# Cloud Computing WS20/21

## Setup of Nextcloud using Kubernetes

Michael Zodel 1090297

Andrej Korinth 1367632

Francisco Seipel-Soubrier 1059186

Nils Krug 1367470

## Table of contents

1. Introduction ........ 3
1.1 Nextcloud ........ 3
1.2 Kubernetes ........ 4
1.2.1 Kubernetes configuration files ........ 5
1.3 High availability cluster ........ 6
2. Cluster architecture ........ 7
2.1. Namespace ........ 8
2.2. Secret ........ 8
2.3. MinIO ........ 8
2.4. Redis ........ 9
2.5. Database ........ 9
2.5.1 MySQL ........ 10
2.5.2 MariaDB ........ 10
2.5.3 MaxScale ........ 11
2.6. Nextcloud ........ 11
2.7. Cron ........ 12
2.8. Kustomize ........ 12
2.9. Continuous Delivery ........ 13
2.9.1 Image Building ........ 13
2.9.2 Deploying ........ 14
3. Setup of Kubernetes ........ 14
3.1. Preparation of the nodes ........ 14
3.2. Installation of the necessary plugins ........ 15
3.3. Setup on the master ........ 15
3.4. Setup on the workers ........ 17
4. Deployment of Nextcloud on a Kubernetes cluster ........ 17
5. Conclusion ........ 19
6. Source code ........ 20
7. Sources ........ 22

# 1. Introduction

The following documentation aims to give a basic understanding of Kubernetes and its functional subparts. The deployment of a Nextcloud web application is used as a real world example. A step by step guide is created for allowing everyone to create a cluster for his or her own use and further testing. In the last chapter we reflect our findings and give an outlook on possible further developments.

## 1.1 Nextcloud

The market for cloud solutions is dominated by American companies in the likes of Microsoft, Google and Amazon. Cloud solutions offer great scalability and fast deployment while also having lower running costs [1]. Big companies like Volkswagen and EON are using these services for their business processes [2].

The 2013 Edward Snowden leak showed, that American companies are tightly connected to government bodies and sharing user data for analysis [3]. These practises are not supervised by the public and the damage for the privacy of its users is difficult to measure. Despite its initial impact, very little has changed regarding the protection of personal privacy. Therefore an alternative is needed for the data conscious user.

Nextcloud started as a fork of Owncloud in 2016 after some of team members including the founder left their previous company. The enterprise part of Owncloud is closed source while all features of Nextcloud are open source. Since its launch the popularity of Nextcloud grew continuously as seen in figure 1, making it one of the biggest alternatives to Amazon and co.

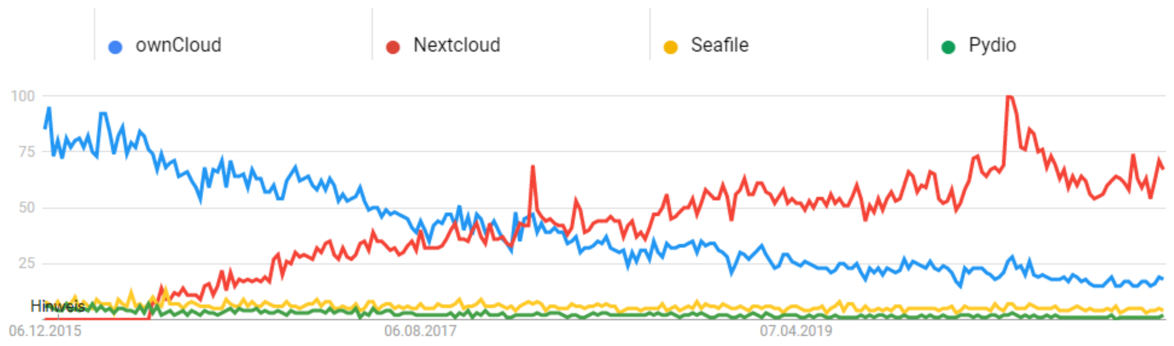


*Figure 1 Google Trends for alternative cloud solutions [4]*

In 2019 Major contracts with Governments have been closed supplying cloud services for several hundred thousand government workers in France, Germany, Sweden and the Netherlands [2]. Due to its open source nature, implementing backdoors for surveillance is unlikely.

Since 2016 the amount of functions increased steadily. While the core function is still sharing files, a text and video chat option has been added. Furthermore several groupware functions have been added, synchronizing calendars and task lists. With plugins for Microsoft Teams, GitLab, Slack, Twitter and many more Nextcloud can be adapted to many different needs [5].

## 1.2 Kubernetes

Kubernetes is a portable and scalable open-source-platform to manage container-based applications and services. It eases up the configuration as well as the automation. Joe Beda, Brendan Burns and Craig McLuckie founded Kubernetes. Later on more Google employees collaborated [6]. 2014 Kubernetes has been announced [7]. Version 1.0 has been published on the 21st July 2015. On that day the Cloud Native Computing Foundation was founded and Kubernetes was donated to it. Since then it remained open source [8].

Kubernetes coordinates computer-, network- and storage infrastructure. It has aspects of the platform as a service (PaaS) as well as infrastructure as a service (IaaS). It also allows the portability between different infrastructure providers.

The main components of Kubernetes are the master and the node components as seen in figure 2. The master component is the control area of the cluster. On the one hand it makes decisions for the cluster such as time scheduling. On the other hand, it identifies and reacts to actions in the cluster, for example the shutdown of one node component. On node components the work for the application or the service is done.

The system orchestrates so called pods as smallest deployable unit on the nodes. Each pod contains one or several containers. These containers share the storage and network resources of the master node. Each container has its own specification on how to run the container. To access the pod, they have a unique IP-address. However, each time a pod fails a new one has to be created and a new IP-address is created as well.

For different pods to communicate as well as to communicate with the users, services are created. The services manage different port forwarding and support the pods to get the relevant data from others. An example for a service is a load balancer, which analyses the volume of workload on different nodes and automatically distributes the workload evenly.

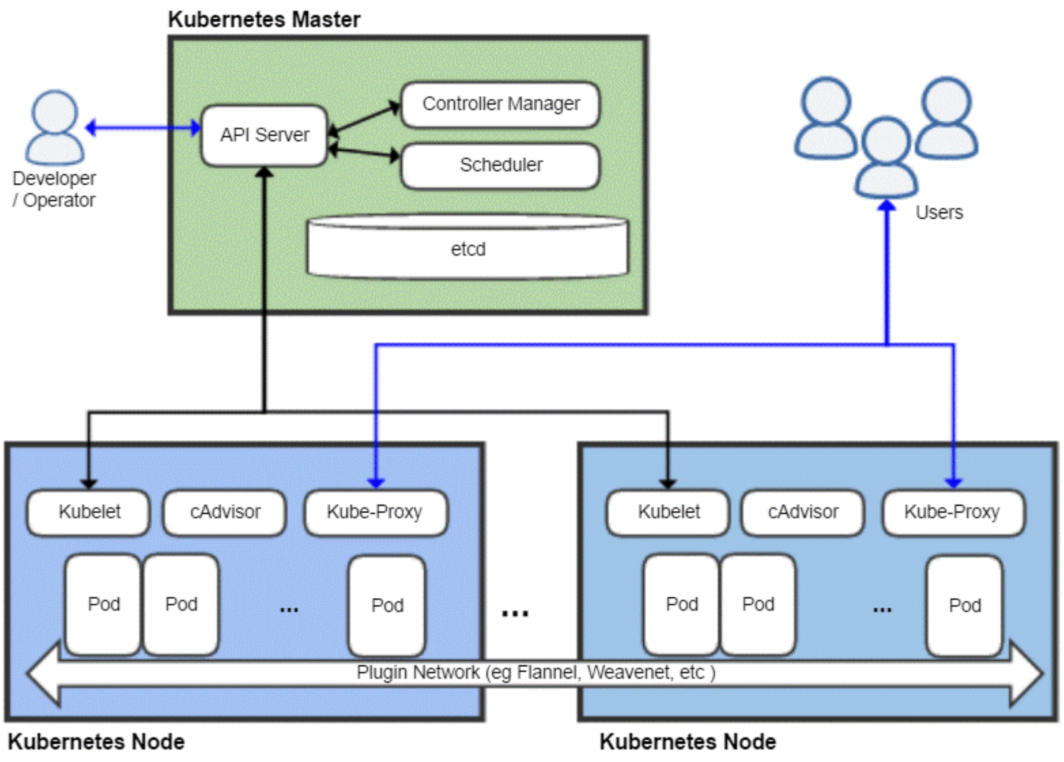


*Figure 2 Kubernetes architecture [8]*

### 1.2.1 Kubernetes configuration files

Kubernetes configuration files are of the type .yaml and therefore are called Yaml files in the following. They are used to deploy different objects inside Kubernetes. With these files it is easy to document the deployed services and pods together with changing things for the deployment.

All files can be interpreted by kubectl by converting them into JSON files. This is done automatically by Kubernetes and is not present to the user. Based on the current structure of the cluster the applied files change an existing object or create new ones.

The base structure of any yaml file for Kubernetes is as following:

```
apiversion:
kind:
metadata:
spec:
```

The ‘apiversion’ specifies the version of Kubernetes API which is used to create the object.

The `kind` describes what kind of object should be created. Examples are `Deployment` for the creation of pods or `Service` which exposes running pods inside the cluster.

The `metadata` is describing data which helps to uniquely identify the object. It often contains of a name, a unique identificator as well as a namespace.
The ‘spec’ is then used to specify the kind of object you want to create. This is different for each kind and has different sub categories based on this. For example a database deployment needs a user and a password as well as a service to connect to. A service on the other side only needs a port.

To create an object based on a file in kubectl you have to use the `kubectl apply` command. It is used as following:

```
$ kubectl apply -f path/to/yaml/file.yaml
```

The option ‘-f’ indicates, that we are applying a file.

### 1.3 High availability cluster

A high availability cluster (HA cluster) refers to a group of hosts or systems therefore called nodes which act as a single system. Seeing a cluster as a single system will allow the system to refer users to other nodes seamlessly and without any downtime in case of a failure. There are different ways HA clusters can be configured.

In case of an active-active HA cluster, every node will simultaneously process data for the users, which is called load balancing, to optimize network efficiency and increase throughput and response times. Therefore the users are balanced between every node of the system. A downside is the increased cost in comparison to active-passive HA clusters. The load balancing algorithm depends on the settings of the load balancer. In case of a Round Robin algorithm with 2 nodes for example, the first user may use the first node, the second user the second node, and the third user will then have to use the first node together with the first user.

In active-passive HA clusters a backup node only starts processing once a failure at the active node is detected, which results in significantly less network efficiency and a lower cost compared to active-active HA clusters, since the users are only using a single node together. The reduced cost stems from the fact that not many backup nodes will be needed at all, since total failure with multiple backup nodes is less likely.

HA clusters differ in size with a minimum of 2 nodes. All nodes must have access to the same shared storage to allow fail over to another host [9].

## 2. Cluster architecture

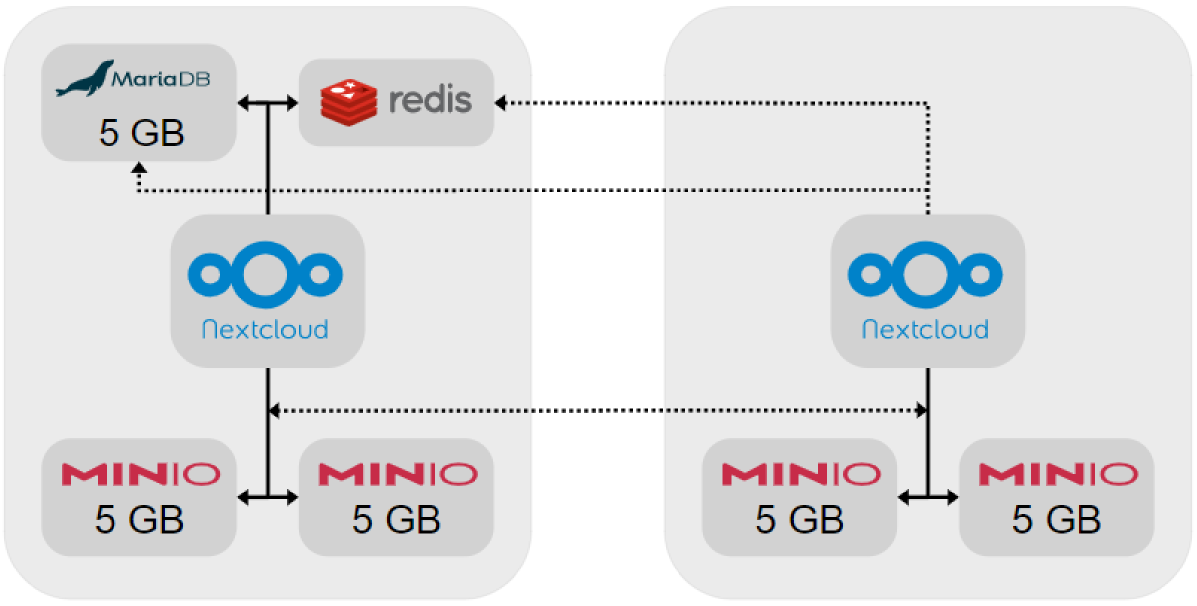


*Figure 3 cluster diagram*

The main objective of the architecture for this cluster was scalability, storage distribution and high availability. As seen in figure 3, every node has its own Nextcloud worker and part of the storage. These node can be scaled horizontally infinite, because the Nextcloud worker is stateless and MinIO has no limit on the number of servers it can manage.

One node on top of that contains the databases. This is the only factor that limits the scalability of the system, as every other node needs to access the data of these databases.

There are two factors that can result in a failure of the system. If the node with the databases is down then the complete system is down. The other is the MinIO object storage, it has replication of data build in. There is no data loss and it will continue to operate as long half of the nodes are online.

As every node can handle requests on port 80, it is possible to use a load balancer with health check redirecting requests to any server in the cluster.

### 2.1. Namespace

The ‘namespace.yaml’ creates a new object of kind ‘Namespace’ with the name ‘Nextcloud’ inside the cluster. With this the other objects can be created inside the namespace to distinguish between different applications, if deployed in the same cluster.

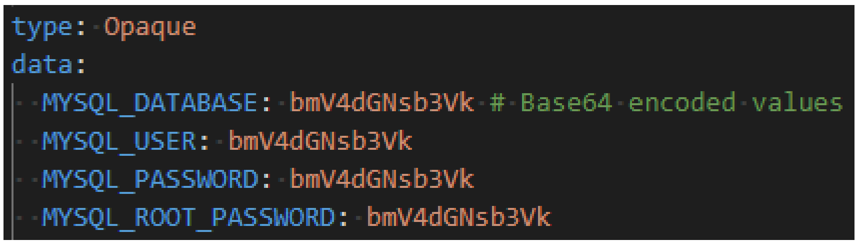
Special about this yaml file is the fact, that a ‘spec’ option is not needed, because the metadata → kind option is used to specify the name of the namespace.

### 2.2. Secret

The ‘secret.yaml’ contains the admin user and password for Nextcloud as well as MySQL credentials. All values inside this file are base64 encoded to not be stolen easily. An example of such encoded values can be seen in figure 4.

These secrets should be changed on every system in order to prevent attackers from getting access to the cluster.

The kind Secret is also slightly different than the general structure. A type of the secret is described as well as different variables inside the data option.



```yaml
type: Opaque
data:
  MYSQL_DATABASE: bmV4dGNsb3Vk # Base64 encoded values
  MYSQL_USER: bmV4dGNsb3Vk
  MYSQL_PASSWORD: bmV4dGNsb3Vk
  MYSQL_ROOT_PASSWORD: bmV4dGNsb3Vk
```

*Figure 4 example snippet of secret.yaml*

### 2.3. MinIO

MinIO is used to store the data of the Nextcloud deployment. Inside the MinIO service only the files are saved, all metadata and the file structure are stored inside the

Nextcloud database. For the deployment of MinIO secrets and tenants are needed. The secrets describe the MinIO credentials as well as the console keys. These are then used inside the tenants. Tenants are a special kind of object which are not present in default Kubernetes. To create such an object you have to install MinIO on the cluster first to use it. The tenant then can be created with the yaml file. Inside the file the amount of replicas is set to two. Based on this two console objects are created for MinIO.

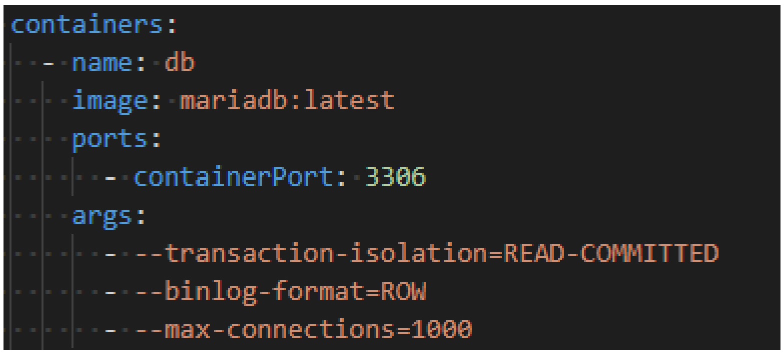
Kubernetes automatically detects the amount of servers available in the cluster as well as their workload and then evenly distributes all replicas across the cluster. Inside the tenant a persistent volume claim is created to store data not only for the runtime but independent of the pod.

### 2.4. Redis

Redis is a modern memory cache that can be used for distributed caching and as key-value store for avoiding file corruption during normal operations. It includes a service and a single pod, which is used for all servers.

### 2.5. Database

The database is needed to store administrative data for Nextcloud. Supported systems are MySQL, MariaDB, PostgreSQL or an Oracle database. Recommended by Nextcloud are the first two options. In the project three different setups haven been tried. One with MySQL replication, one with MariaDB and one with MaxScale. In the current setup MariaDB is used. In the following the yaml files of three setups are described.



```yaml
containers:
  - name: db
    image: mariadb:latest
    ports:
      - containerPort: 3306
    args:
      - --transaction-isolation=READ-COMMITTED
      - --binlog-format=ROW
      - --max-connections=1000
```

*Figure 5 example snippet of deployment.yaml for MariaDB*

### 2.5.1 MySQL

The MySQL setup includes a StatefulSet of the database, which automatically forward the included data to all nodes. The setup is done with a persistent volume claim, a ConfigMap, a service as well as the StatefulSet.

First of all the persistent volume claim is setup to store the data for the database. After this the ConfigMap is created. It is used to configure the master and slave nodes in the cluster. The master is allowed to read and write, the slaves are only allowed to read. After this the service is created which is then used in the StatefulSet deployment.

The StatefulSet deployment is a special kind of deployment. In a StatefulSet the pods are initialized one after the other. First the master is created and afterwards the children. To clone the data and to set the config different bash commands are used. First the initialization of the StatefulSet takes place. The server id is generated and the appropriate config file from the ConfigMap are copied to the server.

After the generation of the id and the copying of the correct ConfigMap the data is copied from the previous node. So the first child copies the data from the master, the second child from the first child and so on. After this the container for the MySQL image is set up. It contains information like the database name, the password, the user and the port to connect to. It also includes a liveness probe and a readiness probe, which determines the configuration for those two probes. Additionally an extra backup for the data is created.

Unfortunately for this approach to work read and write has to be split on application level, which Nextcloud does not support. This means it can only be used as real-time database backup and not for load balancing.

### 2.5.2 MariaDB

The MariaDB setup uses one database for all nodes to connect to. For this setup a persistent volume claim, a service as well as the deployment are created.

Like the MySQL setup, first of all the persistent volume claims are created. Also the next step is to create the service, which has no special things inside.

The last step in the setup is to create the MariaDB deployment. The replicas are set to one and the strategy is set to recreate. This means on deployment the old version is terminated and replaced by the new deployment.

Besides the information about the database name, the password as well as the user the setup also needs some additional arguments. The additional arguments set are the transaction isolation, the binlog format as well as the maximum amount of connections, as shown in figure 5.

The transaction isolation is set to READ-COMMITTED, which means, that for every read, its own fresh snapshot is set and read. The binlog format is set to ROW, so for every update, delete or create a new log is created. Additionally DML statements are not logged.

The maximum amount of connections is set to 1000, to not use too many connections simultaneously.

#### 2.5.3 MaxScale

MaxScale is a database proxy for MariaDB replications. The difference between MYSQL replication and MaxScale is, that it can automatically route write operations independently from Nextcloud.

The folder 'mariadb_maxscale' contains the configuration to setup MariaDB for MaxScale. It is very similar to the normal MariaDB setup with the difference that it is a StatefulSet and starting the DB in replication mode.

MaxScale is configured in a ConfigMap which is loaded inside the deployment. As we have two servers the MaxScale replication is set to two. This is the only setup that can be used as a load balancer and failover for the database.

Unfortunately the latency performance of this approach is very poor resulting in the Nextcloud UI responding unreasonably slow.

### 2.6. Nextcloud

Nextcloud is the main application running inside the cluster. To deploy Nextcloud a persistent volume claim, the deployment, a service and an ingress are needed.

The persistent volume claim is similar to the previously described definitions. The service is a simple service only including a port.

In the deployment the usual information are configured like the container port or the image and the labels. The deployment should have two replicas, so each node has one replica. In the environment the needed variables for Nextcloud are set. The database is set to be MariaDB or MySQL in the given situations, as well as the database name and the user are configured. Additionally a Nextcloud user is set up. The other thing that takes place, is the configuration of MinIO as data storage. It is set as OBJECTSTORE_S3_HOST and a key and secret are entered, to establish a safe connection to the storage.

The ingress is used to make the pods available from outside. It manages the external access to a service in a cluster. The created ingress is set up for http requests and forwards the user to the service with the port 80. This is the created service for the Nextcloud deployment, which results in the user seeing the application Nextcloud in the browser window.

## 2.7. Cron

Cron is a background system scheduling jobs regardless of user interactions. It is set up to run jobs in the background regularly as well as not interfere with the performance of Nextcloud. These task could be for example database clean-ups. The jobs are usually command or shell-based scripts and are scheduled to run periodically at fixed times, dates or intervals. The setup of Cron includes only the deployment of the app.

In the deployment it is configured, that the amount of replicas is set to one, additionally basic information like the labels and the image are set. To execute background jobs, the shell script cron.sh is executed.

## 2.8. Kustomize

Kustomize is a command-line tool which allows a different approach to configuration customization regarding Kubernetes YAML files.

Normally, the YAML files have to be copied and edited manually. Any changes will be permanent and change the standard configuration file. Kustomize instead lets you compose every resource together into a single YAML file. Patches can then be applied to customize the configurations while leaving the original YAML files untouched. Kustomize also provides methods to make customization easier, like generators.

First, at least one base has to be created. A base is a directory which includes several resources in the form of YAML files together with a ‘kustomization.yaml’ file. The ‘Kustomization’ file groups the resources together and allows further customizations. Optionally, an overlay can be added. An overlay is another directory with a ‘kustomization.yaml’ file which purpose is to compose bases together while allowing further customization options.

Patches can be included to change the configuration of the resources without changing the files themselves. A YAML file containing the patch is first placed in the same directory it is to be used in. The ‘Kustomization’ file in the same directory is then edited to include the patch.

In the end, the build command can be used to build the new YAML, which contains every other file and their configurations. This file can then be used further, e.g. pipe it directly into kubectl.

In this project, Kustomize was used to compose the different resources of every technology together into ‘Kustomization’ files. These bases were then further combined into an overlay, built into a single YAML using the build command, and then used with kubectl. While customizations inside the Kustomization files and patches could have been included in this project, it was decided that they were not needed due to the small scope of the project.

## 2.9. Continuous Delivery

Continuous Delivery (CD) makes it possible to deploy changes automatically to the server. For this GitHub Actions are used as the pipeline. The configuration can be found in the GitHub folder and are automatically triggered when pushing a commit to the master branch.

### 2.9.1 Image Building

The first task of the deployment is to build a Nextcloud image and save it into the GitHub Container Registry using the Docker file in the folder ‘dockerfiles/nextcloud’. For Kubernetes Docker images have to be build and hosted outside of the cluster in registries.

For the deployment to work the secret ‘CR_PAT’ has to be set in the secrets section of the GitHub repository. It should contain a `Personal Access Token` with the permission `repo` and `write:packages`, which can be create in the GitHub User settings in the developer section.

### 2.9.2 Deploying

The Second task is to deploy the yaml configuration into the Cluster on the server. Only the files that are built by Kustomize are deployed, as they contain no cluster specific configuration like secrets or volume size configurations.

The kubeconfig is needed for the upload to the Server, which has to be set as a string inside of the ‘KUBE_CONFIG_DATA’ secret in the GitHub repository.

# 3. Setup of Kubernetes

For this setup guide two Ubuntu 20.04 VMs running on VMware Workstation 16 were used. A different virtualization tool can be used, but it needs to be ensured, that the VMs get static IPs. Each node should have at least 2 Cores and 4 GB of RAM for Nextcloud to be running properly. The setup guide is based on several different tutorials. Heman Sharmas guide was used as a foundation [10]. Dimuthu de Silvas explanations were used for setting up Calico [11]. The configuration of Nextcloud is based on Ali Cheatos blog post about deploying Kubernetes with Kustomize [12].

## 3.1. Preparation of the nodes

Open the Terminal with ctrl+T. Set terminal to super user

```
# sudo su
```

Update the system

```
# apt-get update
```

Disable swap file (necessary for starting a Kubernetes cluster)

```
# swapoff –a
```

Comment out the entry for the swap file with '#' in /etc/fstab

```
# nano /etc/fstab
```

Set the hostname according to its role (kmaster of master and kworker_ for each worker). Nodes cannot have the same hostname.

```
# nano /etc/hostname
```

### 3.2. Installation of the necessary plugins

Install OpenSSH-Server if a remote connection is necessary

```
# apt-get install -y openssh-server
```

Install Docker.IO

```
# apt-get install -y docker.io
```

Install curl

```
# apt-get install -y apt-transport-https curl
```

Install Kubernetes

```
# curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key
add -
```

```
# cat <<EOF >/etc/apt/sources.list.d/kubernetes.list
deb http://apt.kubernetes.io/ kubernetes-xenial main
EOF
```

Update the system

```
# apt-get update
```

Install Kubeadm, Kubelet, Kubectl

```
# apt-get install -y kubelet kubeadm kubectl
```

Run another update for all plugins to be updated, afterwards reboot the system

```
# apt-get update
```

### 3.3. Setup on the master

Set terminal to super user again

```
# sudo su
```

Get IP-address of master (if necessary install net-tools)

```
# ifconfig
```

Setup of Kubernetes Network (curly braces must be removed)

```
# kubeadm init --apiserver-advertise-address={ip-address-of-kmaster-vm} -
-pod-network-cidr=192.168.0.0/16
```

After installation a join command will printed, make a copy for joining workers later on

Example:

```
# kubeadm join masterIP:6443 --token m33ryp.6rn9refif \
    --discovery-token-ca-cert-hash sha256:d9b187312464d1375d699b0e444
```

Export config (root user is necessary)

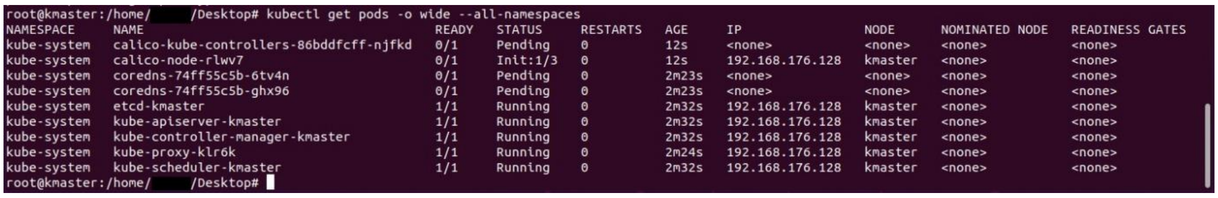
```
# export KUBECONFIG=/etc/kubernetes/admin.conf
```

Install Calico

```
# kubectl apply -f https://docs.projectcalico.org/manifests/calico.yaml
```

The following command can be used to check if the cluster is running properly the result should be similar to figure 6.

```
# kubectl get pods -o wide --all-namespaces
```



```
root@kmaster:/home/[redacted]/Desktop# kubectl get pods -o wide --all-namespaces
NAMESPACE     NAME                                       READY   STATUS     RESTARTS   AGE     IP                NODE      NOMINATED NODE   READINESS GATES
kube-system   calico-kube-controllers-86bddfcff-njfkd    0/1     Pending    0          12s     <none>            <none>    <none>           <none>
kube-system   calico-node-rlwv7                          0/1     Init:1/3   0          12s     192.168.176.128   kmaster   <none>           <none>
kube-system   coredns-74ff55c5b-6tv4n                    0/1     Pending    0          2m23s   <none>            <none>    <none>           <none>
kube-system   coredns-74ff55c5b-ghx96                    0/1     Pending    0          2m23s   <none>            <none>    <none>           <none>
kube-system   etcd-kmaster                               1/1     Running    0          2m32s   192.168.176.128   kmaster   <none>           <none>
kube-system   kube-apiserver-kmaster                     1/1     Running    0          2m32s   192.168.176.128   kmaster   <none>           <none>
kube-system   kube-controller-manager-kmaster            1/1     Running    0          2m32s   192.168.176.128   kmaster   <none>           <none>
kube-system   kube-proxy-klr6k                           1/1     Running    0          2m24s   192.168.176.128   kmaster   <none>           <none>
kube-system   kube-scheduler-kmaster                     1/1     Running    0          2m32s   192.168.176.128   kmaster   <none>           <none>
root@kmaster:/home/[redacted]/Desktop#
```

*Figure 6 list of all pods*

Install Kubernetes dashboard (must be done before nodes are added, or else it is not running on the master)

```
# kubectl apply -f
https://raw.githubusercontent.com/kubernetes/dashboard/v2.0.5/aio/deploy/
recommended.yaml
```

Create a service account

```
# kubectl -n default create serviceaccount dashboard
```

Create an admin-role

```
# kubectl -n default create clusterrolebinding dashboard-admin --
clusterrole=cluster-admin --serviceaccount=default:dashboard
```

Setup cluster role binding

```
# kubectl create clusterrolebinding cluster-system-anonymous --
clusterrole=cluster-admin --user=system:anonymous
```

Get a token, the output must be stored for later usage

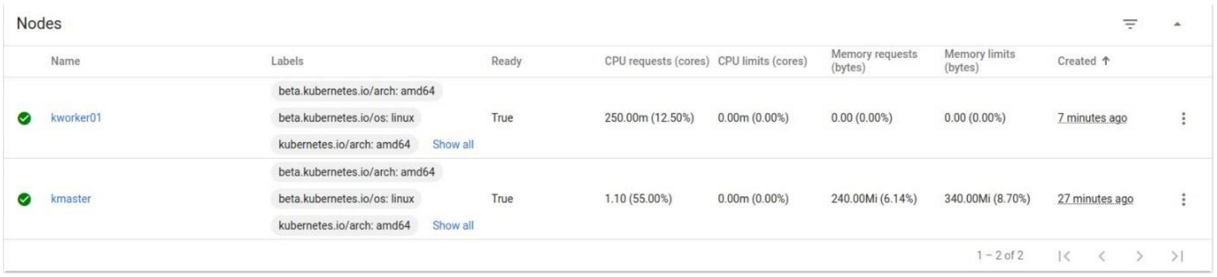
```
# kubectl get secret $(kubectl get serviceaccount dashboard -o
jsonpath="{.secrets[0].name}") -o jsonpath="{.data.token}"|base64 -d;echo
```

Start the proxy service

```
# kubectl proxy
```

The following link can be opened in the browser to access the cluster dashboard. It can only be reached on the master node and not remotely. An example of the dashboard can be seen in figure 7.

```
# http://localhost:8001/api/v1/namespaces/kubernetes-
dashboard/services/https:kubernetes-dashboard:/proxy/
```



Nodes

| | Name | Labels | Ready | CPU requests (cores) | CPU limits (cores) | Memory requests (bytes) | Memory limits (bytes) | Created ↑ |
|---|---|---|---|---|---|---|---|---|
| ✓ | kworker01 | beta.kubernetes.io/arch: amd64<br>beta.kubernetes.io/os: linux<br>kubernetes.io/arch: amd64 Show all | True | 250.00m (12.50%) | 0.00m (0.00%) | 0.00 (0.00%) | 0.00 (0.00%) | 7 minutes ago |
| ✓ | kmaster | beta.kubernetes.io/arch: amd64<br>beta.kubernetes.io/os: linux<br>kubernetes.io/arch: amd64 Show all | True | 1.10 (55.00%) | 0.00m (0.00%) | 240.00Mi (6.14%) | 340.00Mi (8.70%) | 27 minutes ago |

1 – 2 of 2

*Figure 7 screenshot Kubernetes dashboard*

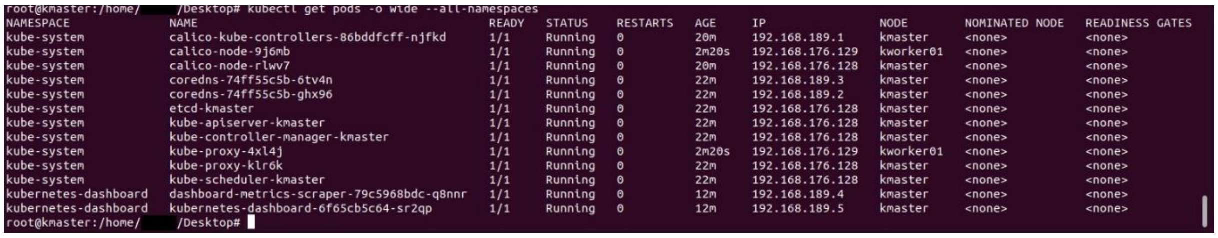
### 3.4. Setup on the workers

Use the join command that was printed during the setup of the Kubernetes cluster

```
# kubeadm join masterIP:6443 --token xyzExample \
    --discovery-token-ca-cert-hash sha256:xyzExampleHash
```

Afterwards the cluster should contain the worker node. It can be identified by the different hostname. The result should be similar to figure 8.

```
# kubectl get pods -o wide --all-namespaces
```



```
root@kmaster:/home/[redacted]/Desktop# kubectl get pods -o wide --all-namespaces
NAMESPACE              NAME                                          READY   STATUS    RESTARTS   AGE     IP                NODE        NOMINATED NODE   READINESS GATES
kube-system            calico-kube-controllers-86bddfcff-njfkd       1/1     Running   0          20m     192.168.189.1     kmaster     <none>           <none>
kube-system            calico-node-9j6mb                             1/1     Running   0          2m20s   192.168.176.129   kworker01   <none>           <none>
kube-system            calico-node-rlwv7                             1/1     Running   0          20m     192.168.176.128   kmaster     <none>           <none>
kube-system            coredns-74ff55c5b-6tv4n                       1/1     Running   0          22m     192.168.189.3     kmaster     <none>           <none>
kube-system            coredns-74ff55c5b-ghx96                       1/1     Running   0          22m     192.168.189.2     kmaster     <none>           <none>
kube-system            etcd-kmaster                                  1/1     Running   0          22m     192.168.176.128   kmaster     <none>           <none>
kube-system            kube-apiserver-kmaster                        1/1     Running   0          22m     192.168.176.128   kmaster     <none>           <none>
kube-system            kube-controller-manager-kmaster               1/1     Running   0          22m     192.168.176.128   kmaster     <none>           <none>
kube-system            kube-proxy-4xl4j                              1/1     Running   0          2m20s   192.168.176.129   kworker01   <none>           <none>
kube-system            kube-proxy-klr6k                              1/1     Running   0          22m     192.168.176.128   kmaster     <none>           <none>
kube-system            kube-scheduler-kmaster                        1/1     Running   0          22m     192.168.176.128   kmaster     <none>           <none>
kubernetes-dashboard   dashboard-metrics-scraper-79c5968bdc-q8nnr    1/1     Running   0          12m     192.168.189.4     kmaster     <none>           <none>
kubernetes-dashboard   kubernetes-dashboard-6f65cb5c64-sr2qp         1/1     Running   0          12m     192.168.189.5     kmaster     <none>           <none>
root@kmaster:/home/[redacted]/Desktop#
```

*Figure 8 list of all pods on two nodes*

# 4. Deployment of Nextcloud on a Kubernetes cluster

The repository, which is explained in chapter two, has to be copied to the master node of the cluster and needs to be set as working directory in the console.

Set the storage

```
# kubectl apply -f ./kubernetes/storage.yaml
```

Set the Nextcloud namespace

```
# kubectl create ns nextcloud
```

Load Secrets and set the secrets in ‘./kubernetes/secret.yaml’ accordingly. Keep in mind the values have to be base64 encoded in order to be loaded into the cluster

```
# kubectl apply -f ./kubernetes/secret.yaml
```

Install Kustomize

```
# https://kubectl.docs.kubernetes.io/installation/kustomize/binaries/
```

Install Krew (git is required)

```
(
  set -x; cd "$(mktemp -d)" &&
  curl -fsSLO "https://github.com/kubernetes-
sigs/krew/releases/latest/download/krew.tar.gz" &&
  tar zxvf krew.tar.gz &&
  KREW=./krew-"$(uname | tr '[:upper:]' '[:lower:]')_$(uname -m | sed -e
's/x86_64/amd64/' -e 's/arm.*$/arm/')" &&
  "$KREW" install krew
)
```

Set Krew namespace

```
# export PATH="${KREW_ROOT:-$HOME/.krew}/bin:$PATH"
```

Update Krew and install MinIO

```
# kubectl krew update
# kubectl krew install minio
# kubectl minio init
```

Build the yaml file for the cluster

```
# ./kustomize build -o ./dist/cluster-config.yaml ./kubernetes/
```

Apply the cluster

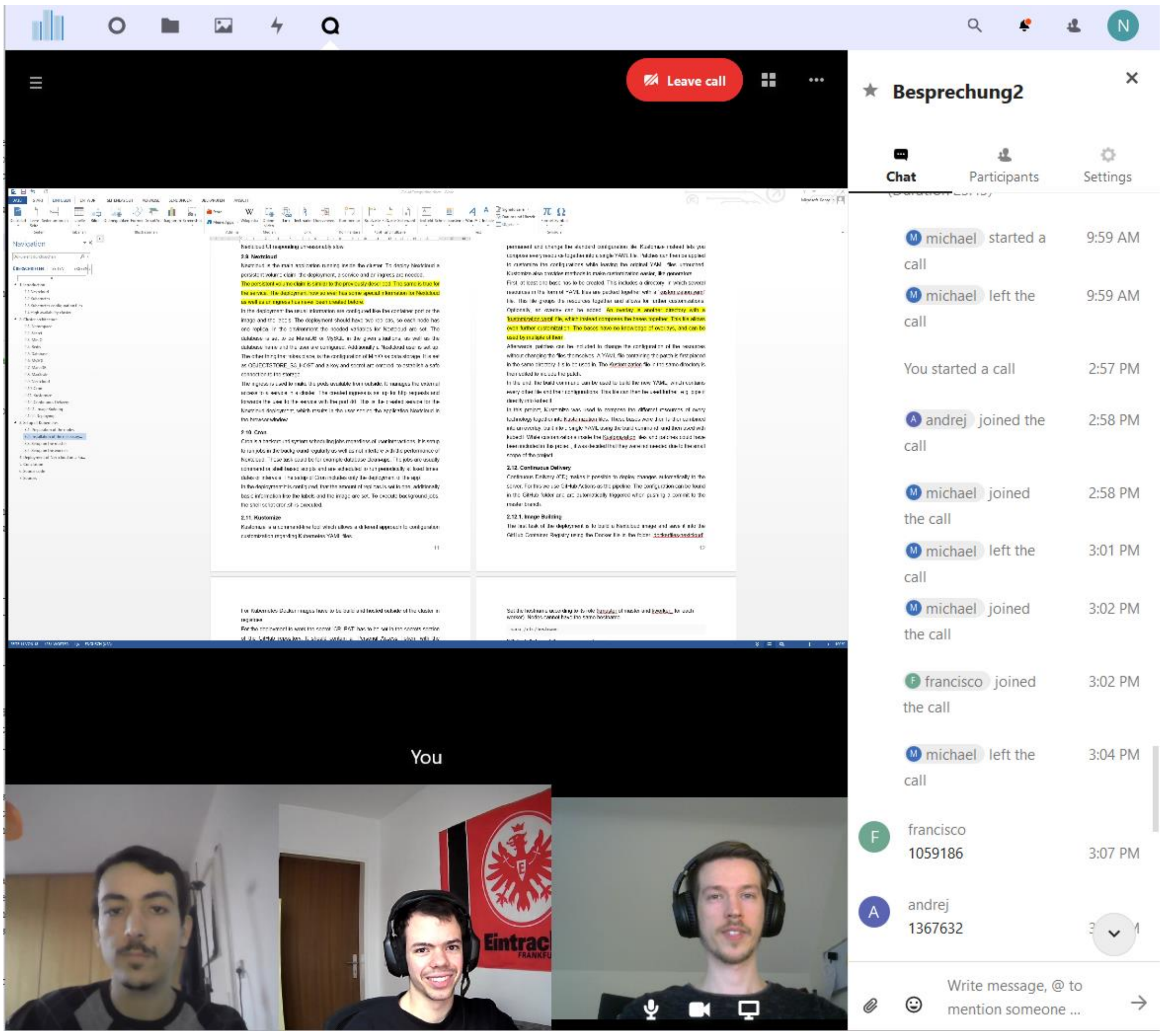
```
# kubectl apply -f ./dist/cluster-config.yaml
```

# 5. Conclusion

In this documentation the basic knowledge about Kubernetes, Nextcloud and Kustomize are given. The deployment of Kubernetes and Nextcloud as well as the used files are described in detail. With this documentation it is possible for the user to set up a multi-node system of Nextcloud and Kubernetes. Some examples of such UI elements can be seen in the figures 10 – 13.

Through the project the team studied the basic setup of a cluster with Kubernetes. Additionally the deployment of a web application had been learned. The deployed application was used during the project to actively work together as seen in figure 9.

In the future, the cluster could be expanded, to work on even more nodes and provide additional features like the simultaneous work on documents. Database replication could be used for increased failure resistance.



michael started a call 9:59 AM

michael left the call 9:59 AM

You started a call 2:57 PM

andrej joined the call 2:58 PM

michael joined the call 2:58 PM

michael left the call 3:01 PM

michael joined the call 3:02 PM

francisco joined the call 3:02 PM

michael left the call 3:04 PM

francisco 1059186 3:07 PM

andrej 1367632

*Figure 9 Nextcloud Voice used for working on the documentation*

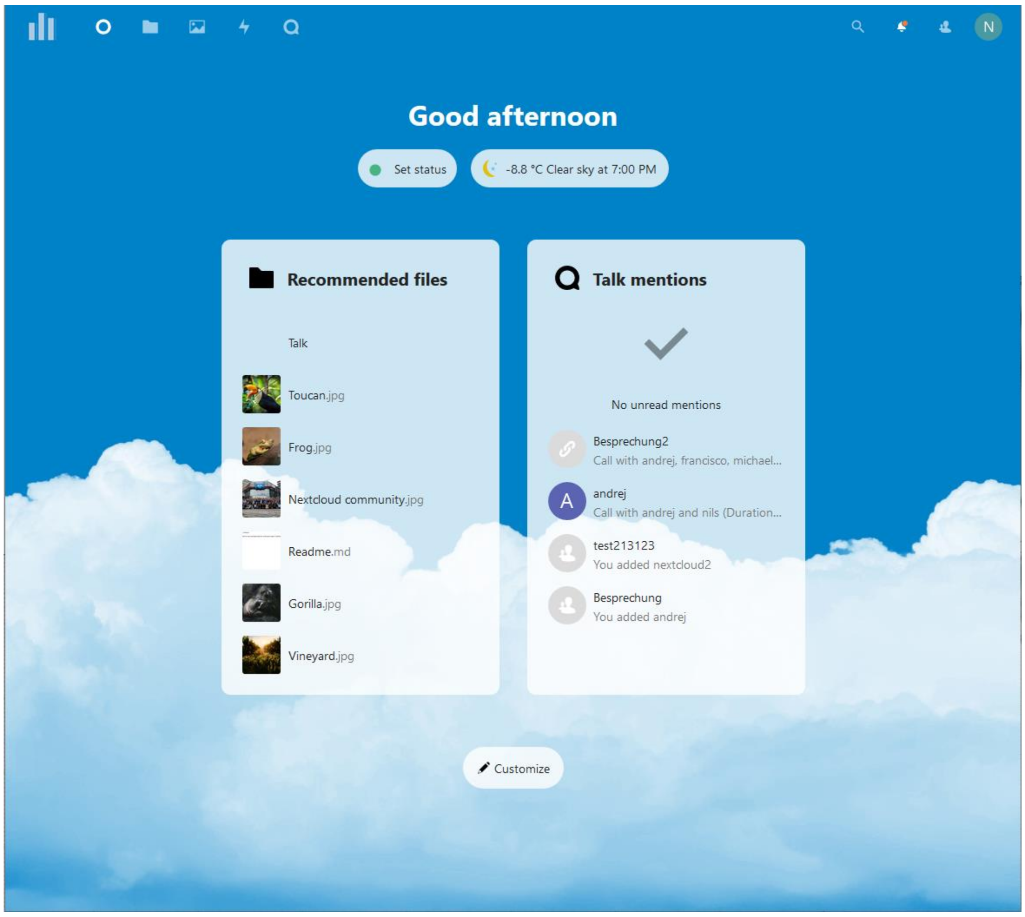


*Figure 10 Nextcloud dashboard view*

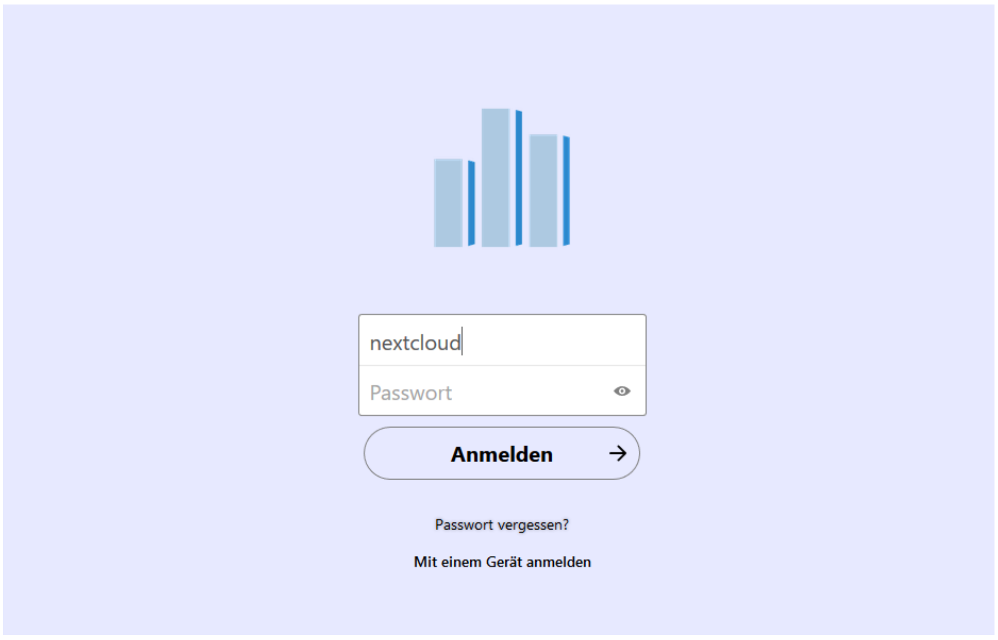
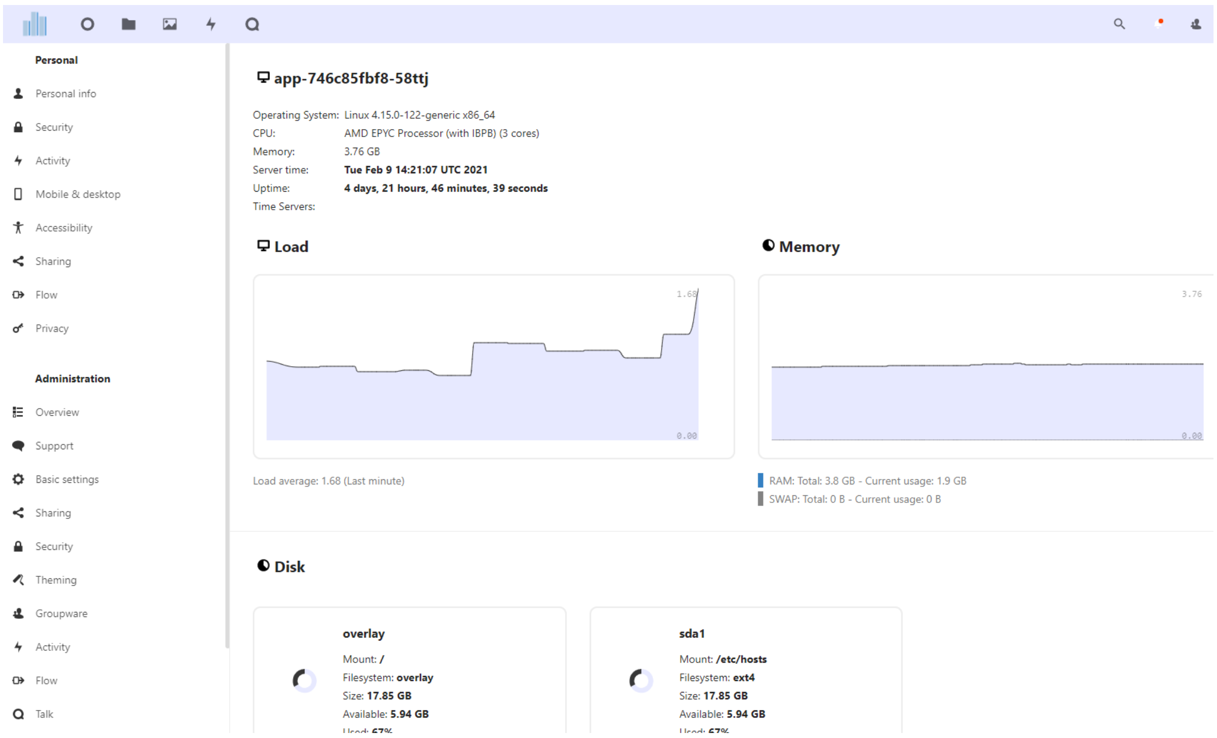


*Figure 11 login screen*



app-746c85fbf8-58ttj

Operating System: Linux 4.15.0-122-generic x86_64
CPU: AMD EPYC Processor (with IBPB) (3 cores)
Memory: 3.76 GB
Server time: **Tue Feb 9 14:21:07 UTC 2021**
Uptime: **4 days, 21 hours, 46 minutes, 39 seconds**
Time Servers:

Load

Memory

Load average: 1.68 (Last minute)

RAM: Total: 3.8 GB - Current usage: 1.9 GB
SWAP: Total: 0 B - Current usage: 0 B

Disk

**overlay**
Mount: **/**
Filesystem: **overlay**
Size: **17.85 GB**
Available: **5.94 GB**

**sda1**
Mount: **/etc/hosts**
Filesystem: **ext4**
Size: **17.85 GB**
Available: **5.94 GB**

*Figure 12 System overview*

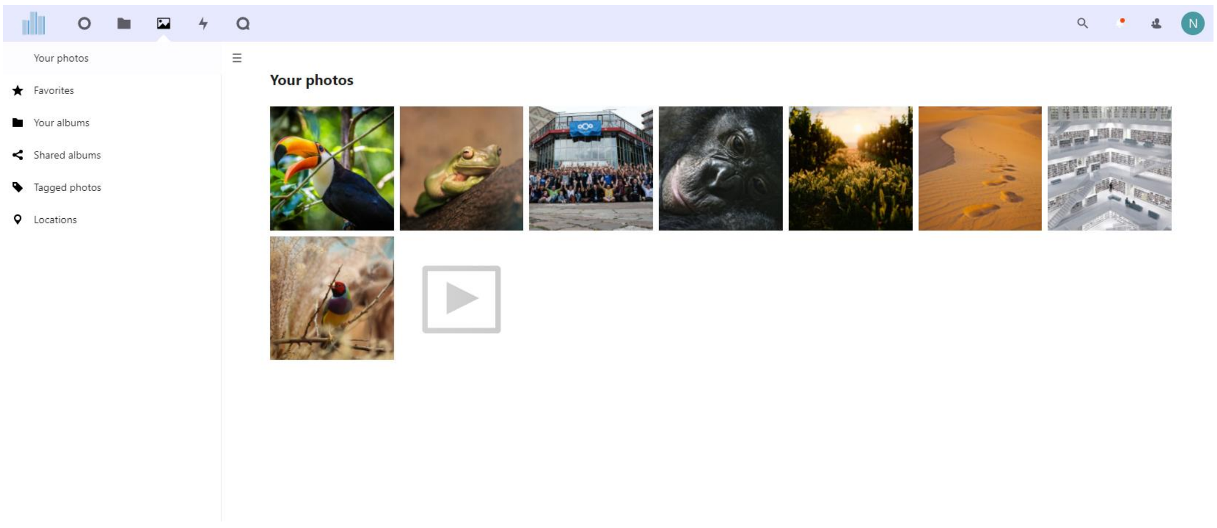


*Figure 13 file upload*

## 6. Source code

https://github.com/lolamtisch/BauCloCom_kubernetes

# 7. Sources

[1] W. A. Jansen, „Cloud Hooks: Security and Privacy Issues in Cloud Computing,“ *44th Hawaii International Conference on System Sciences,* p. 1, 2011.

[2] S. Scheuer und C. Kerkmann, „Handelsblatt,“ 26 08 2019. [Online]. Available: https://www.handelsblatt.com/technik/it-internet/cloud-dienste-eu-staaten-setzen-auf-deutsches-start-up-nextcloud/24942352.html?ticket=ST-3454851-vcfw3Di2rVfwSs6uCQyw-ap3. [Zugriff am 15 10 20].

[3] S. Landau, „Making Sense from Snowden: What’s Significant in the NSA Surveillance Revelations,“ *Spotlight,* pp. 1 - 3, 8 2013.

[4] google, „Google Trends,“ [Online]. Available: https://trends.google.com/trends/explore?q=ownCloud,Nextcloud,Seafile,Pydio. [Zugriff am 15 11 20].

[5] Nexctloud, „Nextcloud,“ [Online]. Available: https://nextcloud.com/blog/nextcloud-hub-20-debuts-dashboard-unifies-search-and-notifications-integrates-with-other-technologies/. [Zugriff am 15 11 20].

[6] C. Metz, „Wired,“ 6 10 2015. [Online]. Available: https://www.wired.com/2015/06/google-kubernetes-says-future-cloud-computing/.

[7] C. Metz, „Google Open Sources Its Secret Weapon in Cloud Computing,“ *Wired,* 10 6 2014.

[8] H.-J. Bader, „pro-linux,“ 07 22 2015. [Online]. Available: https://www.pro-linux.de/news/1/22554/cloud-native-computing-foundation-soll-container-technologien-zusammenbring.html. [Zugriff am 11 2020].

[9] E. Conrad und J. Feldman, „High Availability Cluster,“ in *Eleventh Hour CISSP® (Third Edition*, Elsevier, 2017.

[10] H. Sharma, „Edureka,“ [Online]. Available: https://www.edureka.co/blog/install-kubernetes-on-ubuntu.

[11] D. de Silva, „platformer,“ [Online]. Available: https://medium.com/platformer-blog/running-a-kubernetes-cluster-on-ubuntu-with-calico-9e372fb9175e.

[12] A. Cheaito, „medium.com / Ali Cheaito,“ [Online]. Available: https://medium.com/@acheaito/nextcloud-on-kubernetes-19658785b565.

[13] Khtan66, „wikimedia,“ [Online]. Available: https://commons.wikimedia.org/w/index.php?curid=53571935.