Cloud Computing WS20/21

Setup of Nextcloud using Kubernetes

Michael Zodel 1234567

Andrej Korinth 1234567

Francisco Seipel-Soubrier 1123213213123

Nils Krug 1367470

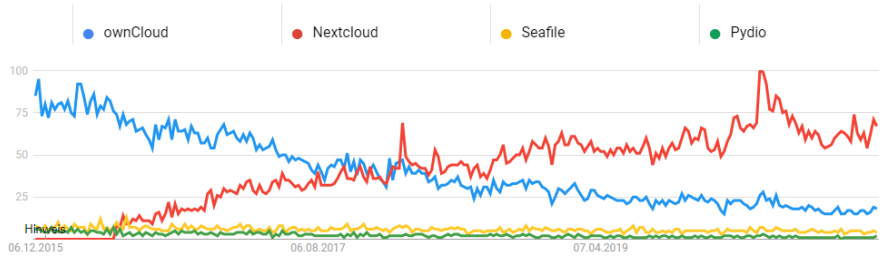
Table of contents

Introduction to 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 (Jansen, 2011). Big companies like Volkswagen and EON are using these services for their business processes (Handelsblatt, 2019).

The 2013 Edward Snowden leak showed, that American companies are tightly connected to government bodies and sharing user data for analysis (Landau, 2013). 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. While the enterprise part of Owncloud is closed source, all features of Nextcloud are open source. Since its launch the popularity of Nextcloud increased steadily making it one of the biggest alternatives to Amazon and co.



Picture 1 Google Trends for alternative cloud solutions

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. (Handelsblatt, 2019). Due to its open source nature, implementing backdoors for surveillance is unlikely. (Source?)

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.

Introduction to Kubernetes

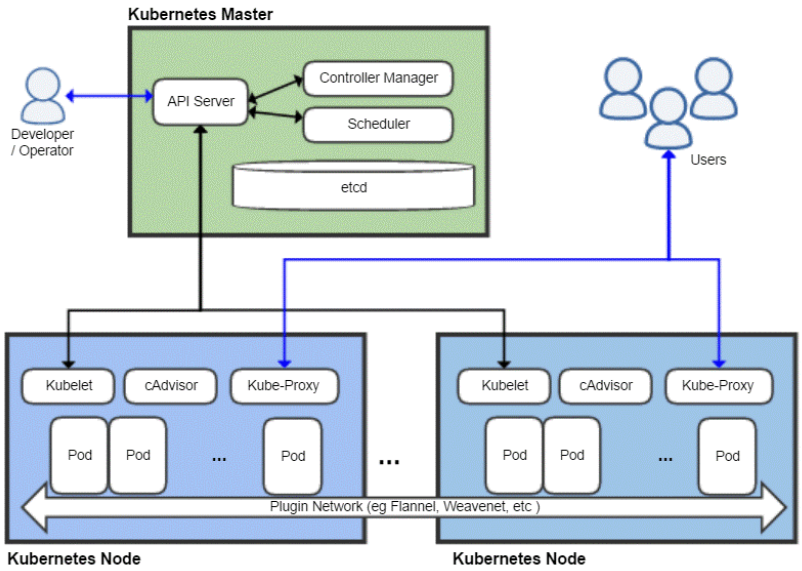
Kubernetes is a portable and scalable open-source-platform to manage container-based applications and services. It eases up the configuration as well as the automatization. Joe Beda, Brendan Burns and Craig McLuckie founded Kubernetes. Later on more Google employees collaborated. (Wired, 2015) 2014 Kubernetes has been announced. (Metz, 2014) 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.Till then it remained open source. (pro-linux, 2015)

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. 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 react 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 an 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 user's services are created. There are several services which can be created for example a network service for the user to access the application. The Services manage different port forwarding and support the pods to get the relevant data from other pods. Another example for a service is a load balancer. The load balancer analyses the volume of workload on different nodes and automatically distributes the workload evenly.



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 (also called “fail over”). 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. Thus the users are balanced between every node of the system. 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.

Yaml Files

Yaml files are used to deploy different objects inside Kubernetes. With these files it is easy to document the deployed services and pods and change 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 you want to create. 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 passwort 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:

Namespace.yaml

The namespace.yaml is the first file which is applied. It 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.

Secret.yaml

The secret.yaml is the next file which is applied. It contains all passwords and users for nextcloud as well as mysql. All values inside this file are base64 encoded to not be stolen easily.
This secrets should be changed for your server every time you build up a new cluster to not allow attackers to get access to your infrastructure easily.
The kind Secretis also slightly different than the general structure. A type of the secret is descibed as well as different variables inside the data option.

Min.IO

After setting up the namespace and the secret minio is set up to Store the data of nextcloud. For the deployment 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 is 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.

Redis

As minio is set up Redis has to be set up. Redis is a modern memcache to use for distributed caching and as key-value store for avoiding file corruption during normal operations. It includes a service as well as the pods.

The service is again very simple, only defining the port of the service. In the deployment the initialized service is then used. This pot is only created once, as it is used for all servers. Additionally in the deployment only the needed values are set to correctly setup a redis pod, like the image or the restart policy.

Database

The next step is to setup the database for nextcloud. The database is needed to store administrative data. As database you can setup MySQL, MariaDB, PostgreSQL and a Oracle database, recommended by Nextcloud are the first two options. We setup two different setups, one with MySQL and one with MariaDB. In the following the yaml files of both setups are described.

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. So first the master is created and afterwards the childs are created. 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 genaration 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.

MariaDB

The MariaDB setup has one database for all node, so all nodes connect to this database. 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 claim is 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 passwort 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. 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 simuntaniously.

Nextcloud

After all the previous setup Nextcloud can be created. For Nextcloud again a persistent volume claim, the deployment, a service and an ingress are needed.
The persistent volume claim is pretty much the same like the ones before. The same is true for the service. The deployment how so ever has some special information for Nextcloud as well as an ingress has never been created before.
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.

Cron

The last step in the setup is to configure Cron. Cron is a background system scheduling jobs regardless of user interactions. It is setup 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.

After all this thing have been created you should have a running cluster of Nextcloud running in kubernetes. For more detailed information about the setup, you should have a look into chapter $Link chapter of installation guide$.

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. This includes a directory, in which several resources in the form of YAML files are packed together with a “kustomization.yaml” file. This Kustomization file groups the resources together and allows for further customizations. Optionally, an overlay can be added. An overlay is another directory with a “kustomization.yaml” file, which instead composes the bases together. This file allows even further customization. The bases have no knowledge of overlays, and can in fact be used by multiple of them.
Afterwards, 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 merely 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.

Setup of Kubernetes

Introduction

This setup is intended for Ubuntu VMs running on VMware Workstation 16. 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.

Prepare the system

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

```
sudo su
```

Kommentiert [n1]: Fixed char size für font

Update the system

```
apt-get update
```

Disable swap file (necessary for starting a Kubernetes cluster)

```
swapoff –a
```

Comment out the entry for the swapfile 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
```

Install 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
```

Open 10-kubeadm.conf and add following line after the last environment variable

```
nano /etc/systemd/system/kubelet.service.d/10-kubeadm.conf
```

```
Environment=”cgroup-driver=systemd/cgroup-driver=cgroupfs”
```

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

```
apt-get update
```

Master setup

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

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

Install kubernetes dashboard (must be done before nodes are added, or else its 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

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

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.

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

Deployment of a multimode Nextcloud instance

Set the storage

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

Set the Nextcloud namespace

```
kubectl create ns nextcloud
```

Load Secrets. Set the secrets in `./kubernetes/secret.yaml` accordingly. Keep in mind the values have to be base64 encoded and load it into your 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
```

What have we learned