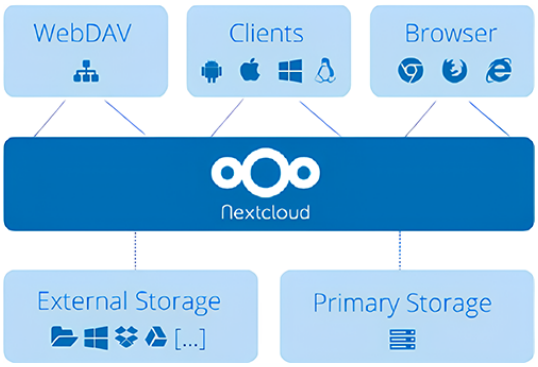
Cloud-Based File Storage System

Project Report - Cloud basic

- Adriano Donninelli adonnine@sissa.it

Introduction

This project aims to address the need for a cloud-based file storage system that enables users to seamlessly upload, download, and delete files while ensuring the privacy of individual storage spaces. In order to not reinvent the wheel and to pursue efficiency, Nextcloud has been selected for its robust features and ease of deployment using Docker containers.



Nextcloud and its features

Nextcloud stands out as a suitable choice for this project due to its comprehensive set of functionalities tailored for file management in a cloud environment. It provides a user-friendly interface, robust security measures, and the ability to scale seamlessly. Moreover, the availability of a Docker container for Nextcloud simplifies the deployment process, ensuring a quick and straightforward usage.

User Authentication and Authorization with Nextcloud

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, streamlining the administrative tasks associated with user management.

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` app can be installed and enabled, providing the common email based signup 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.

Security measures

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`

More conveniently, using the command line (assuming the provided docker-compose deployment):

```
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
```

The system won't be secure unless our users have non trivial passwords. From the same configuration page we can harden the password policy by requiring numbers and symbols. We can also require the password to be changed every few days. To prevent unauthorized access and secure user authentication we can also enable 2 factor authentication.

The clients and the server interact through HTTP protocol, enforcing HTTPS protocol is mandatory in production servers to prevent man in the middle attacks and data snooping.

Database solutions

During development I opted for a simple sqlite database backend for Nextcloud metadata. In a production environment a more powerful backend should be chosen like PostgreSQL or MariaDB, both supported by Nextcloud. Different databases offer different performances but also different replication systems, which are important when we design the scaling and backup solutions of our platform.

Storage solutions

By default Nextcloud stores files in the local file system, which is convenient for a small deployment. In more complex environments it offers flexible solutions for both Object based storage like Amazon S3 and network distributed file systems like NFS. I will discuss in the **scalability** section the most convenient solution based on requirements.

Monitoring

Next cloud provides some monitoring metrics inside its webui by default.

The system metrics can be accessed by: `Administration Settings -> System`.

In order to build a more customizable monitoring system it might be convenient to use a tool like Grafana, backed up by Prometheus for the data collection. These systems are flexible and can be dockerized for easy setup, allowing to monitor also local disks and network operations. I provide more details in the **deployment** section.

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. A python package called `locust` can be used to simulate user interactions with the platform, allowing to gather performance metrics. Further details are discussed in the **deployment** section.

Deployment

The deployment is done through the usage of docker and docker-compose. We leverage docker-compose in order to connect locust / grafana and prometheus 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 the docker containers, `cd` to the folder containing the `docker-compose.yml` file and run

```
docker-compose up -d
```

Monitoring with Grafana

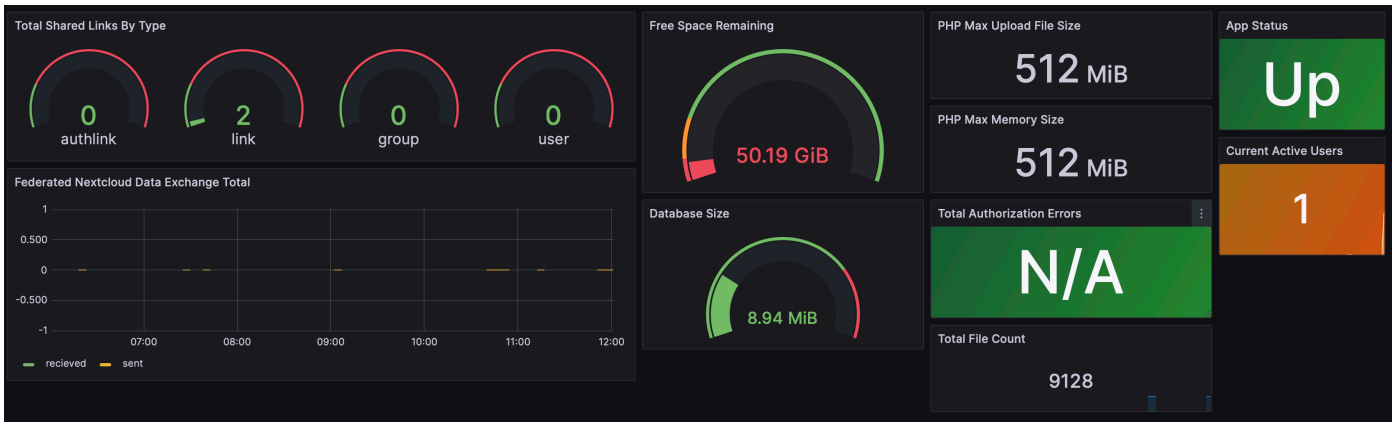
In order to monitor the performance of our system I setup 3 additional containers, one for Grafana, a common monitoring visualization tool, one for prometheus which works as a data source for Grafana and last an open source library which acts as a bridge between the Nextcloud instance and prometheus, called `nextcloud-exporter` (https://github.com/xperimental/nextcloud-exporter).

The docker compose I provide contains a working setup required to connect these containers. Some customization (for example changing the passwords to non default ones) can be applied in order to secure the system.

The only thing we need to setup manually is the Grafana data source and dashboard. In order to do so go to grafana web interface `http://localhost:3000/` and add a new data source for Prometheus.

Set the Prometheus server url to `http://prometheus:9090` (the name of the docker container and the port exposed). Save and test if everything works correctly. Last step is to add a new dashboard. I provide a sample one, `dashboard_sample.json` which can be used as a starting point, simply click on `Create dashboard`, import the json file and select the data source previously created.

The dashboard is now connected and various metrics are visibile. Many more metrics are exported by prometheus and the dashboard can be further customized.



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:

```
sh add_users_to_nextcloud.sh
```

It might take a minute for nextcloud to reload its configuration and accept requests.

To perform load tests we can now use the locust web ui from `http://localhost:8089/`.

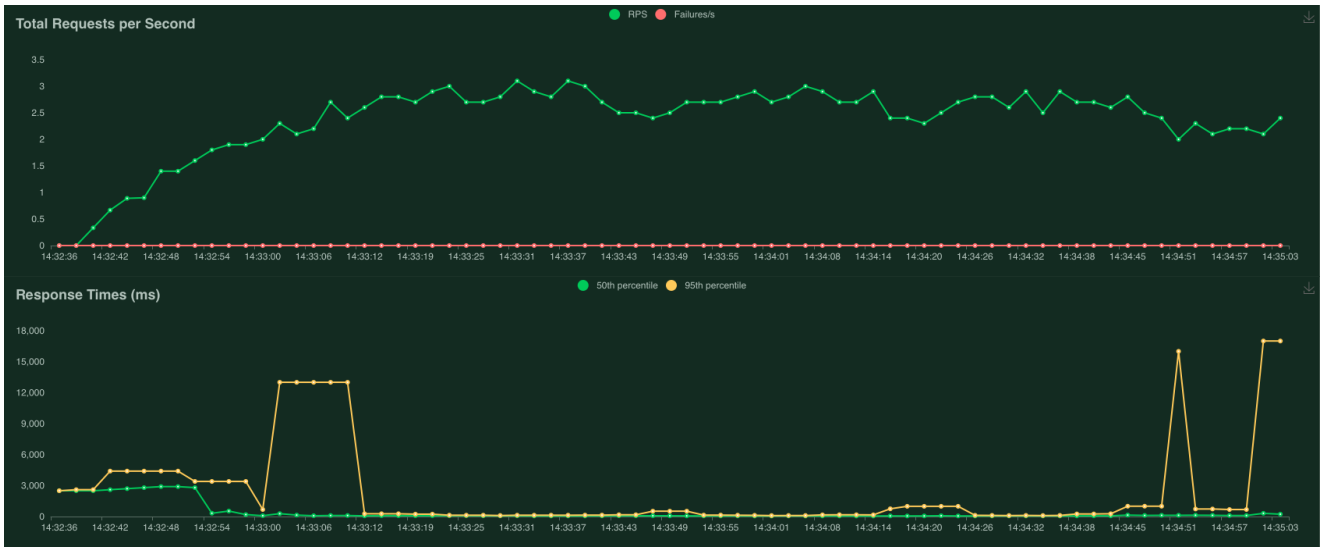
I designed a few tasks, contained in `tasks.py`. With these tasks Locust will, for instance, create new files of different sizes (1kb, 1mb and 1gb), read a user file content, upload a text file and request a list of all the files owned by the user.

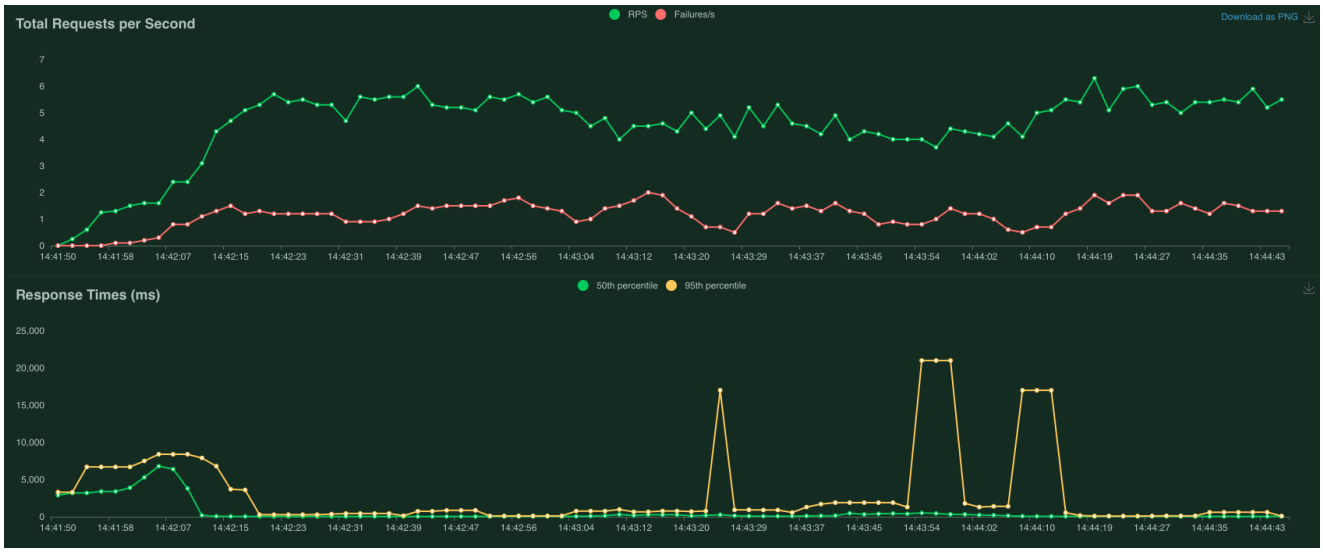
> NOTE: The 1 gb file is not included (because is 1gb!) and should be created with `dd if=/dev/zero of=./test-data/file_1gb bs=1M count=1024`

Once the locust tests are finished it might be convenient to free up all the storage space used by the test users. This can be done with the provided bash script:

```
sh delete_data_test_users.sh
```

I attach two charts of the results, where the run is performed on my laptop which is M2 Macbook Air. Note that encryption is enabled during these tests. I spawn respectively 10 and 20 concurrent users. As we can see my laptop is able to handle 10 users without any failure but starts to struggle and fail requests when we increase to 20.

> 10 user:
>
> 
>

> 20 user
>
> 
>

Analyzing the locust report with 20 concurrent users reveals server issues, particularly in serving requests. Simple PUT requests experience delays of up to 5 seconds. To pinpoint the problem, additional tests would be needed, in order to determine if it's related to CPU or IO constraints.

```
Method  Name                                                  #Reqs     Min (ms)    Max (ms)
PUT /remote.php/dav/files/locust_user0/test_locust_file.txt   407         35          4537
[...]
```

Scalability

I propose two possible solutions to address an increasing demand of the system deployed. Analyzing the advantages/disadvantages and costs of both solutions.

Solution 1: On-Premise Cluster Deployment

If an organization already owns a cluster of nodes or prefer an on-premise solution, it might be beneficial to deploy Nextcloud on multiple nodes with a shared production-ready database. In this scenario, each node receives user requests in order to balance the complessive load and the data is stored on a shared NFS file system. The database, such as PostgreSQL or MariaDB, needs to be robust and capable of handling concurrent connections from multiple nodes. It is also important to implement a redundant replication and backup system to ensure data integrity and availability in case of node failures.

Advantages of Solution 1

1. **Control and Ownership:** The organization has full control over the hardware and infrastructure.
2. **Customization:** On-premise solutions allow for customization of hardware configurations, optimizing the system to specific performance requirements.
3. **Predictable Costs:** While there may be an initial cost for hardware, an organization can have more predictable and potentially lower operational costs over time.
4. **Security Policies:** An Organization can implement and enforce their own security policies without relying on third-party providers.
5. **Data Residency:** On-premise deployment ensures that data stays within the organization physical or virtual infrastructure, which can be important in case of sensible data.

Cost Considerations for Solution 1

- **Infrastructure Costs:** An organization bears the cost of owning and maintaining the physical hardware or virtual machines for the cluster nodes. Note that every few years an on premise solution might require upgrades of the hardware as it becomes outdated or more prone to failures.
- **Backup System Costs:** Implementing a reliable backup and data replication system incurs additional costs.

Solution 2: Cloud Provider Deployment with Autoscaling

Utilizing a cloud provider offers the advantage of scalability and flexibility. In this solution, I propose to use multiple nodes that share a database in the cloud, leveraging the cloud provider's horizontal autoscaling capabilities to handle and balance the load. The data is stored using object storage like Amazon S3, ensuring seamless data storage scalability. The usage of autoscalability features that most cloud providers offer is a must in order to reduce costs and satisfy fluctuating workloads.

Why Amazon with S3?

- **Scalability:** Amazon Web Services (AWS) provides robust autoscaling features, allowing the platform to dynamically adjust resources based on demand. In the principle of Composability this allows to scale only those resources that are needed, in a seamless fashion.
- **Object Storage:** Amazon S3 is a reliable and scalable object storage service, suitable for handling large amounts of file data, it also offers backup and data replication options.

- **Managed Database Services:** AWS offers managed database services like Amazon RDS, which can be used as database backend for Nextcloud, simplifying maintenance and scalability.

Advantages of Solution 2

1. **Scalability:** Cloud providers offer seamless scalability, allowing an organization to adapt quickly to fluctuating workloads and user demands.
2. **Managed Services:** Using a cloud solution might reduce the burden of infrastructure and database maintenance tasks.
3. **Global Availability:** Cloud providers have data centers worldwide, enabling global accessibility, promising consistent availability and reducing latency for users in different geographical locations.
4. **Cost Efficiency:** Cloud providers operate on a pay-as-you-go model, potentially lowering costs for an organization with very variable workloads.

Cost Considerations for Solution 2

- **Usage-Based Costs:** Cloud providers typically charge based on resource usage, which includes computing resources, storage, and data transfer.
- **Object Storage Costs:** Costs are associated with storing data in object storage services like Amazon S3.

Cost Optimization

Comparing the two solutions, on-premise deployment (Solution 1) may incur in higher upfront infrastructure costs, but operational costs might be lower. Cloud provider deployment (Solution 2) offers flexibility and scalability, but costs can accumulate based on usage. I would say that the optimal solution depends on hardware availability and how variable the workloads are.

In order to optimize the costs for Solution 2, if we assume that the data stored is not sensible or that the cloud service provides additional layers of security (such as already storing data in encrypted form), one could disable server side encryption in Nextcloud, leading to smaller file sizes and better performances. This might also apply to Solution 1 in case we have limited amount of computing power or storage memory.

In addition to that if the deployment plan is long term the usage of Reserved instances which many cloud providers offer can reduce costs considerably.

Conclusion

In conclusion, in this project I explored the possible implementation of a cloud-based file storage system using Nextcloud, which offers a robust set of features that align well with the project requirements. Leveraging Nextcloud's user-friendly interface, scalable architecture, and Docker container deployment significantly simplifies the development and deployment process.

The chosen features of Nextcloud, such as user authentication, role-based access control, private storage allocation, and admin management capabilities, provide a solid foundation for building a secure and user-centric file storage platform. The incorporation of security measures, including server-side file encryption and multi-factor authentication, ensures the protection of sensitive user data.

Nextcloud's flexibility in supporting various database backends and storage solutions enables adaptation to diverse deployment scenarios. This report discusses two possible scalable deployment approaches: The on-premise deployment with a shared cluster and the Cloud provider deployment with autoscaling. This offers flexible production solutions to satisfy different requirements.

I also discussed and deployed solutions for both monitoring and testing analysis of the deployed system, allowing to better understand the behaviour and resources requirements of the platform. This project underscores the importance of intelligent decision-making in balancing features, security, scalability, and cost-effectiveness to achieve a comprehensive and efficient solution.