



Milestone 2

Linux Webservices ITFactory

Mathias Wouters 2CCS02

Academiejaar 2022-2023

Campus Geel, Kleinhoefstraat 4, BE-2440 Geel

Inhoudstafel

INLEIDING	4
1 DOCKERFILE WEBSERVER	5
2 DOCKERFILE FASTAPI	6
3 API PYTHON FILE.....	7
4 CONFIGURATION FILE WEBSERVER (NGINX)	9
5 CONFIGURATION FILE FASTAPI.....	11
6 CONFIGURATION FILE DATABASE (MONGODB)	13
7 VAGRANT	15
8 STARTUP	16
9 DATABASE INVULLEN.....	17
10 PROMOTHEUS	19
11 GITOPS WORKFLOW	21
12 BESLUIT	22

INLEIDING

Dit is mijn verslag van Milestone 2 voor het vak Linux Webservices. Het minimale doel dat ik moest waarmaken voor deze milestone was om een kubernetes cluster te maken (ik gebruik hiervoor microk8s) die dat mijn stack runt. Mijn stack bestond uit een NGINX webserver, een FastAPI service en een MongoDB database. In totaal heb ik 2 custom dockerfiles gemaakt voor nginx en voor FastAPI, die dat ik ook via een GitOps workflow automatisch naar docker hub push zodat ik deze terug kan pullen naar mijn kubernetes configuratie files. Ik gebruik in totaal ook 2 VM's omdat ik dit het makkelijkste vond om te loadbalancing toe te kunnen passen. Eén VM is eigenlijk de master en een worker in één, en de andere is gewoon een tweede worker. Deze VM's run ik via vagrant waarbij ik ook gebruik maak van installatie scripts zodat ik automatisch de nodige installaties kan doen.

Voor kubernetes gebruik ik 3 configuratie YAML files waarbij ik voor nginx en voor FastAPI mijn custom dockerfiles pull van de docker hub. Voor MongoDB gebruik ik de officiële dockerfile die dat op docker hub staat. Dan heb ik nog mijn API waarbij ik verbinding maak met de MongoDB database, die dan uit die database de ingevoerde naam haalt en deze op de website zet.

Ik gebruik ook een loadbalancer genaamd MetalLB deze zit standaard bij MicroK8s, ik heb metallb eerst vergeleken met ingress en ik vond metallb precies makkelijker om te gebruiken en daarom heb ik deze dan ook gekozen. Ik heb ook Prometheus geïnstalleerd om mijn cluster te kunnen monitoren.

1 DOCKERFILE WEBSERVER

```
1  # Pull the minimal Ubuntu image
2  FROM ubuntu:22.04
3
4  # Install Nginx
5  RUN apt update -y && apt upgrade -y
6  RUN apt-get -y install nginx
7  RUN apt install -y nodejs
8
9  # Expose the port for access
10 EXPOSE 80/tcp
11
12 COPY ./src/index.html /var/www/html
13
14 # Run the Nginx server
15 CMD ["/usr/sbin/nginx", "-g", "daemon off;"]
```

Line 2: Ubuntu versie 22.04 word gepulled van docker hub en geïnstalleerd.

Line 5-7: Ubuntu wordt geüpdatet en ik installeer de nginx webserver. Ten slotte installeer ik ook nog node js omdat dit nodig was om javascript te kunnen gebruiken op mijn pagina.

Line 10: Zegt tegen docker dat mijn webserver op poort 80 zal luisteren via het tcp protocol.

Line 12: Kopieert de custom webpagina die ik lokaal op mijn host machine heb staan naar de container.

Line 15: commando om nginx te starten.

2 DOCKERFILE FASTAPI

```
1 FROM python:3.10.0-alpine You, 3 days ago • commit
2 WORKDIR /code
3 EXPOSE 8000
4 COPY ./requirements.txt /code/requirements.txt
5 RUN pip install --no-cache-dir --upgrade -r /code/requirements.txt
6 COPY ./app /code/app
7 CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8000"]
```

Line 1: Pull de alpine image van docker hub waar ik python op ga installeren.

Line 2: maakt de /code directory aan in de container.

Line 3: Opent poort 8000 die dat de API zal gebruiken.

Line 4: kopieert de requirements txt file naar de container. Hierin staan verschillende libraries enzo die dat de API zal gebruiken.

Line 5: Zorgt dat gedownload packages niet in het systeem cachen, maar dat dit in de requirements.txt wordt gedaan.

Line 6: Kopieert het python programma dat lokaal staat naar de container.

Line 7: Voert het python programma uit op de gespecificeerde poort.

3 API PYTHON FILE

```

1  ✓ from fastapi import FastAPI      You, 3 days ago *
2  from pymongo import MongoClient
3  from fastapi.middleware.cors import CORSMiddleware
4
5  app = FastAPI()
6
7  ✓ origins = [
8      "http://localhost/",
9      "http://localhost:8080/",
10     "https://localhost.tiangolo.com/",
11     "http://127.0.0.1:5500/",
12     "http://192.168.56.51:8000/users",
13     "http://192.168.56.50/",
14     "*"
15 ]
16
17 ✓ app.add_middleware(
18     CORSMiddleware,
19     allow_origins=origins,
20     allow_credentials=True,
21     allow_methods=[""],
22     allow_headers=[""],
23 )
24
25 client = MongoClient("mongodb://mongo:27017")
26 mydb=client["milestone2-mw"]
27 mycol=mydb["users"]
28
29 #test get hello world
30 @app.get("/")
31 ✓ async def root():
32     return {"message : Hello World"}
33
34 @app.get("/users")
35 ✓ async def test():
36     users = mycol.find()
37     ✓ for user in users:
38         item = user["name"]
39     return {"name": item}

```


Line 1-3: De nodige libraries worden hier geïnstalleerd.

Line 5:

Line 7-23: Dit is nodig om de interactie tussen mijn webpagina en de API waar te kunnen maken. Ik geef eerst de adressen mee waarmee mijn API contact mag mee maken. En dan zeg ik dat CORS (Cross-Origin Resource Sharing) is toegelaten met de gespecificeerde adressen.

Line 25-27: Hier maak ik een connectie met de MongoDB database, waarbij ik ook een client meegeef. Ik zeg ook dat ik de "users" collection moet gebruiken. Binnen in MongoDB gebruiken ze collections i.p.v tables.

Line 30-32: Ik geef hier gewoon een print mee zodat ik telkens makkelijk kan zien of dat mijn API ook daadwerkelijk werkt.

Line 34-39: Hier laat ik de data zien vanuit de database die dat in de users collection staat. Dit laat steeds de laatste toegevoegde waarde zien waardoor het niet echt uitmaakt of dat je de waarden in de database gaat updaten of gewoon een nieuwe toevoegt.

4 CONFIGURATION FILE WEBSERVER (NGINX)

```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: nginx
5    labels:
6      app: nginx
7  spec:
8    replicas: 2
9    selector:
10     matchLabels:
11       app: nginx
12    template:
13     metadata:
14       labels:
15         app: nginx
16     spec:
17       containers:
18         - name: nginx
19           image: r0879842/milestone2-github-webserver:latest
20           ports:
21             - containerPort: 80
22 ---
23  apiVersion: v1
24  kind: Service
25  metadata:
26    name: nginx
27    annotations:
28      service.beta.kubernetes.io/linode-loadbalancer-throttle: "4"
29    labels:
30      app: nginx
31  spec:
32    type: LoadBalancer
33    ports:
34      - name: http
35        protocol: TCP
36        port: 80
37        targetPort: 80
38    selector:
39      app: nginx
```

Line 2: Eerst maak ik de deployment configuratie voor nginx.

Line 8: zegt dat er altijd 2 identieke pods moeten runnen zodat er geen downtime kan zijn als een pod failed of crasht.

Line 12-15: Geeft een template mee zodat kubernetes altijd weet hoe dat de pods er moeten uitzien als er eventueel meer moeten worden bijgemaakt.

Line 16-21: Hier zeg ik dat mijn custom docker image gebruikt moet worden die ik op docker hub heb staan. Ik geef ook de poort mee die moet worden opengezet om nginx te kunnen gebruiken op de host machine.

Line 24: Nu maak ik de service configuratie aan, het verschillen tussen deployment en service is dat de deployment ervoor zorgt dat de pods actief blijven, en de service zorgt voor een netwerk toegang waardoor pods onderling met elkaar kunnen communiceren.

Line 27-28: Dit limiteert het aantal nieuwe connecties van de host ip naar 4

Line 31-39: Hier geef ik mee dat dit een loadbalancer wordt zodat ik een extern ip adres krijg van deze container. Hierdoor kan ik met de gespecificeerde poort de webpagina op de webserver raadplegen vanaf mijn host machine.

5 CONFIGURATION FILE FASTAPI

```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: fastapi
5    labels:
6      app: fastapi
7  spec:
8    replicas: 2
9    selector:
10     matchLabels:
11       app: fastapi
12    template:
13     metadata:
14       labels:
15         app: fastapi
16     spec:
17       containers:
18         - name: fastapi
19           image: r0879842/milestone2-github-api:latest
20           ports:
21             - containerPort: 8000
22 ---
23  apiVersion: v1
24  kind: Service
25  metadata:
26    name: fastapi
27    annotations:
28      service.beta.kubernetes.io/linode-loadbalancer-throttle: "4"
29    labels:
30      app: fastapi
31  spec:
32    type: LoadBalancer
33    ports:
34      - name: http
35        protocol: TCP
36        port: 8000
37        targetPort: 8000
38    selector:
39      app: fastapi
```

Ik gebruik hier ongeveer dezelfde configuratie file als die van nginx. Alleen heb ik hier de namen verandert naar fastapi en pull ik de custom API dockerfile van docker hub. Poort 80 is ook verandert naar poort 8000 omdat er anders een conflict zou zijn want dan zouden 2 verschillende services dezelfde poort gebruiken.

6 CONFIGURATION FILE DATABASE (MONGODB)

```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: mongo
5  spec:
6    selector:
7      matchLabels:
8        app: mongo
9    replicas: 1
10   template:
11     metadata:
12       labels:
13         app: mongo
14     spec:
15       containers:
16         - name: mongo
17           image: mongo:4.4.6
18           imagePullPolicy: Always
19           ports:
20             - containerPort: 27017
21   ---
22   apiVersion: v1
23   kind: Service
24   metadata:
25     name: mongo
26   spec:
27     ports:
28       - protocol: TCP
29         port: 27017
30         targetPort: 27017
31     selector:
32       app: mongo
```

Ook deze configuratie is ongeveer hetzelfde als de vorige alleen gebruik hier de officiële docker image van MongoDB omdat ik hier eigenlijk niet extra bij moest installeren. Ik zet ook de default poort voor MongoDB open namelijk 27017. Ik gebruik ook de "imagePullPolicy" tag. Hierbij staat Always, dat wil zeggen dat bij elke update van de container de image terug opnieuw van de docker hub gepulled wordt.

7 VAGRANT

```
1  #!/bin/bash
2  sudo apt-get update
3  #sudo update-alternatives --config iptables
4
5  #install docker on vagrant vm
6  yes | wget -O docker.sh https://get.docker.com/
7  bash docker.sh
8
9  #give vagrant user sudo rights
10 1 | sudo usermod -aG docker vagrant
11 newgrp docker
12
13 #install kubernetes (microk8s)
14 sudo snap install microk8s --classic
15 microk8s status --wait-ready
16 sudo usermod -a -G microk8s vagrant
```

Line 6-7: Dit zijn de commando's voor de installatie van docker.

Line 10-11: Voeg de user vagrant toe aan de docker group zodat ik niet altijd sudo moet gebruiken.

Line 14-16: Microk8s wordt hier geïnstalleerd. Hier voeg ik de user vagrant ook toe tot aan de kubernetes group.

8 STARTUP

Na het starten van mijn VM's begin ik eerst met het enablen van metallb (mijn loadbalancer), hierbij geef ik ook een ip-range mee die dat de loadbalancer mag gebruiken. Dit doe ik via dit commando "**microk8s enable metallb:192.168.56.50-192.168.56.60**".

Vervolgens voeg ik de ip en hostnaam van worker 2 toe in de /etc/hosts file van mijn master via dit commando "**sudo nano /etc/hosts**". Daarna voer ik "**microk8s add-node**" uit op mijn master en krijg ik een join commando terug die dat ik dan moet uitvoeren op mijn worker 2 vm zodat deze dezelfde cluster gaat joinen. Om dit te controleren of dat het gelukt is gebruik ik het commando "**microk8s kubectl get nodes**".

```
vagrant@worker1-mathiaswouters:~$ microk8s kubectl get no
NAME                                STATUS    ROLES    AGE   VERSION
worker1-mathiaswouters             Ready     <none>    18m   v1.25.4
worker2-mathiaswouters             Ready     <none>    17s   v1.25.4
vagrant@worker1-mathiaswouters:~$
```

Daarna voer ik mijn 3 deployment files uit via deze commando's:

- "**microk8s kubectl apply -f deployment.yaml**"
- "**microk8s kubectl apply -f deployment-fastapi.yaml**"
- "**microk8s kubectl apply -f deployment-mongodb.yaml**"

```
vagrant@worker1-mathiaswouters:~/milestone2$ microk8s kubectl get pods
NAME                                READY    STATUS    RESTARTS   AGE
nginx-7589c65cfb-zfwj7             1/1      Running   0           2m41s
mongo-c5c95d74f-scxbz             1/1      Running   0           2m34s
fastapi-b45565dbf-c2w9j           1/1      Running   0           2m25s
nginx-7589c65cfb-4qwcw             1/1      Running   0           2m41s
fastapi-b45565dbf-7tdd7           1/1      Running   0           2m25s
```

Om vervolgens de web pagina te kunnen bekijken moet ik naar het extern ip adres van de loadbalancer gaan. Die vind ik door het commando "**microk8s kubectl get svc -o wide**" uit te voeren.

```
vagrant@worker1-mathiaswouters:~/milestone2$ microk8s kubectl get svc -o wide
NAME      TYPE          CLUSTER-IP    EXTERNAL-IP    PORT(S)          AGE   SELECTOR
kubernetes ClusterIP    10.152.183.1   <none>          443/TCP          19m   <none>
nginx     LoadBalancer 10.152.183.253 192.168.56.50  80:31023/TCP     21s   app=nginx
mongo     ClusterIP     10.152.183.177 <none>          27017/TCP        14s   app=mongo
fastapi   LoadBalancer 10.152.183.59  192.168.56.51  8000:30055/TCP   5s    app=fastapi
vagrant@worker1-mathiaswouters:~/milestone2$
```

9 DATABASE INVULLEN

```
vagrant@worker1-mathiaswouters:~/milestone2$
vagrant@worker1-mathiaswouters:~/milestone2$
vagrant@worker1-mathiaswouters:~/milestone2$ microk8s kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-7589c65cfb-zfwj7              1/1     Running   0           7m18s
mongo-c5c95d74f-scxbz               1/1     Running   0           7m11s
fastapi-b45565dbf-c2w9j             1/1     Running   0           7m2s
nginx-7589c65cfb-4qwcw              1/1     Running   0           7m18s
fastapi-b45565dbf-7tdd7             1/1     Running   0           7m2s
vagrant@worker1-mathiaswouters:~/milestone2$ microk8s kubectl exec -it mongo-c5c95d74f-scxbz sh
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
# mong
sh: 1: mong: not found
# mongo
MongoDB shell version v4.4.6
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("be21dbf3-0f3b-47b9-b58b-217ad9c85512") }
MongoDB server version: 4.4.6
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
  https://docs.mongodb.com/
Questions? Try the MongoDB Developer Community Forums
  https://community.mongodb.com
---
The server generated these startup warnings when booting:
  2022-12-10T11:00:21.031+00:00: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine. See http://dochub.mongodb.org/core/prodnotes-filesystem
  2022-12-10T11:00:22.001+00:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
---
---
  Enable MongoDB's free cloud-based monitoring service, which will then receive and display
  metrics about your deployment (disk utilization, CPU, operation statistics, etc).

  The monitoring data will be available on a MongoDB website with a unique URL accessible to you
  and anyone you share the URL with. MongoDB may use this information to make product
  improvements and to suggest MongoDB products and deployment options to you.

  To enable free monitoring, run the following command: db.enableFreeMonitoring()
  To permanently disable this reminder, run the following command: db.disableFreeMonitoring()

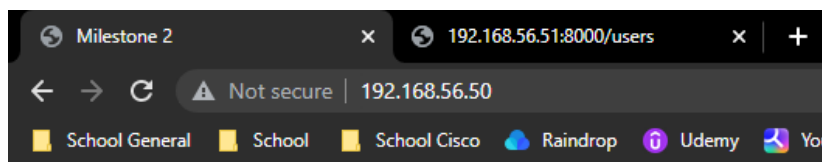
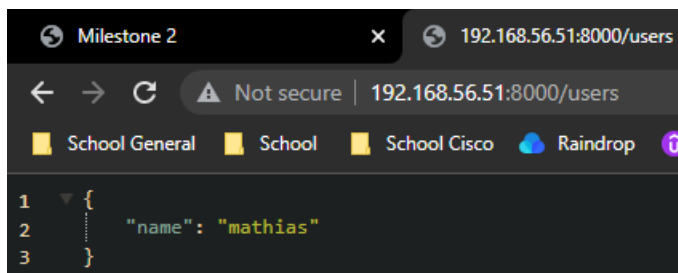
> use milestone2-mw
switched to db milestone2-mw
> db.createCollection("users")
{ "ok" : 1 }
> db.users.insert({"name":"mathias"})
WriteResult({ "nInserted" : 1 })
> db.users.update({"name":"mathias"}, {$set:{"name":"Mathias Wouters"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

Mijn eerste commando is om een shell sessie te starten in mijn MongoDB pod.

Daarna gebruik ik het commando "mongo" om te connecteren met de MongoDB Shell.

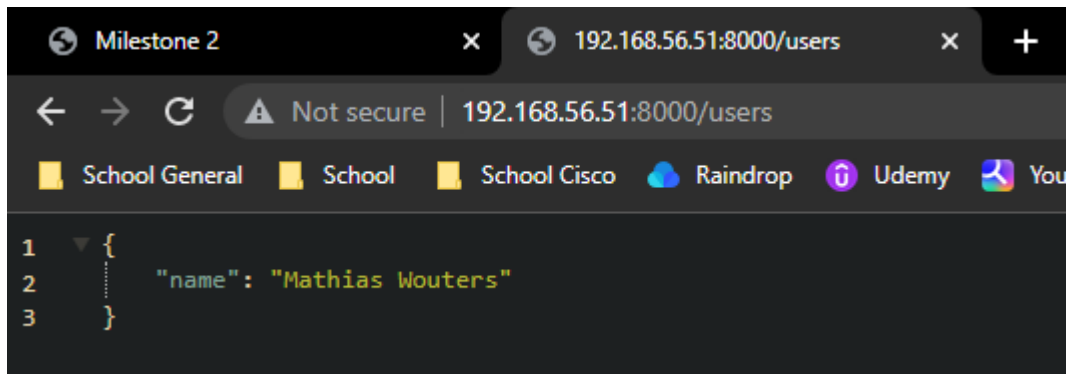
Vervolgens login ik als gebruiker "milestone2-mw" en maak ik een collectie genaamd "users".

In deze collectie voeg ik een waarde toe aan de document "name".

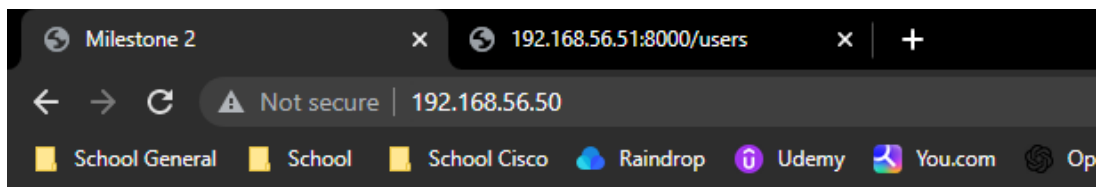


mathias has reached milestone 2!

Als laatste update ik de waarde die al in het document "name" stond met dit als uitkomst:



```
1 {  
2   "name": "Mathias Wouters"  
3 }
```

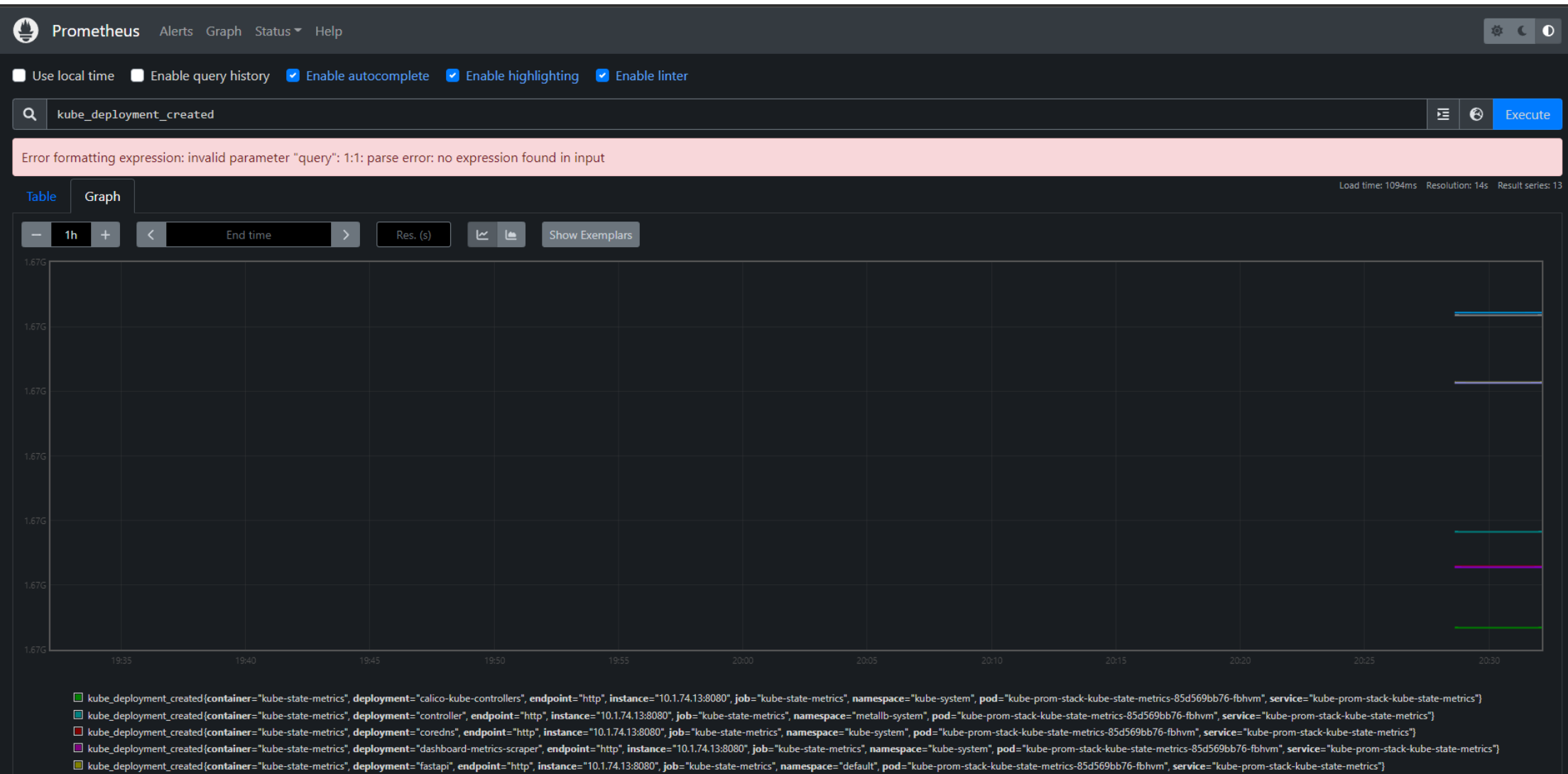


Mathias Wouters has reached milestone 2!

10 PROMOTHEUS

Prometheus is software om een kubernetes cluster te kunnen monitoren. Deze is een default add-on die bij microk8s standaard wordt meegegeven. Dus ik moest dit eerst gewoon enablen via dit commando "`microk8s enable prometheus`". Om prometheus extern te kunnen gebruiken vanaf mijn host machine heb ik eerst moeten port forwarding met dit commando "`microk8s kubectl port-forward -n observability service/prometheus-operated --address 0.0.0.0 9090:9090`".

Op de foto (volgende pagina) heb ik gefilterd op de deployments. Elke deployment heeft een andere kleur gekregen en dan kan je zien wanneer deze is aangemaakt en dan ook wanneer deze offline worden gehaald.



11 GITOPS WORKFLOW

Ik maak gebruik van dezelfde GitOps workflow die dat we in het vak API Development hebben gebruikt. Ik push mijn lokaal gemaakte repo's waarin mijn dockerfiles staan naar GitHub. Dan geef ik mee via een yaml file wat er allemaal juist moet gebeuren.

```

1  name: Deliver container      You, 5 days ago • commit ...
2
3  on: push
4
5  jobs:
6    delivery:
7      runs-on: ubuntu-latest
8      name: Deliver container
9      steps:
10     - name: Check out repository
11       uses: actions/checkout@v1
12
13     - name: Docker login
14       run: docker login -u ${ secrets.DOCKER_USER } -p ${ secrets.DOCKER_PASSWORD }
15
16     - name: Docker Build
17       run: docker build -t ${ secrets.DOCKER_USER }/milestone2-github-webserver:latest .
18
19     - name: Upload container to Docker Hub with Push
20       run: docker push ${ secrets.DOCKER_USER }/milestone2-github-webserver:latest

```

Via Github Actions runt dit proces bij elke push die dat we in deze repo doen. Eerst kijkt hij naar de inhoud van de repo en kopieert dit naar een ubuntu vm. Dan gaat hij inloggen op mijn docker hub account door gebruik te maken van GitHub secrets waarin ik mijn password en gebruikersnaam heb gezet. Deze worden geëncrypteerd opgeslagen op GitHub dus dit is compleet veilig om te doen. Vervolgens doet hij een docker build om de docker container te bouwen zodat deze altijd gebruikt kan worden via de deocker hub. Ten laatste wordt de container geupload naar de docker hub.

12 BESLUIT

Het was een zeer interessant, maar moeilijk project. Omdat het zeker heel moeilijk is om alles juist te vinden hoe het juist moet gebeuren om een cluster met specifiek uw stack te kunnen draaien. Ik was zeer gelukkig toen ik de tekst "mathias has reached milestone 2" op mijn scherm zag, maar toen ik daarna nog eens naar al mijn configuratie files kijk verschoot ik er dan toch van dat ik hier zoveel tijd had ingestoken terwijl je dit makkelijk allemaal in 30-45 minuten kan schrijven en runnen. Ik heb wel enorm veel problemen met vagrant tegen gekomen wat mij dan ook wel echt heel hard frustreerde omdat ik gewoon meer tijd aan vagrant heb besteed dan dat ik echt met kubernetes bezig was.

Maar over het algemeen heb ik enorm veel bijgeleerd over kubernetes en docker dat ik normaal in een les niet zou bijgeleerd hebben en daar ben ik wel enorm blij mee. Want ik denk dat ik hier nog wel leuke projectjes mee kan gaan doen thuis.