Cloud Basic Project

This project is based on Docker and two Docker-compose files. I customized the deployed Nextcloud system from the administration settings and with curl requests from terminal: the main configurations I did are user registration, Configure SMTP server for email for password recovery and email notifications, create a group of users, turn-on some security settings.

Address Scalability:

Design the system to handle a growing number of users and files.

The system I designed is basically a Nextcloud connected with Mysql database.

The easiest way to handle a growing number of users and files is enlarging the Virtual disk of docker, but we could also implement database sharding, where data is distributed across multiple database instances. This helps distribute the database load and ensures better performance as the user and file count increases.

This approach to scaling can save money but can be harder to implement respect to a vertical scaling in which we simply buy a bigger -and more expensive- disk block.

We could also leverage a Cloud Storage Service, that offers potentially unlimited resources, keeping always in mind the pay-as-you-go policy.

Discuss theoretically how you would handle increased load and traffic.

To handle an increasing load and traffic, we opt for a horizontal approach, we could create more instances of Nextlcoud and distribute the load between them.

We could also add a serviece like Redis to implement caching mechanisms within Nextcloud to reduce the load on the MySQL database, this can improve response times.

Address Security:

Implement secure file storage and transmission.

The first task I did to secure file storage is Server-side encryption: it is available directly from the Security section of Administration settings or with a occ command.

```
# to encrypt all the data
docker exec -u www-data nextcloud php occ encryption:encrypt-all
# to decrypt the data
docker exec -u www-data nextcloud php occ encryption:decrypt-all
```

Server-side encryption makes it possible to encrypt files which are uploaded to the server, it has a drawback in efficiency, since it reduces performance.

As far as transmission, now I'm using Nextcloud locally, but if I had to set up remote access to my Next Cloud Server, I would employ secure communication protocols such as TLS/SSL for encrypting data in transit, that is I would use the https protocol. I would buy a domain and then use services like CloudFlared to create a tunnel between my local server and the domain: in this way I don't have to show my IP address and encrypts autonoumsly the connection.

Discuss how you would secure user authentication.

In order to secure user authentication I would enable Two-Factor Authentication: it works with a TOTP authenticator and can be enforced for all users or specific groups; can be activated with a occ command or in the Security section of Administration Settings.

```
docker exec -u www-data nextcloud php occ app:enable twofactor_totp
docker exec -u www-data nextcloud php occ twofactorauth:enforce --on
docker exec -u www-data nextcloud php occ twofactor:disable <user> totp
```

I would also force the use of strong passwords enabling a monthly expire of the password, which can be done in the Security section of Administration settings.

Discuss measures to prevent unauthorized access.

First of all I would leverage 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.

I would also use the Suspicious Login tool provided by Nextcloud, but this needs to be trained with examples of bad logins.

I could also set a small number of login attempts before the user account is blocked.

Discuss Cost-Efficiency:

Consider the cost implications of your design.

If we keep this system local, the costs are close to zero. The main cost may be an External Hard Drive to increase the available memory of my system.

Let's now suppose we want to make it accessible from the internet and use the system in a small company.

The costs of my design would be the domain and (if needed) the Object Storage service.

About the domain, it's cost is often fixed, we would then need the service of CloudFlared to create a tunnel between my local server and the domain, which is free!

We would then may need an Object Storage service on the cloud, whose cost follws the pay-as-you-go policy.

Discuss how you would optimise the system for cost efficiency.

The first problem to face is right-sizing storage and cost efficiency; citing an article of AWS about Cloud Storage Serivces: "while estimating the server needs for Nextcloud in terms of computation or a database is straightforward, right-sizing disk space is a challenge. Provision too much, and you will have to pay for unused capacity. Provision too little, and you will not be able to store all the files and content you want to work on."

What I would do is, depending on the needs and the money available, to have an hybrid approach: investing in some basic disk space for every-day tasks and rely to the cloud when extra resources are needed.

We must admit that this approach can save money, but it requires attention and domain expertise, since when the resources are again avaialable in the base storage we have to be able to safely move back the resources from the cloud to our base storage, not to pay for a service which is not needed anymore.

Another choice aimed to improve cost/efficiency is to choose Object File Storage instead of traditional storage because it's much cheaper: cents/gb vs dollar/gb.

Deployment:

Provide a deployment plan for your system in a containerized environment on your laptop based on docker and docker-compose.

The first step to do is clearly to install Docker, bundled with Docker Compose.

Then write the Docker-Compose file, putting together the NextCloud and the MariaDb instances.

After that customize the Docker Compose files with appropriate configurations such as volume mounts, network settings and admin credentials.

Execute docker-compose up -d command in the directory where the Docker Compose file is located.

Finally access Nextcloud: in the web browser navigate to the Nextcloud URL http://localhost.

Additionaly, customize Nextcloud settings: log in to Nextcloud or use curl requests to configure user registration, SMTP server for email, security settings, etc.

Discuss how you would monitor and manage the deployed system.

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, if you are on the cloud, because of the pay-as-you-go policy, the more resources you allocate, the larger the bill! 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.

I would also execute regular tests on the performace of my system to make the right adjustements where needed.

Choose a cloud provider that could be used to deploy the system in production and justify your choice.

I would choose Amazon Web Services as the Cloud Provider, precisely the Amazon S3 Object File Storage System, which in many blogs and sites is referred to as the best provider in terms of costs, reliability, and scalability for object storage solutions, paired to the low-latency access. In addition the pay-as-you-go pricing model of AWS can help save money and pay only for the resources needed.

The second ingredient is a domain to gurantee remote access. There is plenty of solutions, but if we want to keep a seamless integration we could stick with AWS, in particular AWS Route 53, which proposes a pay-as-you-go policy, while other companies offer a fixed price $/year.

And then we need a tunnel between my local port and the domain. This can be done also with AWS by leveraging services such as AWS Elastic Load Balancer (ELB) or AWS API Gateway, which provide secure and scalable options for exposing local services to the internet. But as mentioned before, we could also use Cloudflared.

We remember that there is plenty of solutions but I decided to stick with AWS to operate seamlessly in the same enviroment.

Test your infrastructure:

Consider the performance of your system in terms of load and IO operations

This was the most challenging and interesting aspect of my work in terms of writing code, bug fixing and interpreting the results.

First of all, the choice of the application through which perform the test: Locust, a Python library to perform authomated tests of load and IO operations, allowing to set a number of users and a spawn rate. The latter is important because it affects how quickly the load is ramped up during a test and can impact the behavior of the target system under load: the greater the spawn rate the more stressed will be the system.

Where I found the `docker-compose.yml` for locust:

https://github.com/locustio/locust/blob/master/examples/docker-compose/docker-compose.yml

Where I foud the `locustfile.py` -I modified it- to perform the tests:

https://github.com/MorrisJobke/load-testing/blob/master/locust/locustfile.py

Tests were done on my laptop, MacBook Air M2.

Test small file This test was performed with 1 locust worker, 30 users, spawn rate = 10 for approxiamtely 2 minutes.

Method	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
HEAD	15000	15000	17000	19000	20000	20000	21000	21000
PROPFIND	79	93	110	130	160	190	2000	7700
DELETE	410	440	470	520	600	650	800	1400
GET	120	130	150	170	200	230	290	450
PUT	420	450	480	530	610	670	960	7100
Aggregated	140	170	210	330	450	530	730	21000

We can note that the response time for the login (HEAD request), is quite high, because we are using 30 users, spawning them at a rate of 10users/sec.

We can see that DELETE and PUT take almost the same time, but as expected GET is much faster than PUT.

We note that PROPFIND request are the fastest.

Test medium file This test was performed with 1 locust worker, 20 users, spawn rate = 5 for approxiamtely 2 minutes.

Method	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
HEAD	4500	5200	6100	11000	11000	12000	12000	12000
PROPFIND	280	300	330	360	420	480	4500	6700
DELETE	360	400	440	480	570	640	760	840
GET	310	340	360	400	480	560	690	1100
PUT	490	540	610	690	790	930	5500	7200
Aggregated	330	360	400	460	560	670	1400	12000

Here we can see that the HEAD request are faster, becasue we are using less users.

Again the PROPFIND request is the fastest, and the GET request is faster than PUT request and DELETE request.

Test large file Due to an high CPU usage by locust in uploading a large file during the test, I performed it with only 5 users and a spawn ratio of 1, for approxiamtely 1 minute, after that the Locust Worker Container crushed.

Method	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
HEAD	840	1300	1300	1400	1400	1400	1400	1400
PROPFIND	1100	1100	2600	2600	2700	2700	2700	2700
DELETE	1500	1500	1500	1600	1600	1600	1600	1600
GET	8800	11000	11000	11000	11000	11000	11000	11000
PUT	21000	21000	23000	23000	28000	28000	28000	28000
Aggregated	2600	4500	11000	11000	21000	23000	28000	28000

We can note that, by using less users the HEAD requests are faster.

Reasonalbly, dealing with a large file, the DELETE request is faster than the GET request, and the GET request is faster than the PUT request.

The complete reports are available in the folder `/test/reports` of cloud_project.

Conclusion

This project was interesting and useful, helping and forcing me to get in touch with Docker-Desktop and its features, for many reasons:

when trying to perfrorm the tests, I tried to install locust directly on my OS, but I couldn't manage the dependicies and I figured out how important Containers can be!

Another interesting aspect is for sure the connection between containers, which I had no experience about: now I understood how to establish a communication between two isolated containers.

Finally, I understood the importance of being able to personalize a File System, in the future I may need to have a secure and efficient file storage system to collaborate with others: being able to manage users, monitor their activities and personalize the system is crucial.