Cloud Computing final project report

Isac Pasianotto

March 2024

Contents

1 Exercise 1

1.1 Requirements

The assignment requires the deployment of a scalable and secure file storage system. In particular the requirements are:

1. *Management of user authentication and authorization*
   (a) User should be able to sing up, log in and log out [1.3]
   (b) User should have different roles, i.e. admin and regular user [1.3]
   (c) Admin should be able to create, delete and modify users [1.3]
   (d) Regular user should have access to a personal storage space [1.3]
2. *Management of file operations*
   - User should be able to upload, download, delete and modify from their private storage space [1.3]
3. *Scalability*
   (a) The designed system should handle a growing number of users and files [1.6]
   (b) A theoretical discussion about the way to increase load and traffic [1.6.1]
4. Security
   (a) The secure file storage and transmission should be implemented [1.4]
   (b) Discussion about how to secure user authentication [1.4, 1.2]
   (c) Discussion about measures to prevent unauthorized access [1.4]
5. Cost-effectiveness
   - Discussion about the costs implications of the designed system and how to optimize them [1.6.3, 1.6.4, 1.6.5, 1.6.2]
6. Deployment
   (a) Provision of containerized environment based on docker [1.2]
   (b) Choice of a cloud provider to deploy the system in a production scenario [1.6]
7. Testing
   (a) Assessment of the system performance: load [1.5.2]
   (b) Assessment of the system performance: I/O [1.5.2]

Following the assignment suggestion, the presented solution will be based on the Nextcloud platform.

1.2 Technical deployment details

The deployment of the system is based on the use of Docker containers. In particular, since more than one container is used, the `docker-compose` tool is used to manage the deployment. The complete of the multi-container setting configuration is done in the `docker-compose.yaml` file. All the user container are taken from the Docker Hub and properly configured, no custom images are used.

All the container shares the same network called `nextcloud_network`, which can be created with:

```
$ docker network create nextcloud_network
```

1.2.1 Used images

The `docker-compose.yml` file uses the following images[1]:

1. `nextcloud:28.0.2-fpm`: base image for the Nextcloud platform. In particular two images are used, the principal one for having the desired storage platform and a second one which is going to run the built-in `cron.sh` script for background tasks[2].
2. `mariadb:11.2.3`: the databases the official Nextcloud documentation suggests to use.
3. `nginx:1.25.4-alpine3.18`: the web server used to serve the Nextcloud platform, the other alternative could have been apache.
4. `redis:7.2.4-alpine`: the cache server; when a client requests a file, the server will first check if the file is in the cache, this should enhance the performance.
5. `caddy:2.7.6-alpine`: a lightweight web server, used as a reverse proxy. It was chosen because it automatically manages the SSL certificate[1.4].

Three volumes are used to store the data: `nextcloud_data`, `db_data` and `caddy_data`. To run the project, first create those volumes with: `docker volume <volume_name>`, then run[3]:

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

The `docker-compose` tool will create the containers, then the system will be ready to use and accessible at `https://localhost`[4]; the first time the system is accessed, the user will be asked to create an admin account.

[1] This configuration is heavily inspired by one of the examples provided by the Nextcloud team.

[2] This solution was inspired by this and this forum pages

[3] If your system relies on SELinux, this will probably cause some issues. For the purpose of this exercise, I've just disabled it with `sudo setenforce 0`. In a production environment, it should be properly configured.

[4] A self-signed certificate is used, so the browser will show a warning.

1.3 Nextcloud characteristics

Nextcloud offers out-of-the-box all the functionalities required at points 1 and 2 of the requirements.

Using the provided web interface, the admin can create, delete and modify users. Obviously, every created user can log in and log out to the system. Every user (normal or admin) has access to a personal storage space, where they can upload, download, delete and modify files that are not accessible by other users.

The admin can also limit the storage space available to each user (default no limitation, but 1GB, 5GB, 15GB quotas are available).

1.4 Security measures

1.4.1 Storage security

Nextcloud software comes with a lot of security features which can help to fulfill the requirements at point 4. First of all, for what regards the secure file storage, there is the possibility to enable the server-side encryption, which encrypts the files before they are uploaded to the server. This can be activated both form the web interface with a admin account (*Administration settings* → *Administration Security* → *Server-side encryption*) or from the command line with the `occ` command:

```
$ docker exec --user www-data nextcloud-app /var/www/html/occ app:enable
    ↪ encryption
$ docker exec --user www-data nextcloud-app /var/www/html/occ encryption:enable
$ # to encrypt all the files already uploaded:
$ echo "yes" | docker exec -i --user www-data nextcloud-app /var/www/html/occ
    ↪ encryption:encrypt-all
```

1.4.2 User authentication security

Moreover, Nextcloud provides useful features to enhance the security of the user authentication; in the same *Administration Security* section mentioned above, the admin can enable:

- Enforce the use of both upper and lower case letters, enforce the use of numbers and enforce the use of special characters in the user password;
- Forbid the use of common passwords;
- Set the minimum length of the password;
- Check the password against the haveibeenpwned.com database.
- Tune the number of days after which the user is forced to change the password and the history of the password (i.e. the user cannot use the same password for a certain number of times).
- Enable the two-factor authentication.

Additionally, Nextcloud is compatible with OAuth 2.0 protocol, which can be used to authenticate the user with a third-party service, and it has an actionable app to prevent brute force attacks.

1.4.3 SSL certificate

All the already mentioned security measures can only guarantee a good security from the server side. In a real-world scenario, the server will not be the `localhost`, but most likely a domain name. In this case, a critical point is the communication between the user's browser (or client) and the server itself. This is the reason why in the `docker-compose.yaml` file, the `caddy` container is used. In fact it automatically manages the SSL certificate, so the communication between the user and the server is encrypted and uses the HTTPS protocol.

1.5 Test of the infrastructure

1.5.1 Load test

To perform a load test, I've used the `locust` testing tool. It is a Python library that allows to define a set of user behaviors and then simulate a large number of users that perform those behaviors.

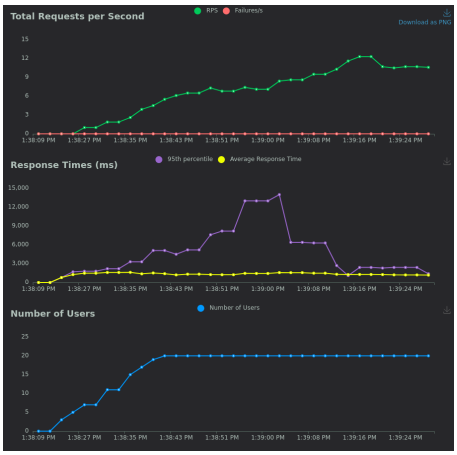
In the `realistic-scenario.py` I have defined a set of behaviors that a user can perform which are:

Task	Probability	Description
`upload_small`	30%	Upload a small file: 13.5KB
`upload_medium`	35%	Upload a medium file: 42.0MB
`upload_large`	5%	Upload a large file: 1.0GB
`download`	30%	Download the medium file

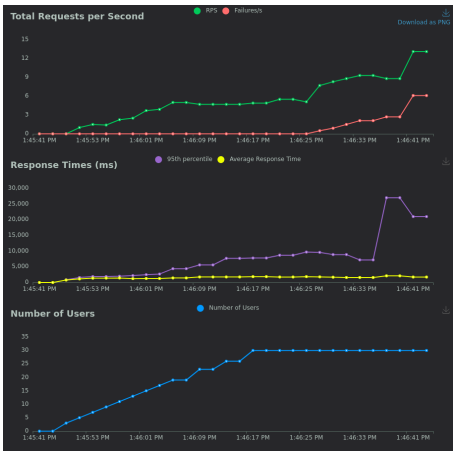
I've performed the test on my laptop, which as a Intel Core i5-8365U CPU and 2×8GB of DDR4 RAM. Locust was set to spawn a user each second, up to a given upper limit, and at each second, for all the currently spawned users, try to perform one of the above tasks.

It turned out that my hardware was able to handle up to 20 users without any issue, but when the number of users was increased to 30, the system started to show the "*429: Too Many Requests*" error[5], see Figure 1.

[5] It's also worth to be mentioned that warning about usage over 90% of the CPU.



(a) Load test with 20 users, no failures are reported



(b) Load test with 30 users, failures start to appear as all the users are spawned

Figure 1: Comparison of load tests with different user counts

1.5.2 I/O test

To asses the I/O traffic, I've used the `docker stats` tool, which provides a real-time view of the CPU, memory, network, and disk I/O usage of the running containers. During the load test reported in the Figure 1, the I/O traffic was monitored in this way. The result is shown in Figure 2.

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
c301e9046ca4	nextcloud-web	0.00%	7.008MiB / 15.25GiB	0.04%	8.99kB / 7.44kB	32.8kB / 0B	9
7318bef8663d	nextcloud-app	0.01%	129.5MiB / 15.25GiB	0.83%	79kB / 40.6kB	1.16MB / 0B	129
a0e6388f4b5b	nextcloud-cron	0.00%	388KiB / 15.25GiB	0.00%	3.29kB / 0B	467kB / 0B	1
802cf1c89eb6	reverse-proxy	0.00%	12.92MiB / 15.25GiB	0.08%	9.48kB / 7.19kB	1.75MB / 131kB	13
b0cf1ddb1723	redis-dbcache	0.28%	2.863MiB / 15.25GiB	0.02%	26.6kB / 12.2kB	0B / 0B	5
3676a81a1156	mariadb-database	0.02%	141.3MiB / 15.25GiB	0.90%	19.3kB / 58.4kB	13.9MB / 45.3MB	33

(a) Situation at newly spawned containers

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
841ca50190e3	nextcloud-web	0.00%	7.824MiB / 15.25GiB	0.05%	10.2GB / 8.31GB	696kB / 0B	9
3d602654eb9f	nextcloud-app	0.01%	1.244GiB / 15.25GiB	8.16%	8.33GB / 1.99GB	644MB / 1.85GB	129
0cefe5bdaf2b	nextcloud-cron	0.00%	912KiB / 15.25GiB	0.01%	6.11MB / 3.65MB	516kB / 8.19kB	1
3d3acc3d0017	redis-dbcache	0.25%	3.555MiB / 15.25GiB	0.02%	19.7MB / 8.14MB	590kB / 741kB	5
40f4190ceb60	mariadb-database	0.02%	133.8MiB / 15.25GiB	0.86%	50.5MB / 101MB	40.5MB / 3.08GB	31
be16ea977e0c	reverse-proxy	0.00%	19.58MiB / 15.25GiB	0.13%	8.33GB / 8.32GB	1.85MB / 238kB	13

(b) 20 users performing 1 action per second, 60 seconds

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
c301e9046ca4	nextcloud-web	0.00%	7.562MiB / 15.25GiB	0.05%	11.5GB / 10.3GB	32.8kB / 0B	9
7318bef8663d	nextcloud-app	0.01%	777.7MiB / 15.25GiB	4.98%	10.2GB / 1.26GB	52.4MB / 414MB	129
a0e6388f4b5b	nextcloud-cron	0.00%	612KiB / 15.25GiB	0.00%	398kB / 228kB	1.11MB / 0B	1
802cf1c89eb6	reverse-proxy	0.00%	20.8MiB / 15.25GiB	0.13%	10.3GB / 10.3GB	1.75MB / 131kB	13
b0cf1ddb1723	redis-dbcache	0.25%	3.438MiB / 15.25GiB	0.02%	2.92MB / 1.87MB	0B / 0B	5
3676a81a1156	mariadb-database	0.02%	159.1MiB / 15.25GiB	1.02%	2.91MB / 9.04MB	14MB / 164MB	47

(c) 30 users performing 1 action per second, 60 seconds

Figure 2: I/O traffic during the load test

We can see that the container which are most stressed are the proxy the web server and the nextcloud app, which is the one that handles requests. The database server is not stressed. This can be explained by the fact that the database is used to store the metadata of the files, and not the actual files, so the I/O traffic in this case is lower.

1.6 Deployment plans for hypothetical real-world scenario

In a real-world scenario, the requirements of the system will be much higher than the one of the test, but also the resources available will be much more permissive. In order to scale this solution to a production scenario some paths can be followed.

1.6.1 Approach-independent considerations

Imagining a much higher traffic, some primary considerations can be made: first of all, some of the containers can be scaled vertically (i.e. increase the resources available to the container), for example the `redis` container, which is used as a cache server, in this way the cache hit rate will be increased, and the communication with the database will be reduced. Also the `caddy` container can be scaled vertically, in order to handle more SSL connections.

Since `caddy` acts also as a reverse proxy, it can be used to balance the traffic among the different instances of the `nextcloud` and `nginx` containers which means that more than one instance of the `nextcloud` and `nginx` containers can be used, allowing to scale horizontally the system and distributing the traffic among the different instances. Finally, also the database can be scaled horizontally, in order to handle more requests and to guarantee more reliability.

In addition to that, an evaluation about the trade-off between security and performance should be made. The server-side encryption can be disabled, in order to reduce the CPU usage, but this will reduce the security of the system. The possibility of disabling it and leave to the users the responsibility of encrypting the files which must be kept secret should be evaluated.

1.6.2 On-site approach

Assuming that the organization has its own data center, the system can be deployed on it.

Advantages:

- The organization has full control over the infrastructure;
- The costs are fully predictable, easier to manage and optimize;
- The data are stored in the organization's data center, so the organization has full control over them.
- The organization does not have to rely on a third-party organization and adopt its own policies, procedures and security measures.

Disadvantages:

- The organization has to manage the infrastructure, which can be complex and expensive;
- The organization has to manage the security of the infrastructure, which can be complex and expensive;
- The organization must take care of the scalability of the project all by itself.
- The initial investment to purchase the hardware can be high.

1.6.3 IaaS approach

Assuming that the organization does not have its own data center, and considering the cost of acquiring and maintaining it too high, the system can be deployed on a cloud provider.

In the case of *Infrastructure as a Service* (IaaS) approach, the organization can rent the resources needed with the "pay as you go" model. Still on the organization to manage the rented resources, install and configure the system to run the wanted software.

Advantages:

- The organization does not have to invest in hardware;
- The scale-up and scale-down of the resources is easy and fast;
- The organization does not have to manage the physical infrastructure, which can be complex and expensive;
- The provider can offer a object storage service, that is a reliable and scalable way to store the files which offers also backup and disaster recovery features.

Disadvantages:

- The price of the resources could be less predictable;
- The renter may increase the price of the resources;
- The organization still to take care of the security of the system it is running;
- There are risks leaved to the provider the organization has to rely on (eg. Network, "zero-day" vulnerabilities for the hypervisor, etc.);
- The data are not physically stored in the organization's data center;

1.6.4 PaaS approach

In this case (*Platform as a Service*), the organization can rent a platform where the vendor takes care of the hardware infrastructure, but also to installing and configuring the software. In this scenario, theoretically, the organization just needs to manage the application and the data, and the vendor will take care of the rest.

Advantages:

- The organization does not have to invest in hardware and in time to configure it to properly run the software;
- The organization need to manage only the application and the data;
- Most of the security measures which are needed are on the vendor side;
- The organization can rely on the vendor to take care of the scalability of the system.

Disadvantages:

- The more the vendor takes care of, the more you pay: this solution should be the more expensive w.r.t. the IaaS and on-site approaches;

- The more the vendor takes care of, the less control the organization has over the infrastructure, and the more it has to trust the vendor for the security concerns;
- The data are not physically stored in the organization's data center;

1.6.5 SaaS approach

Finally, the organization can rent a software as a service. In this case the organization does not have to manage anything, just to use the software with the features provided by the vendor and pay for it.

Advantages:

- The organization does not have to invest in hardware and in time to configure it to properly run the software;
- The organization does not have to manage the application and the data, which can be time-consuming and expensive;
- The organization does not have to manage the security of the system;
- The organization does not have to manage the scalability of the system;

Disadvantages:

- Probably the most expensive solution;
- The organization has no control (and probably no knowledge) about the infrastructure and the security measures;
- The organization must completely rely on the vendor for the security and the reliability of the system;

2 Exercise 2

2.1 Requirements

For the *advanced module* of the course, the assignment was to redeploy the services from the previous exercise, but using *kubernetes* and *helm*. In particular it was required to create a *one-node* kubernetes cluster using `k8s` and deploy the services using `helm`. The service must be accessible from IP or FQDN, and eventual database or third-party services must run in their own pods.

2.2 Solution

The solution is based on the official nextcloud helm chart which is a *community maintained* that comes with almost all the feature needed for this exercise. This chart relies on the Bitnami `PostgreSQL`, `Redis` charts. The difference in the structure between the kubernetes-based solution and the previous one are the following:

- The external database is now a `PostgreSQL` instance instead of `mariadb`. This is done due to an issue with mariadb and the nextcloud helm chart unresolved at the time of writing.
- The access to Nextcloud is now done through a `Kubernetes service`[6] instead of using `Caddy` as a reverse proxy.

For this exercise, I've created the kubernetes cluster using a virtual machine with `vagrant`.

2.3 PV and PVC

In order to persist the data, and to avoid data loss in case of pod deletion or crash (Pods-crash proof was a requirement), I've used a `PersistentVolume` claimed by pods using a `PersistentVolumeClaim` for both the `PostgreSQL` and the `Nextcloud` pods.

```
apiVersion: v1
kind: PersistentVolume
metadata:
    name: nextcloud-pv
spec:
    capacity:
    storage: 5Gi
    volumeMode: Filesystem
    storageClassName: local-path
    accessModes:
    - ReadWriteMany
    persistentVolumeReclaimPolicy: Recycle
    local:
    path: /home/vagrant/pvs/nextcloud
```

[6] I've considered also the Ingress API and tried to implement it. Another possible solution (not tested) were to use the `Gateway API`, which is meant to be a replacement for the Ingress API which is flagged as *frozen*.

```
   nodeAffinity:
   required:
       nodeSelectorTerms:
       - matchExpressions:
       - key: kubernetes.io/hostname
           operator: In
           values:
           - ex2-00
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nextcloud-claim
spec:
  accessModes:
    - ReadWriteMany
  volumeMode: Filesystem
  resources:
    requests:
      storage: 5Gi
  storageClassName: local-path
```

2.3.1 Limitation of the back-end storage choice

The `local-path` storage simply maps a directory on the host machine to a volume in the pod. This is the simplest way to provide a persistent storage, but it has some limitations, in particular:

- *Single point of failure*: If the host machine fails, the data is lost.
- *No redundancy*: Out of the box no redundancy is provided, if the storage unit which provides the directory fails, the data is lost.
- *Force pod to run on a specific node*: The pods which relies on those PVs must run on the same node which provides the storage (set in the `nodeAffinity` field).
- *No dynamic provisioning*: The PVs must be created manually.
- *No scalability*: The storage is upper-bounded by the capacity of the host machine.

A possible improvement could be to use a `NFS` server to provide the storage among all the nodes of the cluster.

2.4 Steps to be taken to have high availability

In order to have high availability, the following steps must be taken:

- *Multi-node cluster*: Using a multi-node cluster, the pods can be scheduled on different nodes, so that if a node fails, the pods can be rescheduled on another node. This does not come for free, as it requires a more solution for the PVs, as mentioned in the previous section.

- *Database replication*: The database can be distributed among different nodes
- *Pod anti-affinity*: Pods can be scheduled on different nodes, this both ensures high availability and could improve the performance (pod replicas are not meant to communicate with each other, so there is a better distribution of the computation load with no communication overhead).

Note that the Nextcloud helm chart already provides a `replica` field, which can be set to a value greater than 1 to have multiple replicas of the same pod. Moreover it comes also with the Horizontal Pod Autoscaler which can be used to automatically scale the number of pods based on CPU or memory usage. This can be useful, since as said in the previous exercise, enabling the server-side encryption can be CPU intensive, Spawning more pods can help to distribute the load.

2.5 Advantages and disadvantages of the kubernetes-based solution

Some of the advantages of the kubernetes-based solution are the following:

- *Scalability*: The solution can be easily scaled by adding more nodes to the cluster, or by adding more replicas of the pods.
- *High availability*: The solution can be made highly available by using a multi-node cluster and by using the `replica` field of the helm chart.
- *Resource management*: The kubernetes scheduler can be used to distribute the pods among the nodes, and the Horizontal Pod Autoscaler can be used to automatically scale the number of pods based on CPU or memory usage.

Some of the disadvantages of the kubernetes-based solution are the following:

- *Minimum requirements*: Compared to the docker approach which has no minimum requirements, to run kubernetes a minimum of 2GB of RAM and 2-core CPU is required.
- *Complexity*: The kubernetes-based solution is more complex to set up.
- *Software stack*: Kubernetes is a newer technology, and it is not as mature as docker, lots of API or features are still in beta state (eg. MetalLB) or can be flagged as *frozen* (eg. Ingress API). Keeping up with the latest changes can be more challenging.

2.6 General schema of the solution

As previously said, the solution is based on the official nextcloud helm chart, which was used in combination with a well-defined set of values reported in the `values.yaml` file.

Following the README.md instructions, the final result will be the following:

- The access is balanced by MetalLB and the service is exposed through an external IP.

- Once the service is reached, the traffic is redirected to one of the nextcloud pods.
- Each nextcloud pod is composed by three containers:
  - The nextcloud application
  - A sidecar container which is used to run cron jobs
  - A container for the nginx web server
- All the nextcloud pods are connected to the same PVC, which is used to persist the data.
- The nextcloud pods are connected to the PostgreSQL database, which resides in its own pod and is connected to its own PVC.
- A Redis pod is used to cache the data, no PVC is used for this pod since in case of crash the application still works, even if slower until the cache is rebuilt.
- A separate service used to monitoring the nextcloud instance, which redirects the traffic to the monitoring pod. The monitoring pod is based on nextcloud-exporter metrics exporter tool.

2.6.1 Possible improvements

Some possible improvements to the presented solution are:

- *Encrypted connection*: Accessing with `http` is not secure, the connection should be encrypted using `https` in a real-world scenario.
- *Backup*: A backup solution must be implemented, in order to avoid data loss in case of disaster.
- *Distributed solution*: As previously said, running this solution on a multi-node cluster has many advantages even if it requires a more complex setup.
- *More robust ingress solution*: Instead of exposing the service, some other solution like the Gateway API could be used.
- *Access Roles Based Control*: To improve the security: all the container are performing their tasks as root, this can be avoided by using a more fine-grained access control.