Cloud Computing Report

Roberta Lamberti [SM3600007]

April 2024

Contents

1 Introduction

This project intends to evaluate the necessity of a Cloud Storage File System, employing containerization techniques via Docker and Docker-Compose. The

system is designed to facilitate user interactions such as uploading, downloading, and deleting files while maintaining the privacy of individual user spaces. Additionally, it provides administrators with capabilities for operations and management of the system's architecture without granting them access to the private spaces of users. To avoid building from the ground up, I opted for pre-existing Docker utilities, specifically choosing the Nextcloud image to manage the web user interface.

2 Nexcloud

Nextcloud is a powerful, non-commercial cloud software for self-hosting and cloud computing, suitable for both personal and business use.
Among the suggested options, Nextcloud stood out as the most compelling platform for several reasons. The primary factor is the ease of implementation, particularly when utilizing Docker containers.
Nextcloud issupported by an extensive documentation and, being an open-source project, it benefits from a vast community of developers and users, which greatly streamlined the whole implementation process. Neverthless, Nextcloud provides an extensive array of features that extend beyond mere file storage. It enables the developer to implement robust security protocols and integrate various other software and services, making the system highly adaptable and appropriate for a diverse range of use cases.

2.1 User authentication, authorization and file operations

One of the main advantages offered by Nextcloud is its adaptability and versality in creating users. The requirements about user authentication and authorization are guaranteed:

- **Sign-in up and login**: Nextcloud comes with an integrated user management system that facilitates registration, sign-in, and sign-out pro-

cesses. The system is crafted to be intuitive, allowing users to effortlessly maneuver through the platform.

- **Role Based Access Control:** Nextcloud features role-based access control (RBAC), enabling the designation of various user roles like regular users and administrators. This capability is essential for regulating access within the system, making sure that administrators possess the authority to oversee user accounts, and that regular users maintain privacy within their own storage areas.

- **Private Storage:** Nextcloud automatically provides each user with a dedicated private storage space, guaranteeing built-in isolation among the different user accounts right from the start.

- **Admin management capabilities:** Administrators can utilize the Nextcloud admin interface to handle user accounts effectively, which encompasses tasks such as creation, modification, and removal of user accounts, in addition to allocating roles and permissions. Generally, this functionality is vital for upholding the system's integrity in terms of security and user management.

Nextcloud is a outstanding tool even in file handling capabilities, providing users with a secure and user-friendly process for uploading files to their designated private storage areas. With an user-friendly interface, individuals can effortlessly upload files from their devices. Similarly, downloading files is made easy; users can retrieve their files from private storage through the Nextcloud interface to their local devices. Additionally, users have the option to delete files from their storage, a functionality that is smoothly integrated into the user interface, allowing for efficient file management and ensuring users retain only the data they require.

2.2 Address Security

Nextcloud is equipped with an array of security measures that administrators can easily configure via the admin interface. All security options are accessible through the Admin settings in the security section and can be activated with minimal effort. Through experimentation with these settings for this exercise, it was clear that while they are not imperative for the core functions, they are strongly advised for practical deployments. In real-world scenarios, safeguarding sensitive data is critical, and these settings must be enabled and meticulously adjusted to ensure the utmost level of security.

3 Deployment with Docker

Following the guidelines provided by this *repository*, the deployment of Nextcloud has been achieved using docker and docker Compose. The scripts and configuration files are available at this different *repository*. The images used for the deployment are listed below:

- **Nextcloud:** This setup launches the Nextcloud application and is tailored through environment variables that define database connections, administrative credentials, and additional configurations. The image exposes port 8081 to grant access to the Nextcloud web interface.
- **Mariadb:** This image deploys a MariaDB database server, serving as the backend storage solution for Nextcloud. Configuration is managed using environment variables to establish the root password, database name, and user credentials.
- **Locust:** An attempt to build a locust image inside a container has been tried, but some communication issues between the containers made it not possible. Locust has been run locally on the port localhost:8089.

These Docker images collaborate within a containerized environment to

deploy and test the Nextcloud application along with all its related services. To launch the Docker containers, it's necessary to create a *docker-compose.yaml* file with all the desired features configured. After that, simply navigate to the docker-compose.yml file and execute the following command:

```
docker-compose up
```

This command will deploy the container on the chosen port, where the user or the admin can access to the private storage area. The admin has access to configure the password and the permissions. After building the container, it's possible to set eventual 2-factor autentication, minimum length password requirements and many other features in the Security settings by the admin. All these steps are necessary before creating the users through this file: *create nextcloud users.sh*.

3.1 Testing

To assess the performance of the Nextcloud infrastructure, load testing have been conducted using Locust, a tool that performs the simulation of concurrent user activity on the system. Developers can craft tests using standard Python code, and Locust offers comprehensive reports on how the Nextcloud instance performs under these simulated conditions. As discussed above, building the locust image inside a container has not been possible, even if several attempts have been done. The locust has been opened on a page on the browser on the port 8089. Operations of download, get and delete are performed by the *test.py*

Figure 1: Testing of a small file, a .jpg

0.5

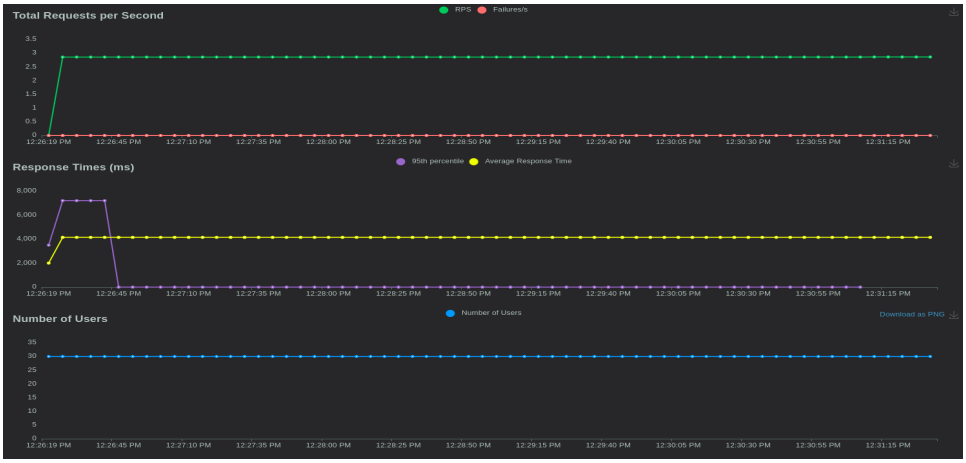

Figure 2: Testing of a medium file, a .pdf

0.5

Figure 3: Testing of a generated file of 1gb

The number of users accessing to the cloud has been set in the first case has been set to 30/50 (with the create nextcloud users.sh 50 users have been created, including the admin). TRS has been set to 15. In the first plot we can notice how the failures are homogeneously low, but still present. The average reponse time is quite low, so that means the requests are answered quite fast. While this is what happens for a small file, it's not the same for the medium and large files. In second and the third cases the number of users has been set to 45, with trs set to 30. It's possible to notice how the failures stay costantly set to zero in both the cases, and the same happens for the average response time, that is quite higher compared to the small file. Considering that the size of the files has increased, it's fair it takes a bit more of time to perform all the required operations.

4 Ex. Advanced 01 -Deployment with Kubernetes

This part of the report concerns the advanced module of the Cloud Computing course. The assignment consists in deploying the same cloud-storage system but in Kubernetes, using helmchart. Specifically, it's required to create a one node kubernetes cluster using K8S, deploying the services through helmchart. In this case, instead of K8S, the choice falls on K3s, a light-weight version of K8s. The entire configuration is available at this *repository*.

4.1 What is Kubernetes?

Kubernetes is an open-source container orchestration system for automating software deployment, scaling, and management. While Docker is a container runtime that lets developers build, ship, and run containers, Kubernetes offers an additional game-changer feature: container orchestration. Mixed up together, they make possible to containerization of applications and their systems at scale.

4.2 What is Helmchart?

A Helm chart is a package that contains all the necessary resources to deploy an application to a Kubernetes cluster. This includes YAML configuration files for deployments, services, secrets, and config maps that define the desired state of your application.

4.3 Implementation

The implementation of the first exercise has been done through simple commands. After installing k3s and helmchart following the tutorials on their documented websites, I have simply run the following command:

```
helm install -f values.yaml ex1 nextcloud/nextcloud
```

This is the file with the values needed for building a cloud storage system. Modifying the options inside the *values.yaml* and enabling mariadb from False to True. Other options like PostgreSQL were available, but mariadb has been chosen for a consistent comparison with the previous assignment. The cloud storage system is deployed on Nextcloud on the page http://localhost:8080.

5 Ex. Advanced 02 - MPI @ Kubernetes

The exercise requires to run the OSU benchmark inside the containers held by the pods that in the nodes. The nodes talk via flannel. Two different scenarios have been performed: in the first one, the pods (launcher, worker01, worker02) are on the same node, while in the second scenario the pods are distributed among three nodes (laptophost, nodo2, nodo3). A benchmark to evaluate the latency has been run to show the results delivered by the tests. A benchmark to evaluate the latency has been accomplished to deliver results. The exercise addresses the task of porting simple HPC workflows into Kubernetes by testing the student proficiency and capabilities not only in Kubernetes but also with Docker and HPC workflows. The student is required to create a container with

the OSU Benchmark. The container must have a behavior as expected by the operator.

All the configuration is available at this *repository*.

5.1 Configuration of the nodes

Two nodes have been created to perform this task. The VM have been created through GNOME Boxes and then managed through Virtual Machine Manager. The name of the VM are respectively nodo02 and nodo03. Considering the laptop host (the main machine), we have three components inside the cluster. The three nodes have been connected through a .xml document, kubernet.xml, where IP adressess and the virtual bridge has been set. The virtual bridge, vbr01, has been activated running this command:

```
sudo virsh net-start kubernet
```

In this way, the nodes and the host are connected to each other. After this step, it's necessary to activate the nodes (nodo02 and nodo03) and set through the Virtual Machine Manager GUI in the information, in the NIC (network interface), the bridge device that is vbr01.

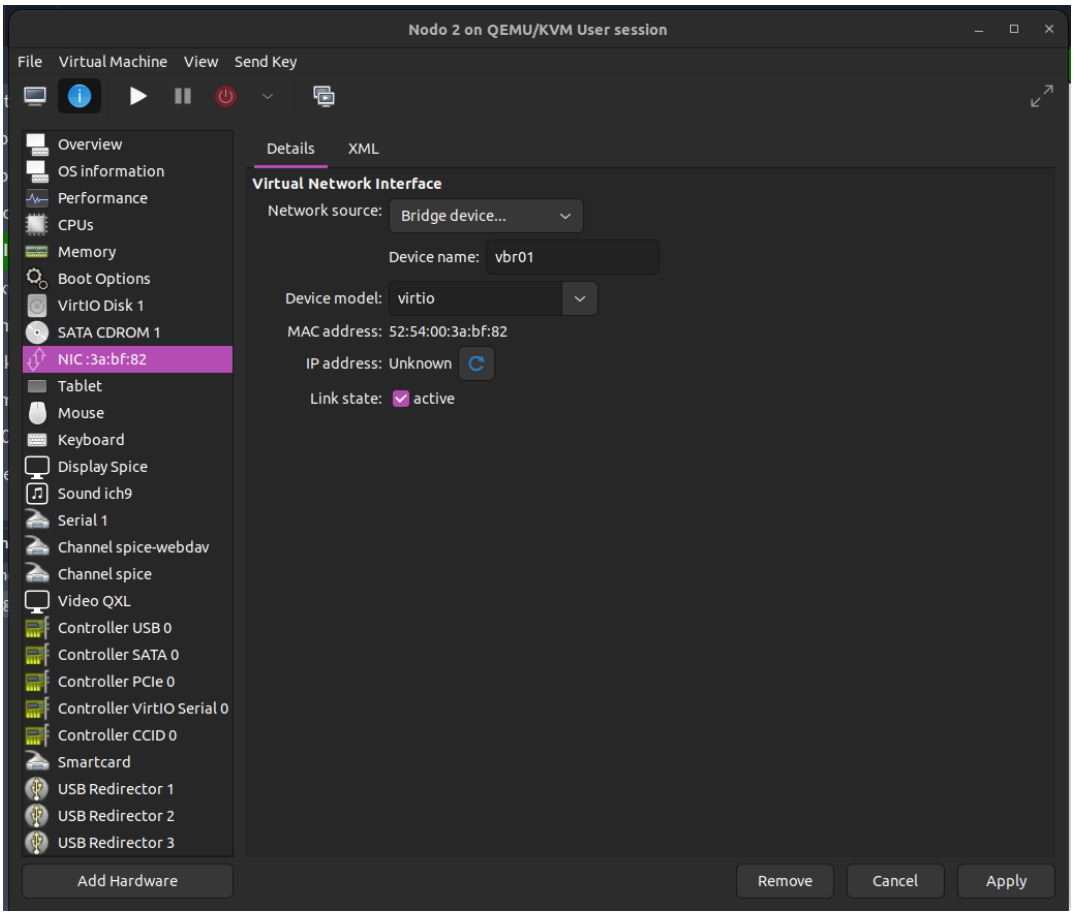

Figure 4: Setting the device bridge into the NIC

After we have made sure the nodes are connected, we can install the MPI Operator on the entire cluster, just running this command from the master. This is possible thanks to one of the best features of Kubernetes, that is kubectl, the command-line tool. It allows the user to run commands against Kubernetes cluster. It's possible to deploy applications, inspect and manage cluster resources, and view logs.

```
kubectl apply con il link sudo kubectl apply -f https://raw.githubusercontent.
```

At this point, it's necessary to create the DockerFile, in which we have put the osu benchmark of MPI Operator. Then we build the image of the container locally with Docker, running the following command:

```
docker build -f Dockerfile -t osuimage .
```

This step needs to be done manually on every node, not only on the master, unfortunately. In this way we have created the image locally. The next step is to create two .yaml files, so that these images could be used to create the containers inside the pods that will be distributed among the nodes. Clearly, the image will be build on all the nodes, not only on the master.

- **FIRST SCENARIO**: the pods are distributed on one node only.

```
apiVersion: kubeflow.org/v2beta1
kind: MPIJob
metadata:
  name: osu-single-node
spec:
  slotsPerWorker: 1
  runPolicy:
    cleanPodPolicy: Running
  sshAuthMountPath: /home/mpiuser/.ssh
  mpiReplicaSpecs:
```

```
Launcher:
  replicas: 1
  template:
    metadata:
      labels:
        app: osu
    spec:
      containers:
      - image: osuimage:latest
        imagePullPolicy: Never
        name: osu-launcher
        securityContext:
          runAsUser: 1000
        args:
        - /entrypoint.sh
Worker:
  replicas: 2
  template:
    metadata:
      labels:
        app: osu
    spec:
      containers:
      - image: osuimage:latest
        imagePullPolicy: Never
        name: osu-worker
        securityContext:
          runAsUser: 1000
        command:
        args:
```

```
  - /usr/sbin/sshd
  - -De
  - -f
  - /home/mpiuser/.sshd_config
affinity:
  podAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
    - labelSelector:
        matchExpressions:
        - key: "app"
          operator: In
          values:
          - osu
      topologyKey: "kubernetes.io/hostname"
```

- **SECOND SCENARIO**: the pods are distributed among two nodes.

```
    apiVersion: kubeflow.org/v2beta1
kind: MPIJob
metadata:
  name: osu-double-node
spec:
  slotsPerWorker: 1
  runPolicy:
    cleanPodPolicy: Running
  sshAuthMountPath: /home/mpiuser/.ssh
  mpiReplicaSpecs:
    Launcher:
      replicas: 1
      template:
        spec:
          containers:
          - image: osuimage:latest
            imagePullPolicy: Never
            name: osu-launcher
            securityContext:
              runAsUser: 1000
            args:
            - /entrypoint.sh
    Worker:
      replicas: 2
      template:
        metadata:
          labels:
            app: osu-worker
        spec:
```

```
containers:
- image: osuimage:latest
  imagePullPolicy: Never
  name: osu-worker
  securityContext:
    runAsUser: 1000
  command:
  args:
  - /usr/sbin/sshd
  - -De
  - -f
  - /home/mpiuser/.sshd_config
```

After creating these two .yaml files, it's possible to transofrm them into objects in Kubernetes, running this command:

```
sudo kubectl apply -f onenode.yaml
sudo kubectl apply -f twonodes.yaml
```

At the end of the entire exercise, two files of benchmarking test have been produced. They are available at onenode.txt and twonodes.txt. The operations performed with the MPI operator are latency and osu scatter.