Cloud Computing Assignment
Final Project Report

Piero Zappi

August 2024

1 Cloud Basic Exercise

1.1 Introduction

The assignment requires to identify, deploy and develop a cloud-based file storage system. Specifically, the system should enable users to seamlessly upload, download and delete files, while ensuring each user has his own private storage space. Finally, the deployed platform should be secure, scalable and cost efficient.
Following the assignment's suggestion, the *Nextcloud* platform has been selected as the foundation for the proposed solution.

1.2 Deployment

The deployment of the file storage system is done through the use of `docker` containers; specifically, since many containers are involved, `docker-compose` is employed to build the system. A `docker-compose.yaml` file is used to set up a multi-service containerized environment to run *Nextcloud* with a *MariaDB* database, *Redis* caching and *Locust* for load testing. The *Redis* container is used as a cache server to store frequently accessed data in memory, reducing direct communication with the database; this approach enhances response times and reduces the load on the database, leading to improved overall system performance. Each service is configured with its respective `docker` image, environment variables, volumes for persistent storage and network settings, ensuring seamless integration; in particular, all containers share the same network.
All the employed deployment files and configuration scripts are available on the author's GitHub repository [1]. Additionally, a `README.md` file provides detailed instructions on how to deploy and use the implemented system.

1.3 Nextcloud Platform

Nextcloud is an open source cloud storage platform backed by an extensive documentation, offering a broad range of features that go beyond basic file storage. It provides a user-friendly web interface, robust security measures out of the box and the ability to seamlessly scale the system to handle increased load and traffic. Additionally, the deployment process is streamlined by the availability of a `docker` container, making it easier to set up and manage the system in various environments.

1.3.1 User Authentication and Authorization

Nextcloud provides built-in features for user authentication and authorization, allowing users to easily sign up, log in and log out through an intuitive interface. The platform implements role-based access control, where users can be assigned different roles, such as regular users or admins, ensuring that each user has the appropriate permissions. Admins can manage user accounts directly from the *Nextcloud* admin dashboard, providing the ability to create, delete, and modify accounts with ease, enhancing both security and user management within the system.

1.3.2 Storage and File Operations

Upon account creation, each user is assigned a private storage space governed by a global quota. Admins have the ability to adjust individual storage quotas for each user through the *Nextcloud* administrators web interface. Users can easily upload, download, and delete files within their allocated private storage space, providing a seamless, secure and efficient way to manage their personal data.

1.3.3 Security

Nextcloud provides a comprehensive set of robust and useful security features designed to enhance the security of the deployed system, all of which can be configured through the administrators web interface.
In order to implement secure file storage and transmission, *Nextcloud* offers the possibility to enable the server-side encryption to encrypt the files before they are actually uploaded to the server. This feature comes with limitations like a performance penalty, but it greatly strengthens the system's security by preventing unauthorized access to sensitive data. Note that this encryption can also be activated via the command line using the `occ` command (see the `system_setup.sh` script available on the author's GitHub repository [1]).
To further secure user authentication and prevent unauthorized access, admins can enable the two-factor authentication through the dedicated administrators interface. Moreover, the same configuration page also allows admins to harden the password policy by setting the minimum password length, the user password history (number of times the user can use the same password), the number of days until a user password expires and the number of login attempts before the user account gets disabled. Additionally, admins can also impose strong password practices among the users, by forbidding common passwords, enforcing the use of both upper and lower case, numeric and special characters in the user passwords and finally by checking the passwords against the list of breached passwords from `haveibeenpwned.com`.
Overall, these features significantly increase the security of the system, ensuring robust protection against unauthorized access.

1.3.4 Monitoring

Nextcloud provides a set of built-in features for monitoring and managing the deployed system, all accessible through the administrators web interface. These tools offer valuable insights into the system's performance and security, allowing admins to track essential system metrics, such as CPU and memory usage, storage capacity and network activity. Additionally, *Nextcloud* offers comprehensive logging capabilities that allow admins to monitor user activities, file access, and system events. These logs make it easier to detect unusual behaviors, troubleshoot issues and maintain the overall security of the system.

1.4 Test of the infrastructure

In order to evaluate the deployed system's performance in terms of load and IO operations, a comprehensive load test was conducted using the *Locust* load testing tool. *Locust* is an open-source `Python` library that allows to simulate high traffic on a system to assess its performance under stress; developers can define custom test scenarios using regular `Python` code to simulate user interactions with the platform, providing insights into how well the system handles concurrent users and various IO operations. Additionally, the *Locust* web interface provides detailed performance reports, giving a clear view of the system's behavior under the simulated load conditions.
Different tasks to be performed by users were defined in a `Python` script, with each task assigned a specific weight. This weight indicates the relative frequency at which each task is executed by users during the test, allowing the simulation to more accurately reflect real-world usage patterns. Specifically, the tasks to be performed during the load test were the following:

- login and authenticate in the system;
- list the files in the user's private storage space;
- read the content of a file contained in the user's private storage space;
- upload a 5 KB file to the user's private storage space;
- upload a 5 MB file to the user's private storage space;
- upload a 1 GB file to the user's private storage space;
- download a 5 MB file from the user's private storage space.

The `Python` script is available on the author's GitHub repository [1]; additionally, the `README.md` file provides detailed instructions on how to perform the load test.
The test was executed on a MacBook Pro M2, with *Locust* configured to spawn one user per second, up to a maximum of 50 concurrent users. Each second, all currently spawned users attempted to perform one of the defined tasks. The test was scheduled to run for a total of 120 seconds and was performed with server-side encryption enabled. The following chart displays the results of the *Locust* load test:

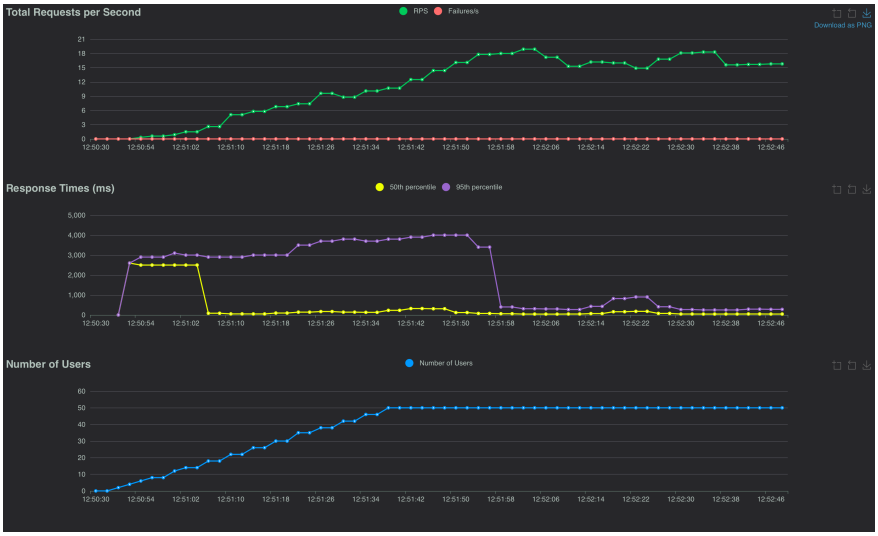

Figure 1: results of the *Locust* load test with up to 50 concurrent users

The results in figure 1 demonstrate that as more users were spawned, the system efficiently scaled to handle an increasing number of concurrent requests. Notably, the failure rate remained at 0% throughout the entire test, indicating that all requests were successfully processed without errors, even under higher loads. By examining the response times, it is evident that the system quickly stabilized. In fact, the median response time dropped significantly early in the test and remained consistent, showing that the system adapted well to the increasing load and efficiently allocated resources. Overall, this analysis revealed that the deployed system is robust, with good scalability and the capability to manage a growing number of concurrent users effectively. The absence of any failures and the stable response times suggest that the system could maintain reliable performance under stress.

1.5 Scalability and Cost-Efficiency

In a production environment, the load and traffic on the deployed *Nextcloud* instance would likely be significantly higher than those simulated during the load test. This section presents several solutions to scale the developed cloud-based file storage system, providing an analysis of their advantages, disadvantages and associated costs.

1.5.1 Cluster Deployment

If an organization already owns a cluster of nodes, it can deploy the *Nextcloud* platform on this existing infrastructure.
Advantages:

- Full control over the hardware and the infrastructure allows the organization to customize, configure, and scale the system to meet specific needs.
- Data remains within the organization's infrastructure, which is crucial for handling sensitive information, ensuring complete control over data security.
- The organization can adopt and implement its own policies and security measures without relying on third-party providers.
- Operational costs become more predictable and can be lower in the long run, as there are no ongoing fees associated with third-party services.

Disadvantages:

- The organization must bear the initial investment to purchase the hardware and all the required infrastructure, as well as the ongoing costs for maintenance and upgrades.
- Setting up and maintaining the infrastructure and the *Nextcloud* instance requires technical expertise, particularly for larger and more complex deployments.

1.5.2 Cloud Provider Deployment

If an organization does not already own a cluster and determines that purchasing the necessary hardware and infrastructure is not cost-effective, either due to the high initial investment or the unpredictable and fluctuating demand for resources, the *Nextcloud* system can be deployed on a cloud provider like AWS.

AWS (Amazon Web Services) offers virtually unlimited scalability, allowing organizations to dynamically adjust resources based on their specific needs. Whether the *Nextcloud* instance needs to handle a few users or thousands, AWS can scale resources up or down to meet the current demand. Additionally, it provides robust autoscaling features, ensuring that the deployed platform remains highly available and can handle increased traffic without downtime. Thanks to its composability, AWS allows organizations to scale only the resources that are necessary, optimizing both performance and cost. It also offers advanced security features, which are essential for maintaining the security and privacy of data. The wide range of services provided by AWS can be easily integrated with *Nextcloud*, such as S3 for handling large amounts of data with object storage and RDS for managed databases, which simplifies maintenance and scalability. Finally, AWS's pay-as-you-go model ensures that organizations only pay for the resources they actually use, making it an ideal solution for handling unpredictable and fluctuating workloads. AWS also offers tools for monitoring and optimizing costs, further enhancing its suitability for production environments.
While AWS is primarily recognized as an *IaaS* provider, an organization can choose to adopt a specific cloud service model (*IaaS*, *PaaS*, or *SaaS*) depending on its needs and objectives. Each model offers different levels of control, flexibility, and responsibility.
Advantages:

- The organization avoids the initial investment to purchase the hardware and all the required infrastructure.
- Cloud providers offer virtually unlimited scalability, enabling the organization to adapt resources to fluctuating and unpredictable workloads.
- The organization is relieved from managing and maintaining the physical infrastructure.
- The pay-as-you-go model can potentially lower costs, particularly for variable workloads.

Disadvantages:

- The organization must rely on third-party providers.
- Storing data in the cloud may raise concerns, particularly for sensitive information.
- Operational costs may become less predictable.
- The more services the provider manages, the higher the costs for the organization.
- As the provider takes on more responsibility, the organization has less control over the infrastructure.

The **Infrastructure as a Service** (*IaaS*) model provides the most control, allowing the organization to rent and manage the resources it needs on a pay-as-you-go basis, while still being responsible for managing, installing and configuring the system. The **Platform as a Service** (*PaaS*) model offers a platform where the organization only needs to manage the application and the data, without dealing with the underlying hardware infrastructure or the system installation. Finally, the **Software as a Service** (*SaaS*) model delivers fully managed software applications over the internet, with the cloud provider handling everything from infrastructure to application management, leaving the organization to simply use the system.

2 Cloud Advanced Exercise

2.1 Introduction

The assignment requires to redeploy the cloud-based file storage system using the *Kubernetes* container orchestration platform alongside the *Helm* package manager. Specifically, the system needs to be deployed on a single-node *Kubernetes* cluster (`k8s`), which must be created and properly configured.

2.2 Deployment

The deployment of the *Nextcloud* instance is based on the use of *Helm*. Specifically, the *official Nextcloud Helm chart* is employed, with appropriate custom values configured in a `values.yaml` file. This chart deploys *Nextcloud* pods, each containing three containers: the *Nextcloud* application, a sidecar container for cron jobs and an nginx web server container. Additionally, these pods are linked to a PVC (*persistent volume claim*) for storage, an external *PostgreSQL* database running in its own pod and connected to a separate PVC, and a *Redis* pod used for caching. Each pod is equipped with the necessary probes to handle potential issues. Moreover, a *MetalLB* load balancer is configured to expose the *Nextcloud* instance to the outside world via an external IP address. Finally, *Kubernetes Secrets* were used to securely manage sensitive information such as passwords and tokens. All the employed deployment files and configuration scripts can be found in the author's GitHub repository [1]. Additionally, a `README.md` file provides comprehensive instructions for building and setting up the single-node *Kubernetes* cluster, as well as deploying and accessing the *Nextcloud* service.

2.3 Back-End Storage

To prevent data loss in the event of pod crashes or accidental deletions, as mentioned in the previous section, *persistent volumes* (PVs) and *persistent volume claims* (PVCs) were employed to provide persistent storage for both the external *PostgreSQL* database and the *Nextcloud* pods. This approach ensures reliable, durable storage that remains intact even if individual pods are terminated or recreated. Specifically, a directory on the host machine was mapped to a volume in the *Kubernetes* cluster using a local path provisioner. However, this back-end storage choice comes with several limitations:

- **Single Point of Failure**: if the host machine fails, all data stored on its local path is lost, as the storage is not replicated across multiple nodes.
- **Lack of Dynamic Provisioning**: local path storage does not support dynamic provisioning, meaning PVs must be manually created before deploying the service.
- **Node Affinity Constraints**: PVs are tied to a specific node, so the pods relying on them must run on that same node. This limits the ability to scale the application horizontally across multiple nodes.
- **Limited Scalability**: the storage capacity is limited by the physical storage available on the host machine, which can constrain the ability to scale the application as demand grows.

Given these limitations, local path storage is typically suitable for development, testing, or small-scale production environments where data persistence is important, but high availability and scalability are not critical requirements. For production environments that

require higher resilience and scalability, distributed storage solutions like *Ceph* might be more appropriate.

2.4 High Availability Considerations

In a production environment, achieving high availability for the deployed file storage system is crucial to minimize downtime and ensure continuous access to the service. To accomplish this, the following steps should be taken:

- **Adopt a Distributed Storage Solution**: as discussed in the previous section, replace the local path storage with a distributed storage system like *Ceph*. This approach ensures that data is replicated across multiple nodes, significantly reducing the risk of data loss in the event of a node failure.
- **Enable Dynamic Provisioning**: implement storage classes that support dynamic provisioning, allowing *Kubernetes* to automatically create and manage *persistent volumes* (PVs) as needed. This not only improves scalability, but also enhances the flexibility of the storage system.
- **Expand to a Multi-Node Cluster**: deploy the service on a multi-node *Kubernetes* cluster to distribute pods across different nodes. This setup ensures that if a node fails, the affected pods can be rescheduled on another available node, maintaining service continuity.
- **Deploy Multiple Replicas**: deploy multiple replicas of the service to ensure that if one instance fails, others continue to handle requests. The *official Nextcloud Helm chart* provides a `replica` field that can be configured to deploy multiple replicas.
- **Improve Load Balancing**: implement an *Ingress Controller* to manage and route external traffic to the services efficiently, ensuring that requests are balanced across available pods.
- **Horizontal Pod Autoscaler (HPA)**: configure the *Horizontal Pod Autoscaler* to automatically adjust the number of pod replicas based on current load or resource utilization, ensuring that the service can handle varying levels of demand. The employed chart also allows to efficiently set up and enable HPA.

2.5 Comparison with the Docker Solution

In the `docker-compose` deployment, services were deployed as standalone containers, each running independently and managed directly by `docker`. In contrast, the *Kubernetes*-based solution deploys these services within pods, where the containers are managed by a container orchestrator. The *Kubernetes* deployment offers a more scalable, flexible and robust environment; in fact, it allows the service to be easily scaled by either adding more nodes to the cluster or by increasing the number of replicas. Moreover, *Kubernetes* automates key tasks such as scaling, load balancing, and self-healing, thereby managing the deployed service more efficiently. As discussed in the previous section, the system can achieve high availability, making the *Kubernetes*-based solution the most suitable for large production environments. However, the *Kubernetes* deployment is more complex to set up and can require more resources. In fact, while the `docker-compose` solution has no minimum requirements, machines running *Kubernetes* must be equipped with at least 2 CPUs and 2 GB of RAM. Therefore, the `docker-compose` solution still remains a valid

choice for development, testing and small-scale production environments, particularly due to its simplicity in setup and management.

3 References

1. Piero Zappi, `Cloud_Computing_FinalProject` GitHub repository. URL: `https://github.com/PieroZ01/Cloud_Computing_FinalProject/tree/master`