CLOUD BASIC

CLOUD BASED FILE STORAGE SYSTEM DEPLOYMENT

NIGAM VISHAL [SM3800014]

University of Trieste

1. **INTRODUCTION**

This project is designed to fulfill the need for a cloud-based file storage system that allows users to effortlessly upload, download, and delete files while maintaining the privacy of individual storage areas. To leverage existing solutions and ensure efficiency, Nextcloud has been chosen due to its comprehensive features and ease of deployment with Docker containers.
.

Nextcloud and Its Features

Nextcloud is an exceptional choice for this project, offering a wide range of functionalities specifically designed for efficient file management in a cloud environment. Its features include:

1. **User-Friendly Interface**:
    - Nextcloud provides an intuitive and easy-to-navigate interface, making it accessible for users of all technical levels. The interface allows for seamless file uploads, downloads, sharing, and management.
2. **Robust Security Measures**:
    - **Encryption**: Nextcloud supports end-to-end encryption, ensuring that files are secure both during transmission and while stored on the server.
    - **Authentication**: It offers multiple authentication methods, including two-factor authentication (2FA) and Single Sign-On (SSO), enhancing user security.
    - **Access Controls**: Fine-grained access control policies enable administrators to define who can access, modify, or share files, ensuring data privacy and compliance with regulations.
3. **Scalability**:
    - Nextcloud is designed to scale efficiently, accommodating growing amounts of data and increasing numbers of users. Its architecture supports horizontal scaling, which allows for the addition of more servers to handle larger workloads.
    - **Performance Optimization**: Nextcloud includes performance optimization features, such as caching mechanisms and support for high-performance backend storage solutions, to maintain speed and responsiveness even under heavy loads.
4. **Integration and Extensibility**:
    - **Apps and Plugins**: Nextcloud's extensible platform supports a wide range of apps and plugins, enabling additional functionalities such as calendar and contact management, collaborative editing, and integration with third-party services.
    - **APIs**: It offers comprehensive APIs for integrating with other systems, allowing for seamless interoperability within existing IT infrastructures.
5. **Deployment with Docker**:
    - The availability of a Docker container for Nextcloud simplifies the deployment process significantly. Docker provides a consistent runtime environment, ensuring that Nextcloud can be set up quickly and reliably across different systems.
    - **Ease of Maintenance**: Docker containers make it easier to update, scale, and maintain the Nextcloud instance, as containerized applications are isolated and can be managed independently of the host system.

Tasks Carried

1. User Authentication and Authorization:

Nextcloud facilitates user authentication and authorization with its built-in features, aligning perfectly with the project requirements:

- **User Registration and Login**: Nextcloud offers a straightforward user registration process, allowing users to sign up, log in, and log out effortlessly. The intuitive interface simplifies user experience during these interactions.
- **Role-Based Access Control**: Nextcloud supports different user roles, such as regular users and admins. This built-in role-based access control ensures that each user is assigned the appropriate privileges.
- **Private Storage for Regular Users**: Regular users benefit from Nextcloud's default configuration, which allocates a private storage space for each individual, with a limit defined by a Quota.
- **Admin Management Capabilities**: Nextcloud provides an admin dashboard with features to add, remove, and modify user accounts, simplifying the administrative tasks associated with user management.

2. Security Features

Nextcloud comes with secure defaults to protect the system. For sensitive user data, server-side file encryption can be enabled to prevent unauthorized access, although it may reduce system performance and increase storage requirements. Files remain shareable through the Nextcloud interface but not directly from the server.

To enable encryption via the web interface, one can log in as admin, find the Default Encryption module app, and activate it. Then go to: Administration Settings -> Administration Security -> Server-side encryption

Alternatively, use the command line for the provided docker-compose deployment. Ensure users have strong passwords by setting policies requiring numbers, symbols, and regular password changes. Enforce HTTPS on production servers to prevent data snooping and man-in-the-middle attacks.

```
docker exec --user www-data nextcloud /var/www/html/occ app:enable encryption
docker exec --user www-data nextcloud /var/www/html/occ encryption:enable echo "yes"
docker exec -i --user www-data nextcloud /var/www/html/occ encryption:encrypt-all
```

Password Policy Enforcement with strong password policies in Nextcloud could be implemented to ensure that users create and maintain secure passwords. This includes requirements such as minimum length, complexity, and expiration periods

Analysis of Activities and Logins could also be a measure to prevent unauthorized access is to use tools to analyze activities and logins performed by users. In case of suspicious activities or accesses in the system we can take the necessary countermeasures.

3. Software

3.1 Nextcloud- Nextcloud is well-suited for deployment within containerized environments. It offers Docker images and documentation to facilitate deployment using containerization services such as Docker. Containers are an optimal tool to run this service because they are portable and self-sufficient units that package the application and its dependencies, allowing it to run consistently across different environments.

3.2 Locust -To test and analyze the behaviour of the deployed infrastructure, we use Locust. Locust is an open-source tool for load testing web applications and APIs. It allows users to simulate multiple concurrent users accessing a web application to assess its scalability and performance. With its Python-based scripting and distributed architecture, Locust provides real-time insights into performance metrics.

3.3 Docker Compose file-`docker-compose.yml` file sets up a multi-service environment for Nextcloud, including:

1. **Nextcloud service**: Runs the Nextcloud application.
    - Exposes port 8080.
    - Uses the `nextcloud_data` volume for persistent storage.
    - Connects to the MariaDB database with specified credentials.
2. **MariaDB service**: Provides a database for Nextcloud.
    - Uses the `db_data` volume for persistent storage.
    - Initializes the database using scripts in `./mariadb-init`.
3. **Locust service**: Runs Locust for performance testing of the Nextcloud application.
    - Exposes port 8089 for the Locust web interface.
    - Executes the Locust test script from `./locust/locustfile.py`.
4. **Cleanup service**: Periodically cleans up old files in Nextcloud to free up space.
    - Uses an Alpine image.
    - Mounts the Nextcloud data volume and a custom cleanup script `cleanup.sh`.
    - Runs the cleanup script every hour.

All services are connected through a custom network `nextcloud_network` for internal communication.

4. Monitoring

Next cloud provides some monitoring metrics inside its webui by default.
The system metrics can be accessed by: Administration Settings -> System .

4. Testing & Deployment

Testing:

In order to test the performance of the system in terms of load and IO operations it can be convenient to design stress tests and see how our infrastructure reacts. We have run Locust service which is dockerized as well for performance testing of the Nextcloud application and collected the metrics.

Deployment:

We have used docker and docker-compose. We leverage docker-compose in order to connect locust service to our nextcloud instance. By simple changes to the docker-compose.yml it is possible to switch the Database backend and modify the configuration of Netxcloud.

To run docker containers we used command-

docker-compose up -d

Testing with Locust:

As a security measure by default nextcloud will only authorize requests made from localhost. In order to make the requests from the locust container be accepted we need to add it to the trusted_domains .This is only necessary in order to run load tests correctly.

docker exec --user www-data nextcloud /var/www/html/occ config:system:set trusted_domains 1 --value=nextcloud

Locust needs a few users to simulate interactions with nextcloud. We can spawn 30 test users for it by using the convenient:

./create_users.sh

To perform load tests we have locust url http://localhost:8089/.

I have designed a few tasks, contained in **create_testfiles.py and locustfile.py**. With these tasks Locust will, for instance, create new files of different sizes (1kb, 1mb and 5mb), read a user file content, upload a text file and request a list of all the files owned by the user.

This **`locustfile.py`** script is used for performance testing of a Nextcloud server using Locust. It simulates multiple users (from user1 to user30) performing specific tasks. Each user logs in with HTTP Basic Authentication, then performs a PROPFIND request to check the availability of their directory. The script then uploads a **1kb/1Mb,/5Mb** file to the server, in iterations ,verifies the upload through multiple GET requests, and finally deletes the file. Each user waits between 2 to 5 seconds before repeating the tasks. The script also logs the HTTP response status codes for each operation, helping to identify and document any errors or performance issues during the test.

Given a limited budget and basic infrastructure, AWS is an ideal choice for deploying a Nextcloud-based file storage system due to its cost-effective pricing and minimal management overhead. AWS offers a comprehensive suite of services including compute (EC2), storage (S3), databases (RDS), networking (VPC), and security (IAM), all of which support Nextcloud deployment. AWS's global network of data centers ensures low latency and high performance by placing instances close to users. The platform's scalability, with features like Auto Scaling and AWS Lambda, allows for resource optimization and cost efficiency. AWS also emphasizes security, offering network isolation with VPC, data encryption, identity management, and compliance with certifications such as SOC, ISO, and PCI DSS.

Coming back to the locust test scripts, I attach two charts of the results, where the run is performed on my laptop which is M1 Macbook Pro.I spawn respectively 10 ,20 and 30 concurrent users.

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:

Figure 1. 1KB file get/put/delete comparison between 10/20/30 users

T**otal Requests per Second (RPS)**:

- **Run #1 (10 users)**: RPS starts at around 12 and fluctuates between 10 and 15. This indicates that 10 users are making requests at a stable rate.
- **Run #2 (20 users)**: RPS increases to around 22 and shows slight fluctuations but remains stable overall. This suggests that the system can handle the doubled load effectively.
- **Run #3 (30 users)**: RPS increases further to around 35-38, indicating the system's ability to scale with additional users.

- **Response Times (ms)**:

  - **Run #1**: Average response times are low, around 50ms, with occasional spikes.
  - **Run #2**: Response times spike initially but quickly stabilize. This indicates that the system adjusts to the increased load.
  - **Run #3**: Response times spike more significantly, reaching up to 1000ms but then stabilize. The average response time remains low, showing that the system can manage higher loads but with some initial delays.

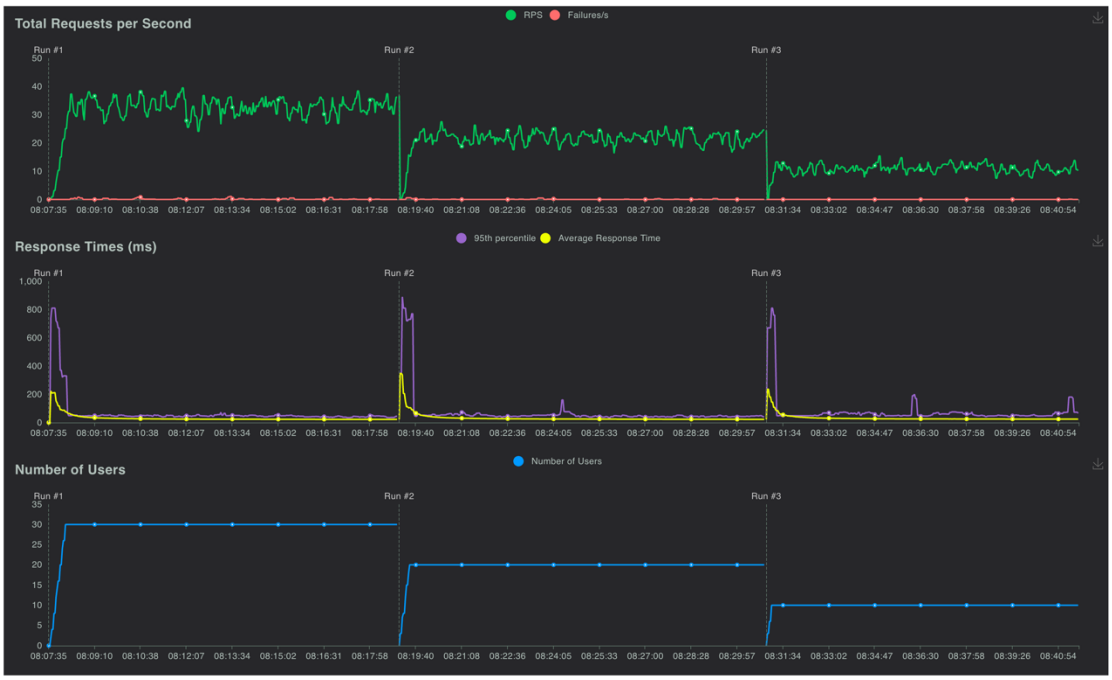
- **Number of Users**:

  - The number of users increases in three stages: 10 users in Run #1, 20 users in Run #2, and 30 users in Run #3.
  - The system handles the incremental user load efficiently, as seen by the corresponding increase in RPS and stable response times.

Figure 2. 5mb file for 30 users

The Locust test with 30 users handling 5MB file operations (PROPFIND, PUT, GET, DELETE) over approximately 8 minutes demonstrates the system's scalability and reliability. The requests per second (RPS) stabilized around 30-40, and response times quickly normalized below 100ms after an initial spike. Notably, there were no failures throughout the test, indicating efficient storage management, likely due to the implemented cleanup script. The results show that the system can handle significant user load and larger file sizes effectively, confirming the robustness and scalability of the Nextcloud deployment.

Figure 3. 1Mb file get/put/delete comparison between 30/20/10 users



The Locust test for 1MB file operations (PROPFIND, PUT, GET, DELETE) was conducted in three runs: the first with 30 users, the second with 20 users, and the third with 10 users. The system handled the load effectively, maintaining a consistent request rate per second (RPS) around 35-45 during each run. Response times initially spiked but quickly stabilized below 200ms on average, indicating good performance under varying user loads. Importantly, there were no significant failures throughout the test, showcasing the system's reliability and efficient handling of file operations even as the number of users decreased. This demonstrates the robustness of the Nextcloud deployment in managing different user loads while maintaining stable performance.

5. Cost-Efficiency

The cost implications of our design are minimal, requiring only a single node (a laptop) accessible from the internet. This setup assumes a small number of users with activities that are not computationally expensive. If user numbers grow or their activities become more demanding, we may need to scale our system, incurring additional costs. Here are some strategies to optimize for cost efficiency:

1. **Resource Usage Optimization**:
    - Optimize resource utilization by right-sizing instances and containers based on workload requirements.
    - Use monitoring tools to identify underutilized resources and scale them down accordingly to minimize costs.
2. **Pay-As-You-Go Model**:
    - Leverage cloud services offering pay-as-you-go pricing models to pay only for the resources consumed.
    - For example, using an Object Storage service in the cloud allows us to pay for storage based on usage, thus reducing the workload on our primary node.
3. **Automated Scaling**:
    - Implement automated scaling to adjust resources in real-time based on demand. This ensures we are only using and paying for what we need at any given time.
4. **Efficient Resource Management**:
    - Use serverless computing options like AWS Lambda for tasks that do not require continuous processing, reducing the need for always-on infrastructure.
5. **Storage Management**:
    - Implement data lifecycle policies to automatically move infrequently accessed data to cheaper storage tiers, ensuring cost-effective storage management.

By implementing these strategies, we can maintain a cost-efficient system that scales effectively with user demand while minimizing unnecessary expenses.

6. Scalability

Handling a Growing Number of Users and Files

1. **Horizontal Scaling**:
    - Expand capacity by adding multiple nodes to distribute the workload. This approach enhances fault tolerance and ensures a robust system but requires orchestration tools for managing the multi-node architecture. It is ideal for long-term scalability.
2. **Vertical Scaling**:
    - Enhance the performance of existing hardware by upgrading CPU, RAM, and storage. This provides an immediate increase in computational power and storage but is limited by the maximum capacity of the hardware. It is a cost-effective short-term solution.
3. **Database Scalability**:

- Use scalable database solutions like MySQL or PostgreSQL to manage increasing data volumes efficiently. Avoid relying on SQLite, which is suitable only for initial testing and not for handling large-scale data operations.

Handling Increased Load and Traffic

1. **Load Balancing**:
   - Implement a load balancer to distribute incoming requests evenly across multiple Nextcloud instances. This prevents any single server from becoming a bottleneck, thereby allowing the system to support a higher number of concurrent users.
2. **Caching**:
   - Integrate caching mechanisms at various levels of the application to reduce backend server load and improve response times. Use caching solutions like Redis to store frequently accessed data and static content, minimizing the number of requests that need processing by the backend servers.
3. **Content Delivery Network (CDN)**:
   - Deploy a CDN to cache and deliver static content closer to users. This reduces the load on the origin servers and improves the user experience by decreasing latency.
4. **Auto-Scaling**:
   - Configure auto-scaling policies that automatically adjust the number of active servers based on the current load. This ensures that resources are scaled up during peak demand and scaled down during low usage periods, optimizing resource utilization and cost.

By implementing these strategies, the system can effectively scale to handle increasing numbers of users and files while maintaining optimal performance and cost-efficiency.

7. Conclusion

In conclusion, this project delves into the implementation of a cloud-based file storage system using Nextcloud, highlighting its robust feature set and suitability for the project's needs. By leveraging Nextcloud's user-friendly interface, scalable architecture, and Docker container deployment, the development and deployment processes are greatly simplified. Key features such as user authentication, role-based access control, private storage allocation, and comprehensive admin management lay a strong foundation for a secure and user-focused file storage platform. Security measures, including server-side file encryption and multi-factor authentication, enhance the protection of sensitive data. Nextcloud's compatibility with various database backends and storage solutions allows for versatile deployment scenarios, including on-premise with shared clusters and cloud-based autoscaling options. Additionally, the project explores solutions for monitoring and testing, providing valuable insights into the system's behavior and resource needs. The integration of Prometheus and Grafana offers further prospects by enabling detailed metrics collection and visualization, enhancing the ability to monitor system performance and resource utilization in real-time. This project emphasizes the importance of balancing features, security, scalability, and cost-effectiveness to develop a comprehensive and efficient file storage solution.