentation. https://docs.nextcloud.com/server/latest/admin_manual/configuration_server/occ_command.html, 2024. Accessed: 2024-03-16.

[17] Launch your website, ideas, and future - spaceship. https://www.spaceship.com/, 2024. Accessed: 2024-03-16.

[18] Reverse proxy — nextcloud latest administration manual latest documentation. https://docs.nextcloud.com/server/latest/admin_manual/configuration_server/reverse_proxy_configuration.html, 2024. Accessed: 2024-03-16.

[19] traefik - official image — docker hub. https://hub.docker.com/_/traefik, 2024. Accessed: 2024-03-16.

[20] Minio object storage for container — minio object storage for container. https://min.io/docs/minio/container/index.html, 2024. Accessed: 2024-03-17.

[21] Cloud object storage - amazon s3 - aws. https://aws.amazon.com/s3/.