

AHMED MAHGOUB

Verlag Milestone 2



Inhoud

| | |
|------------------------------------|-----------|
| 1. INTRODUCTIE | 4 |
| 2. VAGRANT | 5 |
| 2.1. Vagrant init | 5 |
| 2.2. Vagrantfile configuratie | 6 |
| 3. KUBERNETES & KUBEADM | 8 |
| 3.1. Kubernetes installatie | 8 |
| 3.2. Worker nodes | 9 |
| 4. DEPLOYMENTS | 13 |
| 4.1. Apache | 13 |
| 4.1.1. De pods bereiken | 16 |
| 4.2. FastAPI | 17 |
| 4.3. PostgreSQL | 22 |
| 5. PROMETHEUS | 25 |
| 5.1. Values.yaml | 25 |
| 5.2. Automatisatie | 26 |
| 6. CONCLUSIE | 28 |

Lijst met figuren

| | |
|--|---|
| Afbeelding 1. Vagrant logo (HashiCorp, 2023) | 5 |
| Afbeelding 2. Kubernetes logo | 8 |
| Afbeelding 3. Kubernetes architecture | 8 |

1. Introductie

In het kader van het vak Linux Web Services ben ik aangegaan met een uitdagende opdracht die een enigszins diepgaand begrip vereiste van het opzetten en beheren van een volledig functionerende web stack in een Kubernetes-cluster. Mijn doel was niet alleen het creëren van een werkende webstack met Linux, Apache, FastAPI, en Postgresql, maar ook het verkennen van de uitgebreide mogelijkheden die Kubernetes biedt, zoals orkestratie, schaalbaarheid van applicaties en portabiliteit tussen verschillende cloudproviders.

Dit verslag geeft een redelijk gedetailleerd overzicht van de stappen die ik heb ondernomen om deze uitdaging aan te gaan. Het project is niet alleen gericht op het realiseren van een werkende webstack, maar ook op het verkennen van de vele mogelijkheden die Kubernetes biedt, zoals orkestratie, het schalen van applicaties en portabiliteit tussen providers.

De volgende secties van dit verslag beschrijven de stappen en resultaten van dit project, variërend van het opzetten van een Kubernetes cluster met behulp van Kubeadm, gebruik maken van verschillende Kubernetes resources en “resource-monitoring” met Prometheus.

Naast dit document heb ik ook een github repository aangemaakt waarin alle scripts en YAML bestanden te vinden zijn: <https://github.com/ahm282/milestone2>

2. Vagrant

Vagrant is een open-source tool voor het beheren van virtuele machine-omgevingen. Het biedt een eenvoudige manier om identieke ontwikkelomgevingen aan te maken en te configureren op verschillende machines, waardoor het makkelijk wordt om code te delen en samen te werken met anderen. Vagrant kan ook gebruikt worden om op een snelle manier “production-like” omgevingen te creëren om te testen en uit te rollen.¹

Vagrant gebruikt een configuratiebestand genaamd Vagrantfile om de gewenste toestand van de virtuele machine (VM) te definiëren. Het Vagrantfile kan het besturingssysteem, softwarepakketten en andere instellingen definiëren die op de virtuele machine geïnstalleerd moeten worden. Vagrant gebruikt een virtualisatie provider zoals VirtualBox of VMware om een virtuele machine te creëren en te beheren.²



Afbeelding 1. Vagrant logo (HashiCorp, 2023)

2.1. Vagrant init

Vagrant init is een tool binnen de Vagrant CLI interface die gebruikt kan worden om virtuele machine-omgevingen aan te maken en te beheren op een consistente en reproduceerbare manier.³ Vagrant init initialiseert de huidige directory als een Vagrant omgeving door een initieel Vagrantfile aan te maken als er nog geen bestaat.

Voor dit project heb ik de “generic/ubuntu2204” Vagrant box gebruikt. Hiermee beschikte ik over een generieke gebruiksklare Ubuntu 22.04 ontwikkelomgeving/VM.

A screenshot of a PowerShell terminal window. The window title is 'PowerShell 7.3.8'. The terminal output shows a notification for a new PowerShell stable release (v7.3.9) and the successful execution of the 'vagrant init "generic/ubuntu2204"' command. The output of the command states that a 'Vagrantfile' has been placed in the directory and provides instructions on how to use Vagrant.

```
PowerShell 7.3.8
A new PowerShell stable release is available: v7.3.9
Upgrade now, or check out the release page at:
https://aka.ms/PowerShell-Release?tag=v7.3.9

PS G:\project> vagrant init "generic/ubuntu2204"
A 'Vagrantfile' has been placed in this directory. You are now
ready to 'vagrant up' your first virtual environment! Please read
the comments in the Vagrantfile as well as documentation on
'vagrantup.com' for more information on using Vagrant.
PS G:\project>
```

¹ HashiCorp, 2023

² HashiCorp, 2023

³ HashiCorp, 2023

2.2. Vagrantfile configuratie

Het configureren van Vagrantfiles is gelukkig niet ingewikkeld. Vagrantfiles zijn geschreven in Ruby⁴, maar er is geen voorkennis van de programmeertaal nodig om wijzigingen aan te brengen. Om een vlotte gebruikservaring te realiseren, heb ik verschillende aanpassingen gedaan aan de virtuele machine, zoals het verhogen van het beschikbare RAM-geheugen. Ik heb ook een statisch IP-adres geconfigureerd en een gesynchroniseerde map ingesteld tussen mijn Windows-besturingssysteem en de virtuele machine. Zodra de virtuele machine is klaargezet, gebruik ik een eigen provisioning-script om een Kubeadm-cluster te installeren en in te stellen.

```
vb.memory = "4096"
config.vm.network "private_network", ip: "192.168.33.10"
config.vm.synced_folder "./docker/", "/vagrant"
```

```
11 # Create a private network, which allows host-only access to the machine
12 # Note: this requires the 'network' plugin.
13 # Note: this sets up the network, but leaves the machine's network
14 # configuration empty. This means you will likely see the device named
15 # 'enx000000010000' in the guest.
16 config.vm.network "private_network", ip: "192.168.33.10"
17 # Create a public network, which generally mimics a bridge to
18 # your network. Bridged networks make the machine appear as another
19 # physical device on your network.
20 # Note: this requires the 'network' plugin.
21 # Note: this sets up the network, but leaves the machine's network
22 # configuration empty. This means you will likely see the device named
23 # 'enx000000010000' in the guest.
24 # Note: this is not the recommended configuration for production
25 # environments.
26 config.vm.network "public_network", bridge: "VagrantBox Wi-Fi & BT/221 Wireless LAN Card"
27 # Share an additional folder to the guest VM. The first argument is
28 # the path on the host to the actual folder. The second argument is
29 # the path on the guest to mount the folder. And the optional third
30 # argument is a set of non-required options.
31 config.vm.synced_folder "./shared", "/vagrant"
32 # Disable the default share of the current code directory, being this
33 # provides improved isolation between the Vagrant box and your host
34 # by making sure your sensitive code is accessible to the Vagrant box.
35 # If you use this you may want to enable additional shared subfolders as
36 # shown above.
37 # config.vm.synced_folder ".", "vagrant", disabled: true
38 # Provider-specific configuration so you can fine-tune various
39 # backing providers for Vagrant. These expose provider-specific options.
40 # Example for VirtualBox:
41 #
42 # config.vm.provider "virtualbox" do |vb|
43 #   # Display the VirtualBox GUI when booting the machine
44 #   vb.gui = true
45 #   # Adjust the number of CPUs to use
46 #   vb.cpus = 2
47 #   # Adjust the number based on your needs
48 #   vb.name = "k8s-master"
49 #   # Customize the amount of memory on the VM:
50 #   vb.memory = "4096"
51 # end
52 # See the documentation for the providers you are using for more
53 # information on available options.
54 # Enable provisioning with a shell script. Additional provisioners such as
55 # Ansible, Chef, Docker, Puppet and Salt are also available. Please see the
56 # documentation for more information about their specific syntax and use.
57 config.vm.provision "shell", path: "./script.sh"
58 end
```

Na het instellen van de VM, voerde ik de commando's "vagrant up" en "vagrant ssh" uit om toegang te krijgen tot een shell binnen de virtuele machine.

⁴ HashiCorp, 2023

```
PowerShell
default: Then you can join any number of worker nodes by running the following on each as root:
default: kubectl kubeadm join 192.168.33.18:6443 --token fatkxc.d1dzn16necrncd \
default: --discovery-token-ca-cert-hash sha256:b2981c6bdca320a4f780193bb4fb5bd151b7f55a1b18d1f6dd6ef88f83
default: poddisruptionbudget.policy/calico-kube-controllers created
default: serviceaccount/calico-kube-controllers created
default: serviceaccount/calico-node created
default: serviceaccount/calico-cni-plugin created
default: configmap/calico-confd created
default: customresourcedefinition.apiextensions.k8s.io/bgpconfigurations.crd.projectcalico.org created
default: customresourcedefinition.apiextensions.k8s.io/oppfilters.crd.projectcalico.org created
default: customresourcedefinition.apiextensions.k8s.io/oppsets.crd.projectcalico.org created
default: customresourcedefinition.apiextensions.k8s.io/blockaffinities.crd.projectcalico.org created
default: customresourcedefinition.apiextensions.k8s.io/calicomodelattests.crd.projectcalico.org created
default: customresourcedefinition.apiextensions.k8s.io/clusterinformations.crd.projectcalico.org created
default: customresourcedefinition.apiextensions.k8s.io/felixconfigurations.crd.projectcalico.org created
default: customresourcedefinition.apiextensions.k8s.io/globalnetworkpolicies.crd.projectcalico.org created
default: customresourcedefinition.apiextensions.k8s.io/globalnetworksets.crd.projectcalico.org created
default: customresourcedefinition.apiextensions.k8s.io/hostendpoints.crd.projectcalico.org created
default: customresourcedefinition.apiextensions.k8s.io/ippools.crd.projectcalico.org created
default: customresourcedefinition.apiextensions.k8s.io/ippamconfigs.crd.projectcalico.org created
default: customresourcedefinition.apiextensions.k8s.io/ippamhandles.crd.projectcalico.org created
default: customresourcedefinition.apiextensions.k8s.io/ippools.crd.projectcalico.org created
default: customresourcedefinition.apiextensions.k8s.io/ippreservations.crd.projectcalico.org created
default: customresourcedefinition.apiextensions.k8s.io/subcontrollersconfigurations.crd.projectcalico.org created
default: customresourcedefinition.apiextensions.k8s.io/networkpolicies.crd.projectcalico.org created
default: customresourcedefinition.apiextensions.k8s.io/networksets.crd.projectcalico.org created
default: clusterrole.rbac.authorization.k8s.io/calico-kube-controllers created
default: clusterrole.rbac.authorization.k8s.io/calico-node created
default: clusterrule.rbac.authorization.k8s.io/calico-cni-plugin created
default: clusterrule.rbac.authorization.k8s.io/calico-kube-controllers created
default: clusterrulebinding.rbac.authorization.k8s.io/calico-node created
default: clusterrulebinding.rbac.authorization.k8s.io/calico-cni-plugin created
default: daemonset.apps/calico-node created
default: deployment.apps/calico-kube-controllers created
default: namespace/nginx-ingress created
default: serviceaccount/nginx-ingress created
default: clusterrole.rbac.authorization.k8s.io/nginx-ingress created
default: clusterrulebinding.rbac.authorization.k8s.io/nginx-ingress created
default: secret/default-server-secret created
default: configmap/nginx-confd created
default: ingressclass.networking.k8s.io/nginx created
default: customresourcedefinition.apiextensions.k8s.io/virtualservers.k8s.nginx.org created
default: customresourcedefinition.apiextensions.k8s.io/virtualserverroutes.k8s.nginx.org created
default: customresourcedefinition.apiextensions.k8s.io/trafficservers.k8s.nginx.org created
default: deployment.apps/nginx-ingress created
default: deployment.apps/nginx-ingress created
default: daemonset.apps/nginx-ingress created
default: Downloading https://get.helm.sh/helm-v3.13.1-linux-amd64.tar.gz
default: Verifying checksum... done
default: Preparing to install helm into /usr/local/bin
default: helm installed into /usr/local/bin/helm
default: Kubernetes master node setup completed.
PS G:\kubeadm>
```

```
vagrant@ubuntu2204: ~$ neofetch
      ,--/+oossssoo+/--.
      `:+ssssssssssssssss+`
    -+ssssssssssssssssyyssst-
      .ossssssssssssssssdMMMMysssso.
    /ssssssssshdmmNNmmyNMMHhssssss/
    +ssssssshmydMMMMMMNdddyssssssst+
    /ssssssshNMMHhhyyyhNMMHhssssssst/
    .sssssssdMMHhssssssshNMMHdssssssst.
    +ssshhyyNMMHhsssssssssyNMMHhssssssst+
    ossyNMMHhNMMhssssssssshmmhssssssst+
    ossyNMMHhNMMhssssssssshmmhssssssst+
    +ssshhyyNMMHhsssssssssyNMMHhssssssst+
    .sssssssdMMHhssssssshNMMHdssssssst.
    /ssssssshNMMHhhyyyhNMMHhssssssst/
    +sssssssdmydMMMMMMNdddyssssssst+
    /ssssssshdmmNNmmyNMMHhssssssst/
    .ossssssssssssssssdMMMMysssso.
    -+ssssssssssssssssyyssst-
      `:+ssssssssssssssss+`
      ,--/+oossssoo+/--.

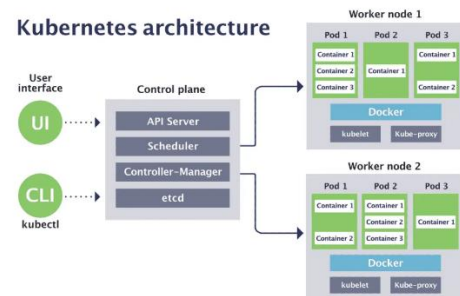
vagrant@ubuntu2204:~$ |
vagrant@ubuntu2204.localdomain
-----
OS: Ubuntu 22.04.2 LTS x86_64
Host: VirtualBox 1.2
Kernel: 5.15.0-69-generic
Uptime: 5 mins
Packages: 762 (dpkg), 4 (snap)
Shell: bash 5.1.16
Resolution: 1024x768
Terminal: /dev/pts/0
CPU: AMD Ryzen 5 5600G with Radeon Graphics (2) @ 3.899GHz
GPU: VirtualBox Graphics Adapter
Memory: 197MiB / 3924MiB
```

3. Kubernetes & Kubectl

Kubernetes, vaak afgekort als K8s, is een krachtig containerorkestratieplatform dat ontworpen is om het beheer en de schaalbaarheid van “containerized” applicaties te vereenvoudigen. In plaats van zich te concentreren op individuele containers, richt Kubernetes zich op het coördineren en orkestreren van deze containers in een gedistribueerd systeem.⁵



Afbeelding 2. Kubernetes logo



Afbeelding 3. Kubernetes architecture

3.1. Kubernetes installatie

Om Kubectl te installeren, gebruik ik een zelfgeschreven bash script. Het begint met het uitschakelen van swapgeheugen, omdat Kubectl niet goed werkt met swapgeheugen. Het script volgt dit patroon:

1. packages bijwerken
2. hostname wijzigen
3. kernelmodules activeren
4. hosts bijwerken
5. systemctl parameters configureren
6. dependencies & containerd installeren
7. kubect, Kubectl & Kubectl installeren
8. cluster opzetten
9. networkplugin installeren
10. nginx Ingress Controller installeren
11. join token aanmaken en opslaan in bestand
12. helm installeren
13. swapgeheugen cronjob toevoegen

⁵ Docker, 2023


```
#!/bin/bash
# Master node setup script
set -e

# Network configuration
echo "192.168.17.11 kube-node1" | sudo tee -a /etc/hosts

# Container engine
sudo systemctl --system
sudo apt --systemd-restart=no install docker.io

# Containerd
sudo apt-get update
sudo apt-get install -y containerd
sudo mkdir -p /etc/containerd
sudo sh -c 'containerd config default > /etc/containerd/config.toml'
sudo sed -i 's/SystemdGroup = false/SystemdGroup = true/' /etc/containerd/config.toml
sudo systemctl restart containerd

# Kubernetes repositories
sudo apt-get update
sudo apt-get install -y apt-transport-https ca-certificates curl gnupg

# Add Kubernetes GPG key
curl -fsSL https://pkgs.k8s.io/core/stable/v1.28/deb/Release.key | gpg --dearmor -o /etc/apt/trusted.gpg.d/kubernetes.gpg
echo 'deb [signed-by=/etc/apt/trusted.gpg.d/kubernetes.gpg] https://pkgs.k8s.io/core/stable/v1.28/deb/ /' | sudo tee /etc/apt/sources.list.d/kubernetes.list

# Install Kubernetes components
sudo apt-get update
sudo apt-get install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl
sudo kubeadm config images pull

# Node registration
kubeadm init --pod-network-cidr=172.17.0.0/16 --apiserver-advertise-address=192.168.17.11

# Configure kubelet
export KUBECONFIG=/etc/kubernetes/admin.conf

# Apply Calico network plugin
kubectl apply -f https://docs.projectcalico.org/main/install/calico.yaml

# Script complete
echo "Kubernetes master node setup completed."
```

3.2. Worker nodes

Om de worker nodes te configureren, pas ik een licht aangepast script toe dat wordt uitgevoerd nadat de master node/control plane klaar is met instellen.

Ik controleer of alles correct is uitgevoerd met het commando "kubectl get nodes".

```
#!/bin/bash
# Worker node setup script
set -e

# Network configuration
echo "192.168.17.12 kube-node2" | sudo tee -a /etc/hosts

# Container engine
sudo systemctl --system
sudo apt --systemd-restart=no install docker.io

# Containerd
sudo apt-get update
sudo apt-get install -y containerd
sudo mkdir -p /etc/containerd
sudo sh -c 'containerd config default > /etc/containerd/config.toml'
sudo sed -i 's/SystemdGroup = false/SystemdGroup = true/' /etc/containerd/config.toml
sudo systemctl restart containerd

# Kubernetes repositories
sudo apt-get update
sudo apt-get install -y apt-transport-https ca-certificates curl gnupg

# Add Kubernetes GPG key
curl -fsSL https://pkgs.k8s.io/core/stable/v1.28/deb/Release.key | gpg --dearmor -o /etc/apt/trusted.gpg.d/kubernetes.gpg
echo 'deb [signed-by=/etc/apt/trusted.gpg.d/kubernetes.gpg] https://pkgs.k8s.io/core/stable/v1.28/deb/ /' | sudo tee /etc/apt/sources.list.d/kubernetes.list

# Install Kubernetes components
sudo apt-get update
sudo apt-get install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl
sudo kubeadm config images pull

# Node registration
kubeadm join --token=token --discovery-token-addr=https://192.168.17.11:443 --node-name=kube-node2

# Script complete
echo "Kubernetes worker node setup completed."
```

```
PS C:\Users\ahmed> kubectl get nodes
NAME                 STATUS   ROLES    AGE   VERSION
k8s-master           Ready   control-plane   5m   v1.28.4
k8s-node1            Ready   node        5m   v1.28.4
k8s-node2            Ready   node        5m   v1.28.4
No resources found in default namespace.
```

Init-script.sh:

```
#!/bin/bash

set -e

# Disable swap
sudo swapoff -a
sudo sed -i 's/^/#/' /etc/fstab

# Update and set hostname
sudo apt-get update
sudo apt-get upgrade -y
sudo hostnamectl set-hostname "k8s-master"

# Load kernel modules
cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
overlay
br_netfilter
EOF

sudo modprobe overlay
sudo modprobe br_netfilter

# Update hosts file
echo "192.168.33.11 k8s-node01" | sudo tee -a /etc/hosts

# Configure sysctl parameters
cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-iptables = 1
net.bridge.bridge-nf-call-ip6tables = 1
net.ipv4.ip_forward = 1
EOF

# Apply sysctl parameters without reboot
sudo sysctl --system

# Install dependencies
sudo apt -y install docker.io

# Install containerd (the container runtime)
sudo apt-get update && sudo apt-get install -y containerd
```

```
sudo mkdir -p /etc/containerd
sudo sh -c "containerd config default > /etc/containerd/config.toml"
sudo sed -i 's/ SystemdCgroup = false/ SystemdCgroup = true/' /etc/containerd/config.toml
sudo systemctl restart containerd

# Set up Kubernetes repositories
sudo apt-get update
sudo apt-get install -y apt-transport-https ca-certificates curl gnupg

# Import Kubernetes GPG key
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.28/deb/Release.key | gpg --dearmor -o
/etc/apt/trusted.gpg.d/kubernetes.gpg

# Add Kubernetes repository
echo 'deb [signed-by=/etc/apt/trusted.gpg.d/kubernetes.gpg]
https://pkgs.k8s.io/core:/stable:/v1.28/deb/ /' | sudo tee
/etc/apt/sources.list.d/kubernetes.list

# Install Kubernetes components
sudo apt-get update
sudo apt-get install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl
sudo kubeadm config images pull

# Initialize Kubernetes master
kubeadm init --pod-network-cidr=172.16.0.0/12 --apiserver-advertise-address=192.168.33.10

# Configure kubectl
export KUBECONFIG=/etc/kubernetes/admin.conf

# Apply Calico network plugin
kubectl apply -f https://docs.projectcalico.org/manifests/calico.yaml

# Install Nginx Ingress Controller
cd /vagrant/kubernetes-ingress/deployments

# Create namespace and service account for the Ingress controller
kubectl apply -f common/ns-and-sa.yaml

# Create a cluster role and cluster role binding for the service account
kubectl apply -f rbac/rbac.yaml
```

```
# Create a default secret with a TLS certificate and a key for the default server in NGINX
kubectl apply -f ../examples/shared-examples/default-server-secret/default-server-secret.yaml
kubectl apply -f common/nginx-config.yaml
kubectl apply -f common/ingress-class.yaml

# Core custom resource definitions
kubectl apply -f common/crds/k8s.nginx.org_virtualservers.yaml
kubectl apply -f common/crds/k8s.nginx.org_virtualserverroutes.yaml
kubectl apply -f common/crds/k8s.nginx.org_transportservers.yaml
kubectl apply -f common/crds/k8s.nginx.org_policies.yaml

# Deploy the Ingress controller
kubectl apply -f deployment/nginx-ingress.yaml

# Create a daemon set with the Ingress controller
kubectl apply -f daemon-set/nginx-ingress.yaml

# Save the join token to a file
sudo kubeadm token create --print-join-command > /vagrant/join-token.sh
sudo chmod +x /vagrant/join-token.sh

# Copy kubeconfig to vagrant user
mkdir -p /home/vagrant/.kube
sudo cp -i /etc/kubernetes/admin.conf /home/vagrant/.kube/config
sudo chown 1000:1000 /home/vagrant/.kube/config

# Install Helm
cd /vagrant
sudo bash ./get_helm.sh

# Setup complete
echo "Kubernetes master node setup completed."

# Add a cronjob to disable swap on boot
echo "@reboot sudo swapoff -a" >> /etc/crontab

# Add a cronjob to restart kubelet on boot
echo "@reboot sudo systemctl restart kubelet" >> /etc/crontab
```

4. Deployments

4.1. Apache

Om een functionele Apache-serverimplementatie te realiseren, had ik de optie om een image van Docker Hub te kiezen. Ik koos er echter voor om het Dockerfile van mijlpaal 1 aan te passen om extra debug mogelijkheden te hebben in het geval van problemen met de database of API.

```
# Use Ubuntu 22.04 as base image
FROM ubuntu:latest

# Environment TZ variable
ENV TZ=Europe/Brussels
ENV DEBIAN_FRONTEND=noninteractive

# Install apache2 and PHP
RUN apt-get update && \
    apt-get -y upgrade && \
    apt-get -y install apache2 && \
    apt-get -y install software-properties-common && \
    add-apt-repository ppa:ondrej/php -y && \
    apt-get update && \
    apt-get -y install php libapache2-mod-php php-mysql php-pgsql

# Expose port 80
EXPOSE 80

# Start Apache in the foreground
CMD ["apache2ctl", "-D", "FOREGROUND"]
```

De dockerfile wordt dan gebouwd met `docker build` en naar een docker repository gepushed. Kubernetes deployments werken niet rechtstreeks met dockerfiles. Toen dat eenmaal in orde was, ging ik aan de slag met de service- en ingress-bestanden.

Een service biedt een stabiel endpoint en een DNS-naam waarmee andere delen van de applicatie of externe clients verbinding kunnen maken met de service, ongeacht de dynamische veranderingen in het aantal en de locaties van de individuele pods.

In de context van Kubernetes verwijst een "Ingress" naar een API-object dat de externe toegang tot services in een cluster beheert, meestal HTTP. Een Ingress kan worden gezien als een verzameling regels die bepalen hoe externe HTTP- en HTTPS-verzoeken moeten worden gerouteerd naar services binnen het Kubernetes-cluster.

Apache-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: apache-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: apache
  template:
    metadata:
      labels:
        app: apache
    name: apache-pod
    spec:
      containers:
        - name: apache-container
          image: ahm282/milestone2:apache
          imagePullPolicy: Always
          ports:
            - containerPort: 80
          volumeMounts:
            - name: html-volume
              mountPath: /var/www/html
      volumes:
        - name: html-volume
          hostPath:
            path: /vagrant/deployments/apache/www
            type: Directory
```

Apache-service.yaml :

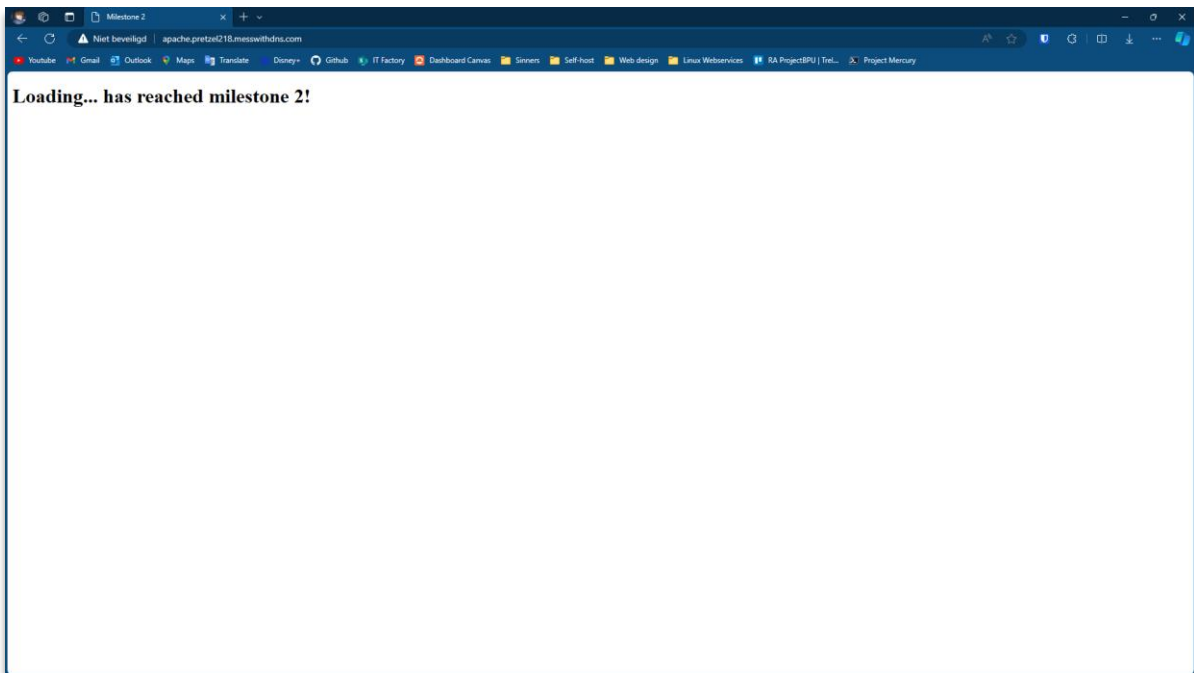
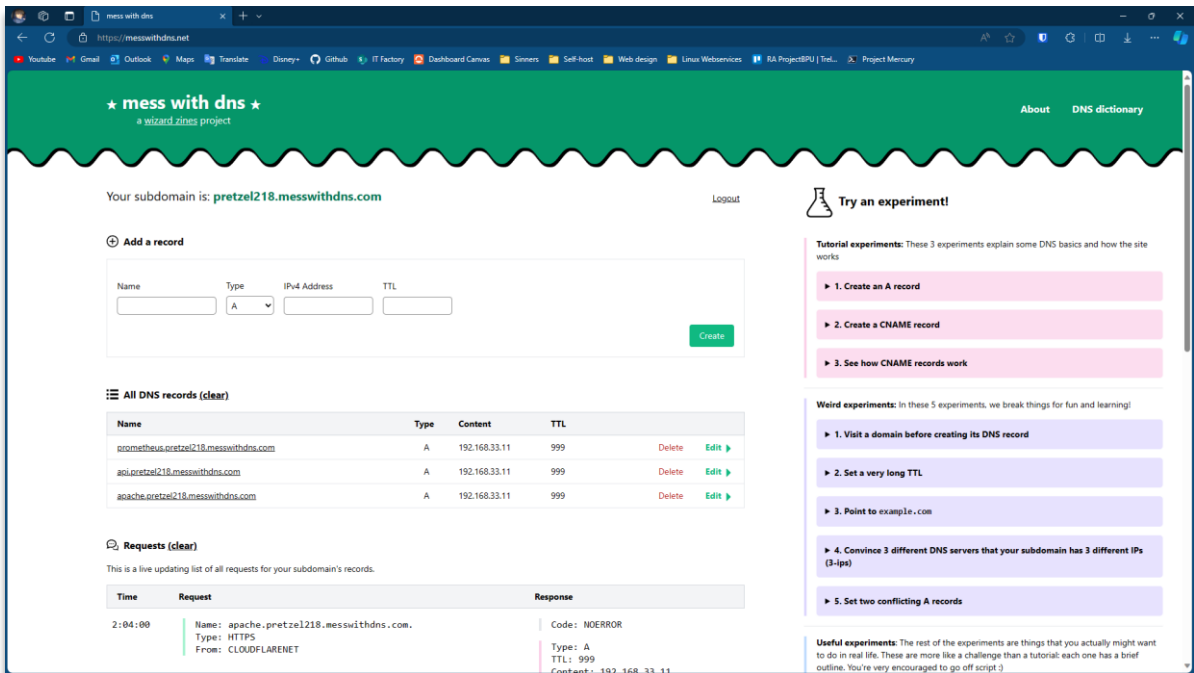
```
apiVersion: v1
kind: Service
metadata:
  name: apache-service
spec:
  selector:
    app: apache
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
  type: ClusterIP
```

Apache-ingress.yaml:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: apache-ingress
  annotations:
    nginx.ingress.kubernetes.io/ingress-class: "nginx"
spec:
  rules:
    - host: apache.pretzel218.messwithdns.com
      http:
        paths:
          - pathType: Prefix
            path: "/"
            backend:
              service:
                name: apache-service
                port:
                  number: 80
```

4.1.1. De pods bereiken

Om de pods te bereiken, had ik een domeinnaam nodig van messwithdns.net om te gebruiken in de ingress-bestanden.



4.2. FastAPI

Het instellen van de API was vergelijkbaar met het instellen van de Apache webserver. Ik gebruikte een aangepast dockerfile om FastAPI en de benodigde libraries te installeren. Het script main.py wordt ook gekopieerd naar de container, waarvoor ook een extra wijziging moet worden aangebracht in het deployment yaml-bestand namelijk, een "volumeMount" van het script.

```
# Use the official Python image
FROM python:latest

# Create a directory for the code
RUN mkdir /code

# Set the working directory
WORKDIR /code

# Copy the requirements file and install dependencies
COPY ./requirements.txt /code/requirements.txt
RUN pip install --no-cache-dir --upgrade -r /code/requirements.txt

# Copy the main script into the container
COPY ./main.py /code/

# Specify the command to run when the container starts
CMD ["uvicorn", "main:app", "--reload", "--host", "0.0.0.0", "--port", "8000"]
```

Main.py:

```
from fastapi import FastAPI, Depends, HTTPException
from sqlalchemy import create_engine, Column, Integer, String, MetaData, Table
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker, Session
from fastapi.middleware.cors import CORSMiddleware
import socket

# Replace the following variables with your own database connection details
DATABASE_URL = "postgresql://kube:kube@postgres-service/milestone"

# Create a FastAPI app
app = FastAPI()
```

```
# Add CORS middleware
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_methods=["*"],
    allow_headers=["*"],
)

# Create a SQLAlchemy engine and database session
engine = create_engine(DATABASE_URL)
metadata = MetaData()

# Define a table
fullname = Table(
    "fullname",
    metadata,
    Column("id", Integer, primary_key=True, index=True),
    Column("name", String),
)

# Create the table if it doesn't exist
metadata.create_all(bind=engine)

# Create a SQLAlchemy model for the "users" table
Base = declarative_base()
class Fullname(Base):
    __tablename__ = "fullname"
    id = Column(Integer, primary_key=True, index=True)
    name = Column(String)

# Define the SessionLocal class
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

# Dependency to get the database session
def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()
```

```

# Endpoint to get name
@app.get("/")
async def root():
    return {"name": "Ahmed Mahgoub"}

# Endpoint to get hostname
@app.get("/id")
async def read_name(db: Session = Depends(get_db)):
    hostname = socket.gethostname()
    return {"hostname": hostname}

# Endpoint to get name
@app.get("/name")
async def read_name(db: Session = Depends(get_db)):
    hostname = socket.gethostname()
    db_name = db.query(Fullname).first()
    if db_name is None:
        raise HTTPException(status_code=404, detail="User not found")
    return {"name": db_name.name, "hostname": hostname}

```

Api-deployment:

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: api-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: api
  template:
    metadata:
      labels:
        app: api
    spec:
      containers:
        - name: api-container
          image: ahm282/milestone2:fastapi
          imagePullPolicy: Always
          ports:

```

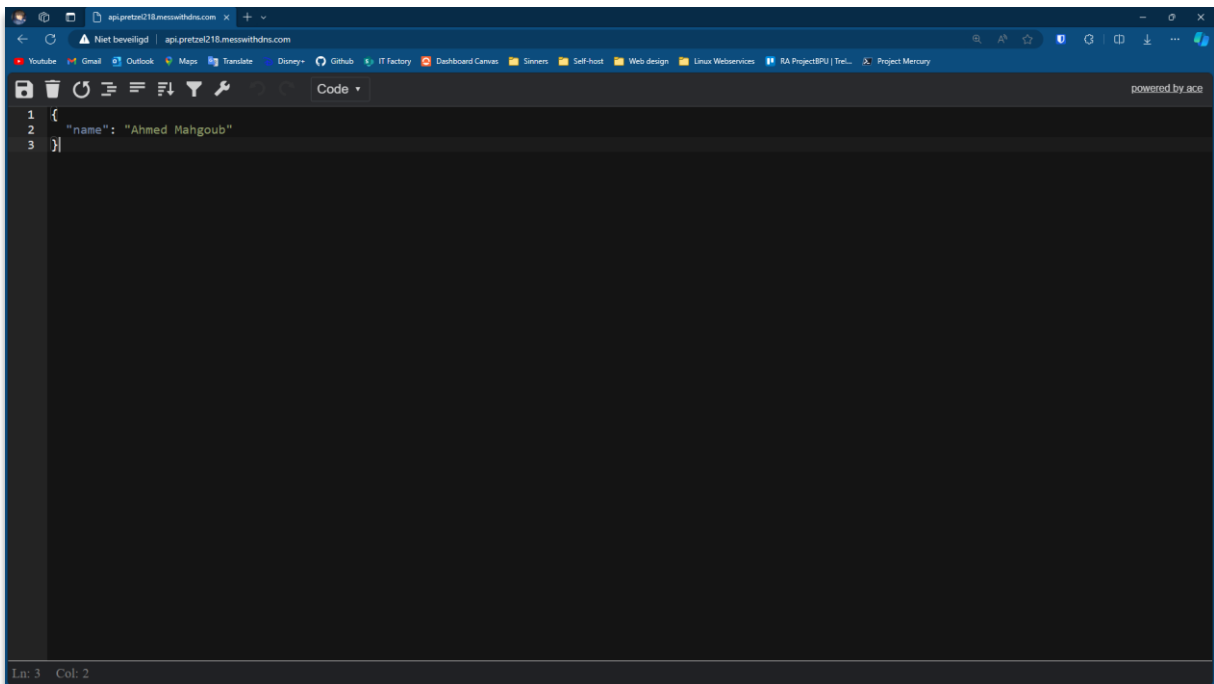
```
        - containerPort: 8000
      volumeMounts:
        - name: main-script
          mountPath: /code
      volumes:
        - name: main-script
          hostPath:
            path: /vagrant/deployments/api/app
            type: Directory
```

Api-service.yaml:

```
apiVersion: v1
kind: Service
metadata:
  name: api-service
spec:
  selector:
    app: api
  ports:
    - protocol: TCP
      port: 8000
      targetPort: 8000
  type: ClusterIP
```

Api-ingress.yaml:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: api-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: "nginx"
spec:
  rules:
    - host: api.pretzel218.messwithdns.com
      http:
        paths:
          - pathType: Prefix
            path: /
            backend:
              service:
                name: api-service
                port:
                  number: 8000
```



4.3. PostgreSQL

Het installeren van de PostgreSQL database verschilde niet veel van de voorgaande stappen. Wat anders was aan deze installatie waren de volumes en het init-script dat gebruikt werd om de tabel aan te maken en er een record in te voegen.

Helaas ontbreekt storage provisioning in mijn kubernetes cluster, waardoor ik geen persistent volumes (claims) kan gebruiken. Dit maakt mijn gegevens in principe efemeer. Om de database draaiende te krijgen en dit probleem te omzeilen, heb ik een emptyDir mount gebruikt voor de databasegegevens.

Postgres-deployment :

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: postgres-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: postgres
  template:
    metadata:
      labels:
        app: postgres
    spec:
      containers:
        - name: postgres-container
          image: postgres:latest
          env:
            - name: POSTGRES_USER
              value: kube
            - name: POSTGRES_PASSWORD
              value: kube
            - name: POSTGRES_DB
              value: milestone
          ports:
            - containerPort: 5432
          volumeMounts:
            - name: psql-data
              mountPath: /var/lib/postgresql/data
            - name: init-scripts
              mountPath: /docker-entrypoint-initdb.d
      volumes:
```

```
- name: psql-data
  emptyDir: {}
- name: init-scripts
  hostPath:
    path: /vagrant/deployments/postgres/init-scripts
```

Postgres-service:

```
apiVersion: v1
kind: Service
metadata:
  name: postgres-service
spec:
  selector:
    app: postgres
  ports:
    - protocol: TCP
      port: 5432
      targetPort: 5432
```

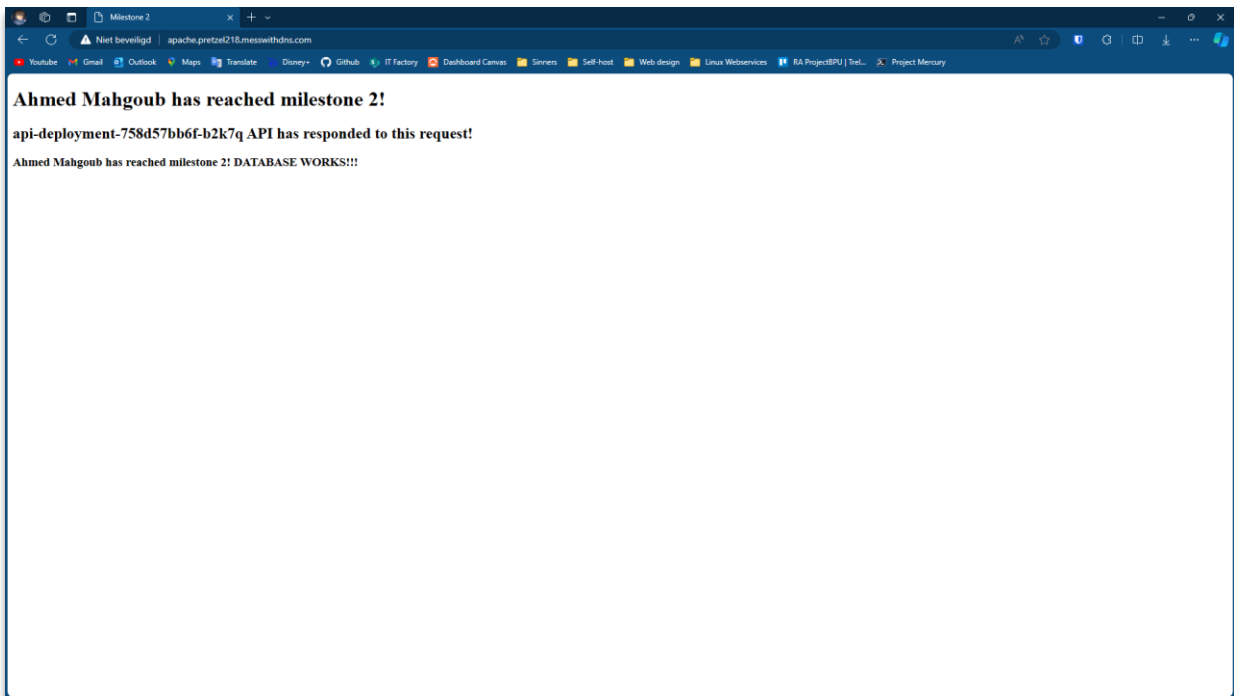
init.sql:

```
\c milestone;

-- Create tables and perform other initialization tasks
CREATE TABLE fullname (
  id SERIAL PRIMARY KEY,
  name VARCHAR(255)
);

-- Insert some initial data
INSERT INTO fullname (name) VALUES ('Ahmed Mahgoub');
```

Eindresultaat:



5. Prometheus

Om Prometheus te installeren en te beginnen met het monitoren van mijn cluster, heb ik Helm geïnstalleerd. Helm is een pakketbeheerder voor Kubernetes-applicaties. Het vereenvoudigt de implementatie en het beheer van Kubernetes-applicaties door een high-level abstractie te bieden die 'charts' wordt genoemd. Een Helm-chart is een vooraf geconfigureerd pakket met Kubernetes-resources die eenvoudig kunnen worden ingezet op een Kubernetes-cluster.

De installatie was vrij eenvoudig, ik hoefde alleen maar een script van hun website te downloaden en uit te voeren. Daarna ging ik naar de Artifact Hub en zocht naar de Prometheus Helm installatiecommando's.

```
curl -fsSL -o get_helm.sh
https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3
chmod 700 get_helm.sh
./get_helm.sh
```

```
helm repo add prometheus-community https://prometheus-community.github.io/helm-
charts
helm repo update
helm install my-prometheus prometheus-community/prometheus --version 25.8.1 -f
prometheus/values.yaml
```

5.1. Values.yaml

Na het installeren van de helm chart start Prometheus een persistent volume claim (PVC). Maar zoals hierboven vermeld, werken PVCs momenteel niet op mijn cluster. Dus moest ik een bestand toevoegen waarin ik de parameters specificeer die ik nodig heb om mijn installatie aan te passen.

Ik heb PVC uitgeschakeld, waardoor Prometheus een emptyDir met efemere gegevens gebruikt. Ik specificeer ook de ingress in hetzelfde bestand.

Values.yaml:

```
server:
  persistentVolume:
    enabled: false
  ingress:
    enabled: true
    hosts:
      - prometheus.pretzel218.messwithdns.com
```

5.2. Automatisatie

Op weg naar de voltooiing van dit project ging ik door een proces van "trial and error" en moest ik veel problemen oplossen. Elk bestand handmatig deployen was erg tijdrovend, dus maakte ik een script om dit deel te automatiseren.

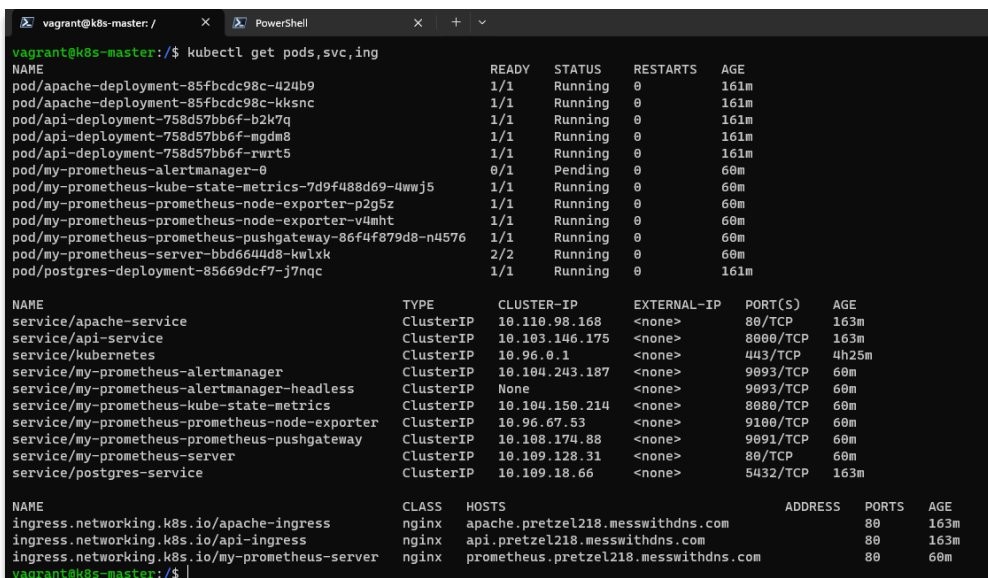
```
#!/bin/bash

# Apply Apache YAMLs
kubectl apply -f apache/apache-deployment.yaml
kubectl apply -f apache/apache-service.yaml
kubectl apply -f apache/apache-ingress.yaml

# Apply API YAMLs
kubectl apply -f api/api-deployment.yaml
kubectl apply -f api/api-service.yaml
kubectl apply -f api/api-ingress.yaml

# Apply Postgres YAMLs
kubectl apply -f postgres/postgres-deployment.yaml
kubectl apply -f postgres/postgres-ingress.yaml

# Deploy Prometheus
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
helm repo update
helm install my-prometheus prometheus-community/prometheus --version 25.8.1 -f prometheus/values.yaml
```



```
vagrant@k8s-master: /$ kubectl get pods,svc,ing
NAME                                READY   STATUS    RESTARTS   AGE
pod/apache-deployment-85fbcdc98c-424b9   1/1     Running   0           161m
pod/apache-deployment-85fbcdc98c-kksnc   1/1     Running   0           161m
pod/api-deployment-758d57bb6f-b2k7q     1/1     Running   0           161m
pod/api-deployment-758d57bb6f-mgdm8     1/1     Running   0           161m
pod/api-deployment-758d57bb6f-rwrts     1/1     Running   0           161m
pod/my-prometheus-alertmanager-0        0/1     Pending   0           60m
pod/my-prometheus-kube-state-metrics-7d9f488d69-4wwj5  1/1     Running   0           60m
pod/my-prometheus-prometheus-node-exporter-p2g5z  1/1     Running   0           60m
pod/my-prometheus-prometheus-node-exporter-v4mht  1/1     Running   0           60m
pod/my-prometheus-prometheus-pushgateway-86f4f879d8-n4576  1/1     Running   0           60m
pod/my-prometheus-server-bbd6644d8-kwLxk  2/2     Running   0           60m
pod/postgres-deployment-85669dcf7-j7nqc  1/1     Running   0           161m

NAME                                TYPE               CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
service/apache-service              ClusterIP          10.110.98.168   <none>           80/TCP           163m
service/api-service                 ClusterIP          10.183.146.175   <none>           8000/TCP         163m
service/kubernetes                  ClusterIP          10.96.0.1        <none>           443/TCP          4h25m
service/my-prometheus-alertmanager  ClusterIP          10.184.243.187   <none>           9093/TCP         60m
service/my-prometheus-alertmanager-headless  ClusterIP          None             <none>           9093/TCP         60m
service/my-prometheus-kube-state-metrics  ClusterIP          10.184.150.214   <none>           8080/TCP         60m
service/my-prometheus-prometheus-node-exporter  ClusterIP          10.96.67.53      <none>           9100/TCP         60m
service/my-prometheus-prometheus-pushgateway  ClusterIP          10.188.174.88    <none>           9091/TCP         60m
service/my-prometheus-server         ClusterIP          10.189.128.31    <none>           80/TCP           60m
service/postgres-service            ClusterIP          10.189.18.66     <none>           5432/TCP         163m

NAME                                CLASS  HOSTS                                ADDRESS          PORTS  AGE
ingress.networking.k8s.io/apache-ingress  nginx  apache.pretzel218.messwithdns.com  80              163m
ingress.networking.k8s.io/api-ingress     nginx  api.pretzel218.messwithdns.com     80              163m
ingress.networking.k8s.io/my-prometheus-server  nginx  prometheus.pretzel218.messwithdns.com  80              60m
vagrant@k8s-master: /$
```

```
vagrant@k8s-master: /vagrant$ helm install my-prometheus prometheus-community/prometheus --version 25.8.1 -f prometheus/values.yaml
NAME: my-prometheus
LAST DEPLOYED: Sat Dec 9 02:21:45 2023
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
The Prometheus server can be accessed via port 80 on the following DNS name from within your cluster:
my-prometheus-server.default.svc.cluster.local

From outside the cluster, the server URL(s) are:
http://prometheus.pretzel218.nesswithdns.com

The Prometheus alertmanager can be accessed via port 9093 on the following DNS name from within your cluster:
my-prometheus-alertmanager.default.svc.cluster.local

Get the Alertmanager URL by running these commands in the same shell:
export POD_NAME=$(kubectl get pods --namespace default -l "app.kubernetes.io/name=alertmanager,app.kubernetes.io/instance=my-prometheus" -o jsonpath="{.items[0].metadata.name}")
kubectl --namespace default port-forward $POD_NAME 9093
##### WARNING: Pod Security Policy has been disabled by default since #####
##### it deprecated after k8s 1.25+, use #####
##### (index.Values "prometheus-node-exporter" "rbac" #####
##### . #####
##### "pspEnabled" with (index .Values #####
##### "prometheus-node-exporter" "rbac" "pspAnnotations") #####
##### In case you still need it, #####
##### #####

The Prometheus PushGateway can be accessed via port 9091 on the following DNS name from within your cluster:
my-prometheus-prometheus-pushgateway.default.svc.cluster.local

Get the PushGateway URL by running these commands in the same shell:
export POD_NAME=$(kubectl get pods --namespace default -l "app.prometheus-pushgateway,component=pushgateway" -o jsonpath="{.items[0].metadata.name}")
kubectl --namespace default port-forward $POD_NAME 9091

For more information on running Prometheus, visit:
https://prometheus.io/
vagrant@k8s-master: /vagrant/deployments$
```

The screenshot shows the Prometheus web interface with a query: `kube_pod_info{node="k8s-node01", namespace="default"}`. The results table displays various pod information metrics.

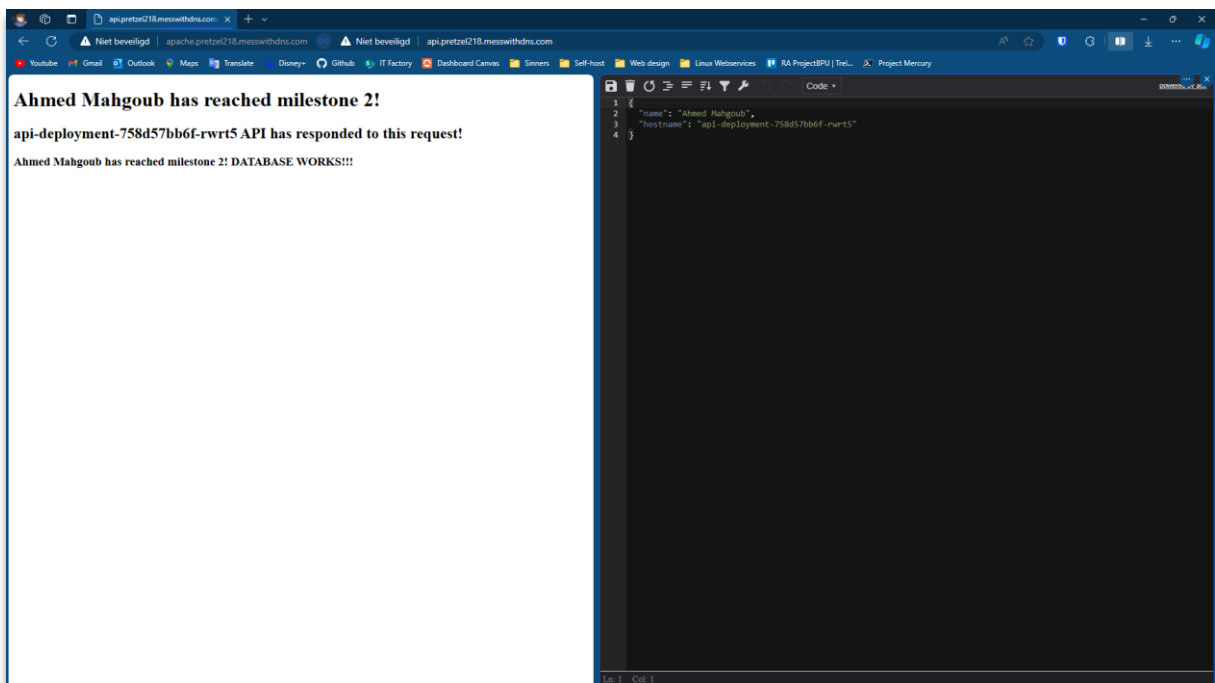
| Pod Name | Node | Namespace | Service | IP | Host Network | Instance | Job |
|--|------------|-----------|----------------------------------|---------------|--------------|--------------------|------------------------------|
| my-prometheus-prometheus-node-exporter-v4mtr | k8s-node01 | default | my-prometheus-kube-state-metrics | 172.17.125.23 | true | 172.17.125.23:8080 | kubernetes-service-endpoints |
| spache-deployment-858cd9c6-2489 | k8s-node01 | spache | my-prometheus-kube-state-metrics | 172.17.125.13 | false | 172.17.125.23:8080 | kubernetes-service-endpoints |
| spache-deployment-858cd9c6-2489 | k8s-node01 | spache | my-prometheus-kube-state-metrics | 172.17.125.13 | false | 172.17.125.23:8080 | kubernetes-service-endpoints |
| api-deployment-758d57bb6f | k8s-node01 | api | my-prometheus-kube-state-metrics | 172.17.125.20 | false | 172.17.125.23:8080 | kubernetes-service-endpoints |
| api-deployment-758d57bb6f | k8s-node01 | api | my-prometheus-kube-state-metrics | 172.17.125.14 | false | 172.17.125.23:8080 | kubernetes-service-endpoints |
| api-deployment-758d57bb6f | k8s-node01 | api | my-prometheus-kube-state-metrics | 172.17.125.14 | false | 172.17.125.23:8080 | kubernetes-service-endpoints |
| postgres-deployment-85669cd77 | k8s-node01 | postgres | my-prometheus-kube-state-metrics | 172.17.125.13 | false | 172.17.125.23:8080 | kubernetes-service-endpoints |

6. Conclusie

Samenvattend, het opzetten van een werkende web stack in een Kubernetes-cluster voor het vak Linux Web Services was een grondige duik in de wereld van containerorkestratie en gedistribueerde systemen. Het was meer dan alleen een technische oefening; het was een pragmatische verkenning van de complexiteit die gepaard gaat met het beheren van moderne applicaties.

Van het configureren van Kubernetes-clusters met Kubeadm tot het begrijpen van deployments, services en Ingress, was het een proces van stapsgewijs navigeren door de technologische lagen. Elk onderdeel, of het nu de Apache-deployment, FastAPI-service of de PostgreSQL-configuratie betrof, voelde als een puzzelstukje dat moest passen in het grotere geheel van een functionerende webstack.

Terwijl ik terugkijk op deze ervaring, waardeer ik de nuchtere aanpak van het realiseren van een werkende infrastructuur. Het project heeft niet alleen mijn technische vaardigheden aangescherpt, maar heeft ook een realistisch beeld geschetst van de uitdagingen en mogelijkheden in de wereld van moderne IT.



Bibliografie

1. HashiCorp. (2023, 10 augustus). *Vagrant by HashiCorp*. HashiCorp. <https://www.vagrantup.com/>
2. HashiCorp. (2023, 28 oktober). *Vagrantfile | Vagrant | HashiCorp Developer*. HashiCorp. <https://developer.hashicorp.com/vagrant/docs/vagrantfile>
3. HashiCorp. (2023, 28 oktober). *Command-Line Interface | Vagrant | HashiCorp Developer*. HashiCorp. <https://developer.hashicorp.com/vagrant/docs/cli>
4. HashiCorp. (2023, 28 oktober). *Vagrantfile | Vagrant | HashiCorp Developer*. HashiCorp. <https://developer.hashicorp.com/vagrant/docs/vagrantfile>

AFBEELDINGEN

1. HashiCorp. (2023, 28 Oktober). *HashiCorp: Infrastructure Enables Innovation*. HashiCorp. <https://www.hashicorp.com/brand>
2. Wikipedia-bijdragers. (2023, 5 Juni). *Kubernetes*. Wikipedia. <https://nl.wikipedia.org/wiki/Kubernetes>
3. Velayudhan, N. (2022, 4 Februari). *Kubernetes Architecture — Deep Dive - techbeatly - Medium*. Medium. <https://medium.com/techbeatly/kubernetes-architecture-deep-dive-520218da0a26>

Bijlagen

Mappenstructuur

