Cloud Computing Final Project

Valentinis Alessio [SM3800008]

03/2024

Contents

1 Introduction

In this work, the aim was to assess the need of a Cloud File System, by using containarization through Docker and Docker-Compose, that enables users to seamlessly upload, download and deleting file and ensuring privacy of individual spaces, while enabling admins of some further operations and management of the structure, while not being able to access private space of the users. In order not to start everything from scratch I opted for using some pre-written utilities for Docker, by using Nextcloud image as frontend to manage the webUI.

2 What is Nextcloud and which features does it propose

For this work, I chose Nextcloud to adopt as a main application for managing my cloud deployement as it's an open-source solution which offers straight-away a lot of features, like user-friendly UI, robust and customizable security measures and the great scalability by its seamless association to Database Management Systems to manage the storing of the various data and metadata. Moreover the existance of a predefined Docker image makes this application a more than suitable choice, ensuring a simple lightweight deployement and a straightforward and fast usage.

In this section I will be analyzing the features that make Nextcloud a secure and versatile environment to deploy a Cloud File System.

2.1 Nextcloud user authentication and authorization

Nextcloud offers straight-away built-in user authentication and authorization features that allow for different sign up and log in policies. In particular:

- **Registration and login**: Nextcloud offers a very simple registration and login processes out of the box, allowing each user to seamlessly sign up, log in or log out. Moreover, if an adim user isn't already created, the first user to sign to the service will be registered as admin: this may seem a huge problem, such as whoever comes first gets the privilege of being admin, but hopefully the provider will first set the whole service before making it available to the public.
- **Role-based access**: Nextcloud supports by default different users' roles, like simple users and admins. This policy ensures that each user is assigned the appropriate privileges, maintaining private user spaces.
- **Privare storage**: Nextcloud offers by default a private storage space to each user, by ensuring out-of-the-box isolation between different users. Furthermore, each user is allocated a predefined space, defined by a value in its configuration file: by defaults it ammounts to 512MB, but it's easilly modifiable by modifying a dotfile of the nextcloud container. This procedure will be further explained in the deployement section.
- **Admin capabilities**: Nextcloud offers admin a practical dashboard that allows him/her to easilly manage the entirety of the File System, by practically enabling encryption, security policies, add, limir and delete users, and much more.

By default, upon authentication, Nextcloud issues an access token that will be used for all future HTTP accesses, that will be stored only on the system of the client requesting it. On top of that, the password is stored encrypted in the associated Nextcloud database.

By using Nextcloud out-of-the box, it doesn't allow for authonomous registration of the client, but only admins can generate its account. To enable this feature, it's necessary to install and enable the `REGISTRATION` app, available on the `User logo -> Applications` section of the admin dashboard: this way any new client can register to the system using the usual email-password credentials.

2.2 Security measures

Nextcloud comes with several out-of-the-box features regarding the security of the file system. In order to enamble the various security systems, the admin should simply go to the `Administration Settings -> Administration Security` and enable whichever module he wants. Some of them, however, require the installation of a dedicated app, available on the `Application` page. One key feature that must be enabled when passing to production is SSE (Server Side Encryption), that enables the storage of users' data in an encrypted form, of course slowing down performance of the server and enlarging the size necessary to store a single file, but ensuring better security of the file system, above all in cases of sensible data. When SSE is enabled, a used is still capable of sharing files from the webUI, but he/she won't be able to share them from the server. To enable this feature, as specified above, you need to first download the `Default Encryption` app, and then enable SSE from the `Administration settings` page.

Even taken these precautions, your system won't be secure unless you use very secure passwords. This constraint can be enable or modified from the same page of the `Administration Settings`, by strengthening the passord policies. Other than this, to ensure even more secure access, we can enable two-factor authentication, always from the security section.

Of course, when passing from a private environment to a production framework, we can't rely on simple HTTP protocol to manage access to the service, forcing us to switch to HTTPS protocol to ensure better security and prevent attacks during communications.

Furthermore, to secure even more the user-server communication and better load-balancing ratios, we can leverage a reverse proxy manager, like *nginx* and connect the docker instance to an existing domain and managing SSL security policies.

2.3 Database solution and caching mechanism

To manage Nextcloud metadata, a default solution provided by the Nextcloud docker image with SQLite. In order to have a system that can be more similar to one that could be put in a production environment, I opted to implement a MariaDB database backend, which is supported straightawat from Nextcloud. Other than MariaDB, Nextcloud offers support also for PostgreSQL: at this point the choice of the Database differ only in performance and replication/backup systems, which are foundamental when we are talking about commercial suites to assess scalability and backup issues.

Furthermore, to try and optimize performance, a caching mechanism has been provided through a Redis container. This solution aims to ease the response time and overall system workload by keeping a cahe of the recently analyzed files. Its benefits will be analyzed in the *Scalability* section.

2.4 Storage solutions

Nextcloud, if not instructed differently, stores all its files in the local file system, which may be a convenient choice if we limit ourselves to a small deployement. In more advanced environments it offers pretty straight-forward solution based on an existing Storage Server, also in the form of a distributed File System (NFS), or to an Object File System, like an Amazon S3 bucket. Based on the requirements, the optimal solution may vary and I'll discuss this topic in the *I dunno* section.

2.5 Testing

In order to have a more convenient and well documented testing environment, with a pretty easy implementation and a practical monitoring tool, I opted to connect this environment to a *Locust* instance, a well-known Python package that allows to simulate user interaction with the system at a customize workload rate. As a first approach I tried to manage this issue with a local instance of Locust, but I quickly realized on how many dependencies it counted on, making the use of a containerized instance of absolute importance. Furthermore, doing this I could practically access via web browser to its instance and monitor real-time the results of the test.

3 Deployement

The infrastructure has been deployed using Docker along with Docker Compose, which serves as a fundamental container orchestration tool developed by Docker. In particular, leveraging docker-compose orchestrating capabilities, we can deploy several docker containers and the respective network and volumes, only with one command: got to the main directory, containing the `docker-compose.yml` file and perform

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

Also, by just modifing the `docker-compose.yml` file you can easilly modify the backend database, the network of the containers, also with the possibility of connecting it to an existing one.

3.1 Nextcloud setting

Before beginning, if you want to modify the maximum space allocated to each user of Nextcloud, you should modify a .dotfile into the Nextcloud instance.

In order to do this, you can enter the Nextcloud container, through

```
docker exec -it nextcloud /bin/bash
apt update
apt install vim
vim .htaccess
```

And then paste this section

```
php_value memory_limit 2G
php_value upload_max_filesize 4G
php_value post_max_size 4G
php_value max_input_time 3600
php_value max_execution_time 3600
```

Now you can access the container instantiation through `http://localhost:8080`, and using the admin credentials set in the environment of the instance in the `docker-compose.yml`, you can access in your admin account. Now your file system is up and running.

3.2 Testing with Locust

As previously announced, in order to assess the performance of my file system, I decided to use the Python library Locust.

Before starting, it's useful to know that as a security measure, docker by default authorizes requests made only by the localhost. So, to le the locust container make all the requests, we have to add it to the `trusted_domains` of the container. In order to achieve this result, we have to do this command:

```
docker exec --user www-data nextcloud /var/www/html/occ config:system:set

trusted_domains 1 --value=nextcloud
```

Warning: If you don't perform this step, even if you are able to create all the users, tests launched from Locust will incur in permission denied, failing all the tests.

Once you have done this step, you can add all the users for the test, by simply perform the predefined script:

```
sh setup.sh
```

Now, you should be up and running to perform your tests by logging to the Locust container through `http://localhost:8089` and start swarming requests.

To perform the test, I provided a `locustest.py` that does different requests:

- PROPFIND: HTTP request that asks to retreive some metadata associated to a directory;
- GET: HTTP method that allows to retreive a default file created for each user;
- PUT: HTTP method that allows to put on the system a file. I developed different methods that allows to upload (and right after delete, in order to preserve space) files of different sizes, namely 1kB, 1MB, 1GB in order to assess scalability of the system.

Warning: In the repository I didn't put the 1GB file, as it exceedes the maximum size allowed from GitHub. To create it, just

```
dd if=/dev/zero of=test_1gb bs=1M count=1024
```

3.3 Results of the load test