Stageverslag

Bachelor toegepaste informatica

Ansible & cloud-init

Student: Maarten De Smedt

Stagebegeleider: Stefaan De Cock

Stagebedrijf: Be-Mobile

Stagementor: Simon Lepla

Academiejaar 2018-2019

Inhoudsopgave

[Voorwoord 3](#_Toc6579475)

[Voostelling bedrijf: Be-Mobile 4](#_Toc6579476)

[Wie zijn Be-Mobile? 4](#_Toc6579477)

[Historiek & Activiteit 4](#_Toc6579478)

[Geografische ligging 5](#_Toc6579479)

[Personeel 5](#_Toc6579480)

[Activiteit 6](#_Toc6579481)

[Business unit stagiair 6](#_Toc6579482)

[Waarom Be-Mobile? 6](#_Toc6579483)

[Opdracht: Ansible & cloud-init 7](#_Toc6579484)

[Beginsituatie 7](#_Toc6579485)

[SMART 8](#_Toc6579486)

[Plan van aanpak 8](#_Toc6579487)

[Uitwerking 8](#_Toc6579488)

[Opzoekwerk 8](#_Toc6579489)

[Lokale testomgeving cloud-init 9](#_Toc6579490)

[Hetzner Cloud testomgeving cloud-init 10](#_Toc6579491)

[Hetzner Cloud testomgeving cloud-init & Ansible 14](#_Toc6579492)

[Kubernetes 16](#_Toc6579493)

[Golang 21](#_Toc6579494)

[API - Demo 24](#_Toc6579495)

[API deployment 29](#_Toc6579496)

[Eindresultaat 29](#_Toc6579497)

[Business doelstellingen 29](#_Toc6579498)

[Persoonlijke doelstellingen 30](#_Toc6579499)

[Eindreflectie 30](#_Toc6579500)

[Bibliografie 30](#_Toc6579501)

[Verklarende woordenlijst 30](#_Toc6579502)

[Bijlage: stagedagboek 31](#_Toc6579503)

# Voorwoord

Gedurende 14 weken heb ik stage gelopen bij Be-Mobile. Dit stageverslag zal bespreken hoe mijn stage verlopen is.

Allereerst is er een voorstelling van het bedrijf met alle info van het bedrijf en wat ze doen. Hierin staat er ook in welke business unit ik werkte. Ook staat er onder dit punt waarom ik koos om bij Be-Mobile stage te doen. Erna vindt u de voorstelling van mijn opdracht “Ansible & cloud-init”, met ook het verloop en resultaat van deze opdracht. Na dit hoofdstuk vindt u wat ik van deze stage vond. Ten laatste is er nog een verklarende woordenlijst, bibliografie met alle bronnen en mijn stagedagboek.

Graag had ik ook Simon Lepla bedankt. Hij was mijn stagementor en heeft dit naar mijn mening uitstekend gedaan. Als ik een vraag had stond hij altijd klaar om te antwoorden en ook gaf hij voldoende vrijheid aan mij om de opdracht op mijn manier uit te voeren. De andere leden van het **devops** team Ziggy en Maarten waren ook altijd hulp vol . Mijn collega stagiair Glenn De Smedt verdient ook een vernoeming. Hij was een zeer aangename collega waarmee ik goed overeenkwam.

Ook had ik graag Stefaan De Cock bedankt voor het begeleiden van de stage. Hij deed dit op een goede, correcte manier.

Ten laatste ben ik Be-Mobile ook zeer dankbaar voor de kans om hier 14 weken stage te doen.

# Voostelling bedrijf: Be-Mobile

## Wie zijn Be-Mobile?

### Historiek & Activiteit

Be-Mobile is ontstaan in 2006 onder de naam TC-Matrix. De huidige CEO Jan Cools en Philip Taillieu namen dit initiatief. Zij zagen dat mobiliteit een gat in de markt was. Eind 2006 kwam de samenwerking met Proximus, die tot op de dag vandaag nog steeds actief is. Be-Mobile is actief in de mobiliteit sector, ze zijn de marktleider in België in deze sector. Het bedrijf heeft verschillende systemen ontwikkelt die verkeersinformatie verkrijgt

In 2007 werd de applicatie Touring Mobilis opgestart. Via deze applicatie kan er live naar de verkeersituatie worden gekeken. Deze informatie en applicatie is ook vrij te gebruiken voor eender wie. Ze is beschikbaar via de site: <https://www.touringmobilis.be/> . Eind 2007 – begin 2008 startte Philip Taillieu het zusterbedrijf flow op. Flow ontwikkelt parkeertechnologieën voor steden.

Begin 2009 werd de applicatie Flitsmeister opgekocht. Flitsmeister is een applicatie die gemaakt is door Nederlanders. De applicatie is wat de voorganger van Waze en Google Maps en nog steeds ontzettend populair in Nederland.

Ondertussen was Be-Mobile ook al sinds het ontstaan van het bedrijf bezig met SMS parking. In 2010 kwam daar ook het betalen van je bus of tram ticket via SMS bij. In 2012 kwam de 4411 app uit. Met de app kan je betalen voor bus of tram tickets en parkeersessies beheren.

Het bedrijf bleef groeien en in 2014 verhuisde naar een nieuwe (de huidige) locatie. In 2015 namen ze ook tolling onder hun hoede. Het bedrijf blijft hierna doorgroeien in alle facetten doormiddel van buitenlandse en binnenlandse overnames en eigen kracht.

Ze gebruiken de verkeersinfo die ze verkrijgen intern of geven ze aan externe. In de historiek van Be-Mobile is al besproken wat voor applicaties ze allemaal hebben uitgebracht en wat ze doen.

Ook geven ze de informatie aan externe. Dan kunnen bedrijven die deze info nodig hebben deze verkrijgen via Be-Mobile. Dat kunnen verkeer informatie services zijn: Garmin, Volvo, Opel,.. Maar ook media kanalen: VRT, Q-Music, De Redactie, Le Soir,… En ook verkeer beheer services: Vlaams Verkeerscentrum, BIVV,..

### Geografische ligging

Het bedrijf situeert zich in Melle. Ze zitten zoals hiervoor al gezegd is nog niet zo heel lang (pas vanaf 2014). Het adres is: Kardinaal Mercierlaan 1a, 9090 Melle.



### Personeel

Bij Mobile werkt er in totaal een kleine 200 man over alle locatie verspreid. Op de hoofdlocatie werkt een 100 man. De werknemers zijn verspreid over 4 business units.

#### Mobile payment services

Dit zijn alle betalingen die worden gedaan over. Publieke transport betalingen, mensen die hun trein of tram ticket betalen met een sms. Parkeer betalingen, de parkeersessie die worden betaald. Dit zijn de 2 hoofd betaling services, maar er zijn ook nog andere kleine betalingen die worden ontvangen. Dit valt hier ook onder.

#### Traffic management services

Deze business unit gaat over het beheren van de verkeergegevens. Hierin worden de verkeersgegevens bekomen en geanalyseerd.

#### Connected vehicles services

Het tolling platform bevindt zicht onder andere onder deze unit. Hierdoor kunnen vrachtwagen chauffeurs automatisch tolling betalen. Ook het traffic platform valt hieronder, dit zorgt er voor dat het verkeer ‘”slim” is. Als er bijvoorbeeld een ambulance snel naar een verkeerslicht komt zal dit op groen springen zodat dit niet wordt gehinderd.

#### Traveller information services

Hierin bevinden alle applicaties zicht ten eerste (zoals 4411).De informatie die ten horen is op de radio over files komt van Be-Mobile. Deze business unit zorgt er voor dat ze die informatie krijgen.

### Activiteit

Be-Mobile is actief in de mobiliteit sector, ze zijn de marktleider in België in deze sector. Het bedrijf heeft verschillende systemen ontwikkelt die verkeersinformatie verkrijgt.

Ze gebruiken deze info dan intern of geven ze aan externe. In de historiek van Be-Mobile is al besproken wat voor applicaties ze allemaal hebben uitgebracht en wat ze doen.

## 

## Business unit stagiair

Ik werk in het devops team onder Simon Lepla. Dit team bevindt zicht eigenlijk niet echt in 1 business unit. Het devops team zorgt ervoor dat alle units goed kunnen functioneren. Ook de taak waar er straks meer uitleg over wordt gegeven is niet voor een bepaalde business unit. Of gebruikt deze om studies uit te voeren.

## Waarom Be-Mobile?

Voor de aanvang van de stage had ik contact gemaakt met een bedrijf. Het ging over de stage en ik zei dat ik nog op zoek was naar een stage plaats. Het bedrijf was zeer enthousiast om een stageplaats voor mij te regelen. Toen het tijd was om op gesprek te gaan, ging ik dus meteen bij dit bedrijf in kwestie, maar de opdracht viel tegen. Het paste niet in het plaatje waar ik naar op zoek was voor een stage en ik moest dus weer vanaf 0 beginnen.

Op de stagetool zocht ik naar verschillende stage bedrijven die mij aanspraken ,en waar er nog plaats was, en daar vond ik verschillende bedrijven. Er waren 3 bedrijven waar nog plaats was die me aanspraken: Be-Mobile, Callebout en atheneum Dendermonde. De stage opdracht van Be-Mobile op de tool sprak me wel het meest aan.

Op de tool stond dat we voor de opdracht in gingen staan voor de migraties van servers. De technologieën waren Docker en **Ansible**, waar ik voor **Linux** mee had gewerkt. Op het gesprek was ik heel snel verkocht. Het onderzoeken naar **Ansible** en **cloud-init** leek me zeer interessant. Ook het leren kennen van een nieuwe technologie als **Kubernetes** was iets dat me zeer hard aansprak. Simon legde ook uit dat Be-Mobile veel **Golang** gebruikt voor het opzetten van servers. Aangezien ik zelf ook wel graag programmeer ten opzichte van andere systeembeheerders voelde ik dat dit iets voor mij was.

De andere bedrijven heb ik niet meer gesproken omdat ik al op voorhand een voorkeur had voor Be-Mobile. Een week later had ik mijn stagedossier ingediend met bevestiging dat ik bij Be-Mobile mijn stage ging doen.

# Opdracht: Ansible & cloud-init

## Beginsituatie

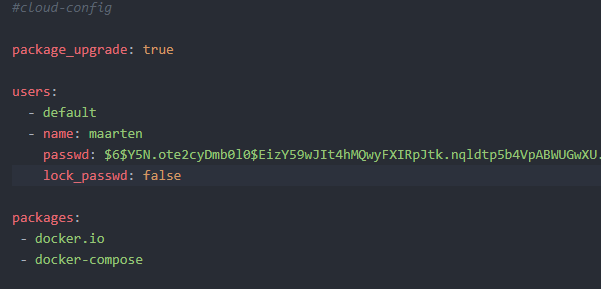
In het bedrijf Be-Mobile werd voor de automatisatie van servers Ansible gebruikt. Anisble is een server configuratie manager die zorgt voor het beheren en deployen van servers. Dit gebeurt via playbooks en rollen.

In een playbook zet je hoe de server moet worden geconfigureerd. Via rollen kan je bepaalde configuraties van servers opslaan om deze later opnieuw aan te roepen als deze nodig zijn. De bestanden van Ansible zijn allemaal yaml bestanden. Hieronder is ene voorbeeld van een playbook.



Maar ze waren op een andere technologie gebotst: cloud-init. Cloud-init is net zoals Ansible een server configuratie manager. Alleen is cloud-init speciaal gemaakt voor cloud systemen. Je kan het er ook wel lokaal mee werken maar dit is niet aangeraden.

Cloud-init maakt gebruik van een cloudconfig bestand. Net zoals ansible is dit een yaml bestand. In dit bestand wordt de configuratie van de servers gedaan. Hieronder is weer een voorbeeld.



Het tweede deel van de opdracht gaat over de eventuele samenwerking of werking met 1 systeem.

Voor het 2de deel moest er gekeken worden om een **API** te maken met golang. Met deze API moest er een worker node toegevoegd worden aan een bestaand kubernetes cluster. De server configuratie moest worden gedaan via Ansible en/of cloud-init. Voor het aanmaken van servers gebruiken ze Hetzner Cloud. Hetzner Cloud is een server provider. Ook het aanmaken van een server op Hetzner Cloud moest in de API verwerkt zijn.

## SMART

Hierin wordt het doel van de opdracht besproken.

Het eerste doel van de opdracht was bekijken of Ansible en cloud-init compatibel zijn.

Een tweede doel dat werd gesteld was om een api te maken doormiddel van golang om ene worker node toe te voegen een kubernetes cluster.

## Plan van aanpak

Hier wordt de plan van aanpak uitgelegd.

Het plan van aanpak werd op voorhand wat voorgelegd door Be-Mobile. Wat wel handig was, ik kreeg altijd verschillende deelopdrachten die ik moest uitvoeren.

Alereerst moest er informatie worden opgezocht over cloud-init, de samenwerking met Ansible en de implementatie op Hetzner Cloud.

Daarna kon ik een lokale testomgeving op zetten met cloud-init. Als dit lukte moest er worden gekeken hoe ik dit op Hetzner Cloud kon doen.

Als dit lokaal allemaal lukte moest ik Ansible combineren met cloud-init.

Hierna moest er een testcluster opgezet worden met kubernetes om dit wat beter te leren kennen. Eerst handmatig dan automatische via cloud-init en/of Ansible.

Daarna moest er worden gewerkt met golang. Met de package van hetzner cloud moesten er servers worden verwijderd, toegevoegd,…

Ten laatste werd al deze kennis samengebracht en werd er gewerkt aan de API.

## Uitwerking

### Opzoekwerk

Allereerst was het zaaks om info op te zoeken over cloud-init, over de eventuele samenwerking met Ansible en over de imlementatie van cloud-init op Hetzner Cloud.

Over cloud-init alleen vond ik zeer veel informatie. Er is een uitgebreide documentatie van alle modules die er zijn en alles dat mogelijk is met cloud-init.

Al vrij snel vond ik dat Hetzner Cloud, cloud-init ondersteunt. Via de optie ‘user-data’ kan je een cloud-config bestand meegeven als je de server aanmaakt. Zo kan je configuraties meegeven aan Hetzner via cloud-init.

Over de samenwerking met Ansible vond ook wel wat info. Cloud-init wordt eigenlijk veel gebruikt in samenwerking met een systeem als Ansible om verder configuraties te doen.

### Lokale testomgeving cloud-init

Daarna was het vooral de zaak om cloud-init wat te leren kennen. Met Ansible was ik al bekend door het vak Linux in het 3de jaar. Via de git repository van <https://github.com/tomkins/cloud-init-vagrant>kon ik lokaal een testomgeving opzetten met cloud-init.

Via de git repo kon ik een iso file aanmaken met in de iso de configuraties van cloud-init. Waardoor ik lokaal een server kon configureren met behulp van cloud-init.

Via vagrant en virtualbox kon ik een **Ubuntu** server opzetten met deze git repo. In deze server moest dan de package *genisoimage* geïnstalleerd worden. Dit deed ik met commando: *apt-get install genisoimage*. In de de repo was er dan een user-data en meta-data bestand. Deze waren de bestanden die ik moest aanpassen. In meta-data kon ik de *local-hostname* en *instance-id* aanpassen. Dit paste ik in het begin vooral aan om te testen of de deze testomgeving werkte. In het user-data bestand kon ik de data van het cloudconfig bestand zetten. De effectieve data om de server mee te configureren

In deze lokale testomgeving was het vooral zaaks om cloud-init wat beter te leren kennen en te bekijken hoe het werkte. De dingen die ik toevoegde zijn hieronder te vinden.

*#cloud-config*

groups:

- ubuntu: [root,sys]

- cloud-users

*# 2 users toegevoegd*

- default

- name: maarten

passwd: $6$Y5N.ote2cyDmb0l0$EizY59wJIt4hMQwyFXIRpJtk.nqldtp5b4VpABWUGwXU.g2XNAHEnE/EIXE8PDZusRNn6m1rCjiY79k3NWioy1

primary\_group: foobar

groups: users

lock\_passwd: false

- name: hans

passwd: $6$3A8NYhbEeeyW$3AzsJVMTpq2vLCI2OD.GdxTth5cwJOTsJE9vWy2T6OklP/IiFJbNrumFvkcukkG2Vdfrd3Ufe.w49aUz8feN91

primary\_group: foobar

groups: users

lock\_passwd: false

*# lock\_passwd true zorgt ervoor dat je niet kan inloggen*

*# een hello world en map aangemaakt*

runcmd:

- [ sh, -xc, "echo $(date) '; hello world!'" ]

- mkdir /home/maarten/testeenmap

*# snelle package om te testen*

packages:

- pwgen

final\_message: "The system is finally up, after $UPTIME seconds"

Via de optie *users* kunnen er gebruikers worden toegevoegd. Ik heb hier de user maarten en hans toegevoegd. Beide hun wachtwoord is geëncrypteerd, maar is ook de naam van respectievelijke gebruiker. Beide zijn ze gewoon lid van de group users en doordat *lock\_passwd* op false staat. Is het mogelijk om in te loggen met deze gebruiker.

Via de optie *runcmd* kan je commando’s laten uitvoeren tijdens het opstarten. Ik deed hier 2 commando’s, een simpele hello world en ook een map dit werd aangemaakt in de home directory van de user maarten.

Via *packages* kunnen er packages worden geïnstalleerd. Ik installeerde gewoon de package *pwgen* aangezien dit gewoon om te testen was.

De laatste optie die ik ook toevoegde was *final\_message.* Dit vond ik zeer handig omdat je de message hiervan altijd onderaan de output of log bestanden vindt.

In */var/log/cloud-init-output.log* kon je de output vinden. Aangezien ik het via vagrant opstartte kon ik de output nooit bekijken toen de server opstartte. Hierin kon ik altijd bekijken of mijn cloudconfig bestand geslaagd was. Heel handig was om dit te doen met commando *tail*.

### Hetzner Cloud testomgeving cloud-init

Na een lokale testomgeving opgezet te hebben moest ik er ook 1 opzetten met Hetzner Cloud. De cloud server provider van Be-Mobile. Voor het begin van deze opdracht heb ik samengezeten met mijn stage mentor om te bekijken hoe hetzner cloud-init ondersteunde. Via mijn opzoekwerk wist ik dat het mogelijk was. Maar de effectieve manier had ik nog niet gevonden.

Allereerst hebben we wel de command line tool van Hetzner geïnstalleerd. Hierdoor kon ik mijn command line servers toevoegen, verwijderen, bekijken,… Hierna heeft mijn mentor ook ene sshkey toegevoegd zodat ik altijd kon inloggen op server die ik maakte. Door het toevoegen van de command line tool zagen de we de dat er bij het creëren van een server de optie *–userdata* kon worden meegegeven. Door het opzoekwerk wist ik dat dit het cloudconfig bestand was.

Het commando dat ik moest runnen om een server aan te maken zag er zo uit.

hcloud server create --name test --image 1 --type 1 --ssh-key maarten --user-data-from-file /d/Desktop/cloud-init/cloudconfig

Via de optie *–name* gaf ik de naam van de server mee. Bij de opties --*image ­*en *--type* gaf ik telkens het id van de gewenste image en servertype. De optie *–ssh-key* was het kiezen van mijn toegevoegde ssh-key, zodat ik kon inloggen op de servers. De laatste optie, *--user-data-from-file,* was het verwijzen naar het cloudconfig bestand.

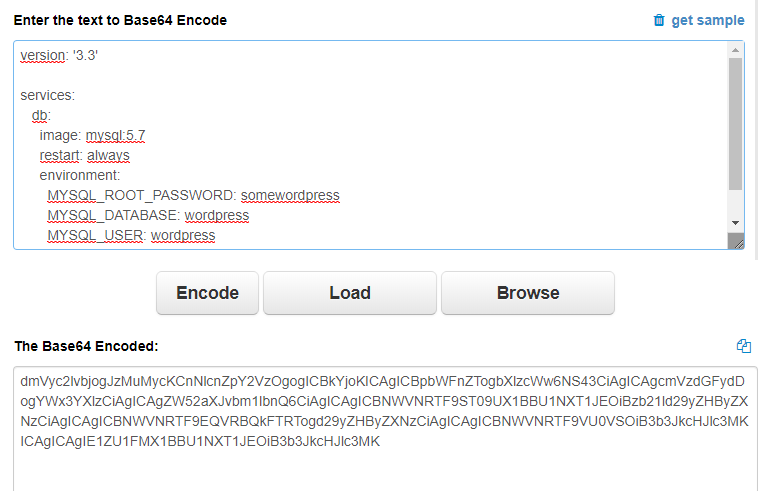
Allereerst testte ik mijn bestaand cloudconfig bestand op server om te bekijken of dit werkte. Alles bleek precies te werken zoals het lokaal werkte. Hierna besloot ik om een 2de al iets “moeilijker” cloud config bestand te maken om testen. Met dit cloud config bestand wou ik docker installeren en configureren. Ik ga bij dit bestand wel alleen de 2 nieuwe delen bespreken.

De eerste nieuwe optie was de optie *package\_upgrade*. Omdat er wordt aangeraden om alles te updaten en upgraden als je docker installeert heb ik dit gedaan. Hier geef je gewoon een boolean true of als aan mee om te kiezen of je dit wilt doen. De default waarde is false.

Hierna koos ik packages die ik wou installeren. Allereerst koos ik de package docker.io om docker te installeren. Na deze package te kiezen startte ik de server ook al is op om te bekijken of alles werkte en docker werd geïnstalleerd. Dit werkte allemaal.

Hierna zocht ik voor een optie om docker te configureren. Allereerst keek ik naar de documentatie van cloud-init of er geen module voor docker configuratie was. Deze vond ik niet, ook op andere sites vond ik hier geen optie voor. Dus koos ik ervoor om dit met docker-compose te doen.

In de documentatie van cloud-init is wel een module om bestanden aan te maken. Dus wou ik met deze optie dit bestand aanmaken. Een fout die ik wel maakte was dat ik de inhoud van het bestand er zo gewoon inzette. Toen ik de server dan opstartte om het bestand te zoeken vond ik het niet, want er was iets fout gegaan. Je moet het een encoded versie van het bestand er in zetten. En dan ook hoe je deze heb encoded. Via de site codebeautify kon ik dit makkelijk doen. Hieronder staat er hoe ik het heb encoded.



Toen ik hierna de server opstartte vond ik wel mijn aangemaakt bestand.

Hierna voegde ik ook de package docker-compose toe aan de server. Ook voegde ik weer de optie *runcmd* toe. Daar gaf ik dan het commando om docker-compose te runnen met de aangemaakte file.

Toen ik de server hierna opstartte werkte alles. Via cloud-init was er een werkende docker server opgestart.

*#cloud-config*

package\_upgrade: true

users:

- default

- name: maarten

passwd: $6$Y5N.ote2cyDmb0l0$EizY59wJIt4hMQwyFXIRpJtk.nqldtp5b4VpABWUGwXU.g2XNAHEnE/EIXE8PDZusRNn6m1rCjiY79k3NWioy1

lock\_passwd: false

packages:

- docker.io

- docker-compose

write\_files:

- path: "home/vagrant/docker-compose.yml"

encoding: "b64"

content: dmVyc2lvbjogJzMuMycKCnNlcnZpY2VzOgogICBkYjoKICAgICBpbWFnZTogbXlzcWw6NS43CiAgICAgcmVzdGFydDogYWx3YXlzCiAgICAgZW52aXJvbm1lbnQ6CiAgICAgICBNWVNRTF9ST09UX1BBU1NXT1JEOiBzb21ld29yZHByZXNzCiAgICAgICBNWVNRTF9EQVRBQkFTRTogd29yZHByZXNzCiAgICAgICBNWVNRTF9VU0VSOiB3b3JkcHJlc3MKICAgICAgIE1ZU1FMX1BBU1NXT1JEOiB3b3JkcHJlc3MK

runcmd:

- sudo docker-compose -f home/vagrant/docker-compose.yml up

final\_message: "The system is finally up, after $UPTIME seconds"

### Hetzner Cloud testomgeving cloud-init & Ansible

De volgende stap was om met cloud-init en Ansible samen te laten werken. Allereest deed ik dit op de manier waarmee ik docker had geconfigureerd. Via de optie *write\_files* maakte ik een klein simpel playbook uit dat ik dan uitvoerde. Het playbook installeert gewoon de package apache.

Ik had ook gezocht of het niet mogelijk was om Ansible op een andere te configureren. Via een aparte Anisble module ofzo. Maar dit is niet mogelijk. Voor **Puppet** en **Chef** waren er wel zo een opties.

Via de optie *packages* voeg ik de package ansible toe. Dan via die optie *runcmd* voer ik het commando uit om de playbook uit te voeren.

*#cloud-config*

packages:

- ansible

write\_files:

- path: "/cloud-init-ansible-playbooks/play.yml"

encoding: "b64"

content: IyBUaGlzIGlzIHRoZSBzc2ggY2xpZW50IHN5c3RlbS13aWRlIGNvbmZpZ3VyYXRpb24gZmlsZS4gIFNlZQojIHNzaF9jb2

runcmd:

- ansible-playbook -i, /cloud-init-ansible-playbooks/play.yml

De volgende stap om dit te doen is iets moeilijker. Aangezien het wat omslachtig is om elke keer ene playbook toe te voegen via de optie *write\_files .* Zei de stagementor dat het misschien beter is om dit via een git clone te doen, en zo de playbook binnen te halen. Hij maakte voor mij dan een account aan op Bitbucket dat toegang had tot een repository en gaf me ook de private key mee van het repo. Zodat ik via ssh de playbook kon binnen halen.

Nog steeds was er ansible nodig qua package. Maar nu was er ook git nodig, normaal gezien installeert cloud-init dit default op een machine. Maar ik heb deze er toch maar bijgezet voor de zekerheid.

Het volgende wat ik deed was de private key toe voegen. Ik deed via de optie *ssh\_keys* en daar de optie *rsa\_private.* Ik kopieerde gewoon de private key daarin.

Bij de optie *runcmd* deed ik daarna ene git clone van de repo en daarna voerde ik mijn playbook uit. Ook deed ik een ssh-keyscan op bitbucket. Zodat er geen error kwam unkown host. Alleen werkte dit niet.

Er was een fout bij het binnenhalen van de repository. Als ik inlogde op de server en dit deed kreeg ik de error “acces denied”. Wat ik raar vond aangezien ik mijn private key had toegevoegd. Maar via stackoverflow vond ik een oplossing. Ik moest aan mijn *ssh\_config* bestand toevoegen dat bitbucket deze privatekey moest gebruiken. Eerste testte ik dit zonder cloud-init dus gewoon handmatig. Dit bleek te werken. Daarna zocht ik voor een manier om dit een cloud-init te verwerken. Spijtig genoeg was er geen extra optie bij de optie *ssh\_keys* om dit de doen. Ik besloot dus om weer met de optie *write\_files* te werken. Ik kopieerde het bestand dit ik handmatig had aangemaakt en maakte dit weer aan met de methode *write\_files*. Ik schreef dit bestand dan weg op de plaats waar het hoor te staan om het oude te overschrijven.

Als ik de server dan opstartte werkte alles. Hieronder is mijn cloudconfig bestand. De private key is niet zichtbaar omdat dit veel plaats in nam. Ook omdat het misschien niet zo veilig is om deze te delen.

*#cloud-config*

packages:

- ansible

- git

*#git normaal al default op machine met cloud-init maar better be safe*

ssh\_keys:

rsa\_private: |

-----BEGIN RSA PRIVATE KEY-----

DATA PRIVATE KEY

-----END RSA PRIVATE KEY-----

write\_files:

- path: "/etc/ssh/ssh\_config"

encoding: "b64"

content: IyBUaGlzIGlzIHRoZSBzc2ggY2xpZW50IHN5c3RlbS13aWRlIGNvbmZpZ3VyYXRpb24gZmlsZS4gIFNlZQojIHNzaF9jb25maWcoNSkgZm9yIG1vcmUgaW5mb3JtYXRpb24uICBUaGlzIGZpbGUgcHJvdmlkZXMgZGVmYXVsdHMgZm9yCiMgdXNlcnMsIGFuZCB0aGUgdmFsdWVzIGNhbiBiZSBjaGFuZ2VkIGluIHBlci11c2VyIG

runcmd:

- ssh-keyscan bitbucket.org >> /etc/ssh/ssh\_known\_hosts

- git clone git@bitbucket.org:be-mobile/cloud-init-ansible-playbooks.git

- ansible-playbook -i, /cloud-init-ansible-playbooks/play.yml

### Kubernetes

Hierna was het werken met cloud-init even gedaan. Voor de volgende stappen van de opdracht moest ik een basiskennis hebben van kubernetes. Aangezien ik deze niet had was zaaks om dit wat beter te leren kennen. Ik kreeg van mijn stagementor een simpele opdracht in kubernetes. Een cluster opzetten met 3 master nodes en 1 worker node dit allemaal met kubernetes 1.12.

Het aanmaken van servers gebeurde nog steeds in Hetzner Cloud alleen waren er nu meer requirements. De *image* moest “Ubuntu-18.04” zijn en het server-type “cx21”. Daardoor zag het commando er nu iets anders uit.

hcloud server create --name master1 --image ubuntu-18.04 --type 3 --ssh-key maarten

De installatie van de packages en instellingen moest eerst handmatig. Wat later kon worden geautomatiseerd, moest worden geautomatiseerd in cloud-init. Mijn stagementor gaf me een handige handleiding om te volgen om kubernetes te installeren. Je moest een paar commando’s doen om dit te installeren. Allereerst moest je de packages *kubadm*, *kubelet* en *kubectl* binnenhalen via een andere package list met een paar commando’s. Daarna kon je de packages installeren. Alleen moest er ook de versie meegegeven worden aangezien het versie 1.12 moest zijn. Ook bleek docker aanwezig te moeten zijn, meer bepaald de versie 18.06. Dit installeerde ik dus ook.

apt-get update && apt-get install -y apt-transport-https curl

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 https://apt.kubernetes.io/ kubernetes-xenial main

EOF

apt-get update

apt-get install -y kubelet=1.12.7-00 kubeadm=1.12.7-00 kubectl=1.12.7-00

apt-mark hold kubelet kubeadm kubectl

Hierna kun het initialiseren van het cluster beginnen. Hier had ik ook een handleiding voor. Allereerst begon ik met het aanmaken van 1 master node. Voor het aanmaken hiervan moest ik een *kubeadm-config.yaml* aanmaken. Dit is het config bestand voor kubernetes. Bij de opties CP0\_hostname en CP0\_IP, moest je respectievelijk de hostname en het ip van je server invullen.

**apiVersion: kubeadm.k8s.io/v1alpha3**

**kind: ClusterConfiguration**

**kubernetesVersion: v1.12.7**

**etcd:**

**local:**

**extraArgs:**

**name: "CP0\_HOSTNAME"**

**listen-client-urls: "https://127.0.0.1:2379,https://CP0\_IP:2379"**

**advertise-client-urls: "https://CP0\_IP:2379"**

**listen-peer-urls: "https://CP0\_IP:2380"**

**initial-advertise-peer-urls: "https://CP0\_IP:2380"**

**initial-cluster: "CP0\_HOSTNAME=https://CP0\_IP:2380"**

**serverCertSANs:**

**- CP0\_HOSTNAME**

**- CP0\_IP**

**peerCertSANs:**

**- CP0\_HOSTNAME**

**- CP0\_IP**

**networking:**

**# This CIDR is a Calico default. Substitute or remove for your CNI provider.**

**podSubnet: "192.168.0.0/16"**

Hierna moest het commando uitgevoerd worden om een cluster aan te maken met deze master node.

kubeadm init --config kubeadm-config.yaml

Hierna deed ik deze commando’s. Het eerste is zodat ik met mijn user root alles kan bekijken en dingen kan uitvoeren op dit cluster. De twee laatste zijn om de netwerk instellingen aan te passen.

export KUBECONFIG=/etc/kubernetes/admin.conf

kubectl apply -f https://docs.projectcalico.org/v3.3/getting-started/kubernetes/installation/hosted/rbac-kdd.yaml

kubectl apply -f https://docs.projectcalico.org/v3.3/getting-started/kubernetes/installation/hosted/kubernetes-datastore/calico-networking/1.7/calico.yaml

Nadat dit gedaan was moest ik bepaalde certificaten en configuratie bestanden kopiëren naar mijn andere toekomstige master nodes. Dit kon ik doen via een scriptje dat ze meegaven

USER=root *# customizable*

CONTROL\_PLANE\_IPS="IP\_MASTER2 IP\_MASTER3"

for host in ${CONTROL\_PLANE\_IPS}; do

scp /etc/kubernetes/pki/ca.crt "${USER}"@$host:

scp /etc/kubernetes/pki/ca.key "${USER}"@$host:

scp /etc/kubernetes/pki/sa.key "${USER}"@$host:

scp /etc/kubernetes/pki/sa.pub "${USER}"@$host:

scp /etc/kubernetes/pki/front-proxy-ca.crt "${USER}"@$host:

scp /etc/kubernetes/pki/front-proxy-ca.key "${USER}"@$host:

scp /etc/kubernetes/pki/etcd/ca.crt "${USER}"@$host:etcd-ca.crt

scp /etc/kubernetes/pki/etcd/ca.key "${USER}"@$host:etcd-ca.key

scp /etc/kubernetes/admin.conf "${USER}"@$host:

done

Hierna begon de configuratie van de andere master nodes. Allereerst de 2de master node ook hierbij kreeg ik een template van een kubeadam-config bestand. Bij de opties CP0\_hostname, CP0\_IP, CP1\_hostname en CP1\_IP, moest je respectievelijk de hostname en het ip van je server invullen

apiVersion: kubeadm.k8s.io/v1alpha3

kind: ClusterConfiguration

kubernetesVersion: v1.12.7

etcd:

local:

extraArgs:

name: "CP1\_HOSTNAME"

listen-client-urls: "https://127.0.0.1:2379,https://CP1\_IP:2379"

advertise-client-urls: "https://CP1\_IP:2379"

listen-peer-urls: "https://CP1\_IP:2380"

initial-advertise-peer-urls: "https://CP1\_IP:2380"

initial-cluster: "CP0\_HOSTNAME=https://CP0\_IP:2380,CP1\_HOSTNAME=https://CP1\_IP:2380"

initial-cluster-state: existing

serverCertSANs:

- CP1\_HOSTNAME

- CP1\_IP

peerCertSANs:

- CP1\_HOSTNAME

- CP1\_IP

networking:

# This CIDR is a calico default. Substitute or remove for your CNI provider.

podSubnet: "192.168.0.0/16"

Een fout die ik maakt was dat ik na deze stap altijd weer *kubeadm init --config kubeadm-config.yaml* deed met dit config bestand op de tweede server. Dit gaf altijd een fout en ik kon er maar niet bijkomen waarom dit was. Met deze fout heb ik toch een paar dagen mee bezig gezeten. Uiteindelijk bleek de oplossing zo stom te zijn dat het een heel domme fout ook bleek te zijn. Blijkbaar moet je dit dit commando niet meer uitvoeren op de andere master nodes. Het stond ook nergens aangegeven maar omdat sommige andere dingen die je niet moest doen ook nergens waren aangegeven ging ik hier vanuit. Maar toen ik dan doorging met de stappen die ik moest doen ging alles zoals het moest doen.

De effectieve stappen die ik moest doen. Allereerst moest ik de bestanden die waren gekopieerd van master 1 naar de goeie plaats zetten. Gelukkig was hier ook een handig scriptje voor.

USER=root *# customizable*

mkdir -p /etc/kubernetes/pki/etcd

mv /home/${USER}/ca.crt /etc/kubernetes/pki/

mv /home/${USER}/ca.key /etc/kubernetes/pki/

mv /home/${USER}/sa.pub /etc/kubernetes/pki/

mv /home/${USER}/sa.key /etc/kubernetes/pki/

mv /home/${USER}/front-proxy-ca.crt /etc/kubernetes/pki/

mv /home/${USER}/front-proxy-ca.key /etc/kubernetes/pki/

mv /home/${USER}/etcd-ca.crt /etc/kubernetes/pki/etcd/ca.crt

mv /home/${USER}/etcd-ca.key /etc/kubernetes/pki/etcd/ca.key

mv /home/${USER}/admin.conf /etc/kubernetes/admin.conf

Daarna moesten deze commando’s worden uitgevoerd om de kubelet te bootstrappen.

kubeadm alpha phase certs all --config kubeadm-config.yaml

kubeadm alpha phase kubelet config write-to-disk --config kubeadm-config.yaml

kubeadm alpha phase kubelet write-env-file --config kubeadm-config.yaml

kubeadm alpha phase kubeconfig kubelet --config kubeadm-config.yaml

systemctl start kubelet

Deze commando’s werden uitgevoerd om de node toe te voegen aan het cluster.

export CP0\_IP=IP 1

export CP0\_HOSTNAME=NAAM 1

export CP1\_IP=IP 2

export CP1\_HOSTNAME=NAAM 2

kubeadm alpha phase etcd local --config kubeadm-config.yaml

export KUBECONFIG=/etc/kubernetes/admin.conf

kubectl exec -n kube-system etcd-${CP0\_HOSTNAME} -- etcdctl --ca-file /etc/kubernetes/pki/etcd/ca.crt --cert-file /etc/kubernetes/pki/etcd/peer.crt --key-file /etc/kubernetes/pki/etcd/peer.key --endpoints=https://${CP0\_IP}:2379 member add ${CP1\_HOSTNAME} https://${CP1\_IP}:2380

Dan werden deze commando’s nog uitgevoerd om de node te markeren als een master.

kubeadm alpha phase kubeconfig all --config kubeadm-config.yaml

kubeadm alpha phase controlplane all --config kubeadm-config.yaml

kubeadm alpha phase kubelet config annotate-cri --config kubeadm-config.yaml

kubeadm alpha phase mark-master --config kubeadm-config.yaml

Dan werd identiek hetzelfde gedaan voor de 3de master node. Het enige andere was het kubeadm-config bestand. Bij de opties CP0\_hostname, CP0\_IP, CP1\_hostname, CP1\_IP, CP0\_hostname en CP0\_IP, moest je respectievelijk de hostname en het ip van je server invullen

apiVersion: kubeadm.k8s.io/v1alpha3

kind: ClusterConfiguration

kubernetesVersion: v1.12.7

etcd:

local:

extraArgs:

name: "CP2\_HOSTNAME"

listen-client-urls: "https://127.0.0.1:2379,https://CP2\_IP:2379"

advertise-client-urls: "https://CP2\_IP:2379"

listen-peer-urls: "https://CP2\_IP:2380"

initial-advertise-peer-urls: "https://CP2\_IP:2380"

initial-cluster: "CP0\_HOSTNAME=https://CP0\_IP:2380,CP1\_HOSTNAME=https://CP1\_IP:2380,CP2\_HOSTNAME=https://CP2\_IP:2380"

initial-cluster-state: existing

serverCertSANs:

- CP2\_HOSTNAME

- CP2\_IP

peerCertSANs:

- CP2\_HOSTNAME

- CP2\_IP

networking:

# This CIDR is a calico default. Substitute or remove for your CNI provider.

podSubnet: "192.168.0.0/16"

Het toevoegen van de worker aan een cluster bleek het makkelijkste van allemaal. Op het einde van het opstarten van de eerste master kreeg je een join commando. Dit moest je gewoon uitvoeren op de server die een worker moest zijn.

export KUBECONFIG=/etc/kubernetes/admin.conf

kubeadm join 159.69.18.139:6443 --token 1uv9vp.n29a064d7msrdj56 --discovery-token-ca-cert-hash sha256:38b03519403ad2b74c4a8e71be71a227b2bb786d8cd4fff24d781df49122aa1a

Na handmatig een kubernetes cluster te hebben opgezet was het nu zaaks om sommige zaken te laten automatiseren. Het installeren van alle packages bijvoorbeeld was het eerste. Hier schreef ik een simpel bash scriptje voor dat ik alle pre requirements uitvoerde. Dit scriptje werkte perfect en deed wat het moest doen.

#!/bin/sh

apt-get update \

&& apt-get install -qy docker.io=18.06.\*

apt-get update && apt-get install -y apt-transport-https curl

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 https://apt.kubernetes.io/ kubernetes-xenial main

EOF

apt-get update \

&& apt-get install -y kubelet=1.12.7-00 kubeadm=1.12.7-00 kubectl=1.12.7-00

apt-mark hold kubelet kubeadm kubectl

systemctl enable docker.service

Hierna probeerde ik dit scriptje wat te “kopiëren” naar cloud-init. Maar ik stuitte op een fout. Doordat de kubernetes packages moeten worden binnengehaald gaf het altijd fouten. Uiteindelijk omdat dit niet noodzakelijke was voor de stage mentor (aangezien zij dit toch al doen via Ansible en ik dit later ook zou moet doen) ben ik hier niet verder meegegaan.

Ook voor het toevoegen van een andere master node aan een cluster geld wat hetzelfde. Ik probeerde dit ook te automatiseren maar de stagementor zei me dat het echt noodzakelijke, waar ik aan verder ging werken hier was. Dat een worker makkelijk kon worden toegevoegd worden aan een cluster. En dit was vrij eenvoudig te implementeren. Gewoon bij het opstarten van de server alle packages installeren en ten laatste het join commando uitvoeren. Hetgeen ik met *write\_files* aanmaak is het script van hierboven.

*#cloud-config*

*# start script om kubernetes en extra te installeren*

write\_files:

- path: "/home/maarten/start.sh"

encoding: "b64"

content: 

runcmd:

- sh /home/maarten/start.sh

- export KUBECONFIG=/etc/kubernetes/admin.conf

- kubeadm join 159.69.18.139:6443 --token 1uv9vp.n29a064d7msrdj56 --discovery-token-ca-cert-hash sha256:38b03519403ad2b74c4a8e71be71a227b2bb786d8cd4fff24d781df49122aa1a

Iets waar ik nu nog niet moest aanwerken maar voor API ging moeten doen. Ervoor zorgen dat het aanvragen van een token per worker gebeurt, zodat elke worker een nieuwe token krijgt. En dat deze token dan verwijdert wordt.

### Golang

De laatste stap voor ik kon werken aan de API was het leren kennen van de programmeertaal Golang. In het bijzonder het programmeren in Golang met de package *hcloud(de package om hetzner cloud servers te beheren en configureren)*. Ik besloot om 3 soorten programma’s te maken: het aanmaken van 1 of meerdere servers, het verwijderen van 1 of meerdere servers en bekijken hoeveel servers online zijn.

Hieronder is een voorbeeld van het programma dat ik geschreven heb om servers aan te maken. Omdat de code van een programma veel plaats inneemt, geef ik maar 1 voorbeeld.

package main

import (

    "context"

    "fmt"

    "io/ioutil"

    "log"

    "net"

    "github.com/hetznercloud/hcloud-go/hcloud"

)

func makenServer(naam, token string) (int, net.IP, string) {

*//variabelen*

    var idServer int

    var ipAdres net.IP

    oke, \_ := ioutil.ReadFile("cloudconfig")

    uhu := string(oke)

*// token geven aan client*

    client := hcloud.NewClient(hcloud.WithToken(token))

*// credentials hcloud server*

    opts := hcloud.ServerCreateOpts{

        Name: naam,

        ServerType: &hcloud.ServerType{ID: 3},

        Image: &hcloud.Image{ID: 168855},

        SSHKeys: []\*hcloud.SSHKey{

            {ID: 432218},

        },

        UserData: uhu,

    }

*// server aanmaken*

    result, \_, error := client.Server.Create(context.TODO(), opts)

*// checken naam*

    for error != nil {

        fmt.Println("Naam bestaal al kies een nieuwe")

        fmt.Scanln(&naam)

        optse := hcloud.ServerCreateOpts{

            Name: naam,

            ServerType: &hcloud.ServerType{ID: 1},

            Image: &hcloud.Image{ID: 2},

            SSHKeys: []\*hcloud.SSHKey{

                {ID: 432218},

            },

        }

        result2, \_, err := client.Server.Create(context.Background(), optse)

        error = err

        result = result2

    }

*// info van server returnens*

    idServer = result.Server.ID

    ipAdres = result.Server.PublicNet.IPv4.IP

    return idServer, ipAdres, naam

}

*// aantal servers*

func aantalServers(token string) int {

    opts := hcloud.ServerListOpts{}

    client := hcloud.NewClient(hcloud.WithToken(token))

    servers, \_, \_ := client.Server.List(context.Background(), opts)

    return len(servers)

}

func main() {

    var token string = "h7cwcYE0ZP6QiuymDr1GoNwpwMXYH74SNPRHBsFg2ZnMIg9WaVSU1OZBwPPX4MRc"

*// geef naam*

    fmt.Print("Geef naam van server: ")

    naam := ""

    fmt.Scanln(&naam)

*//maak server*

    id, ip, naam := makenServer(naam, token)

*// return de info van de server*

    fmt.Println("----- Het ID van de server: ----- ", id)

    fmt.Println("----- De naam van de server: ----- ", naam)

    fmt.Println("----- Het IP Adress van de Server: ----- ", ip)

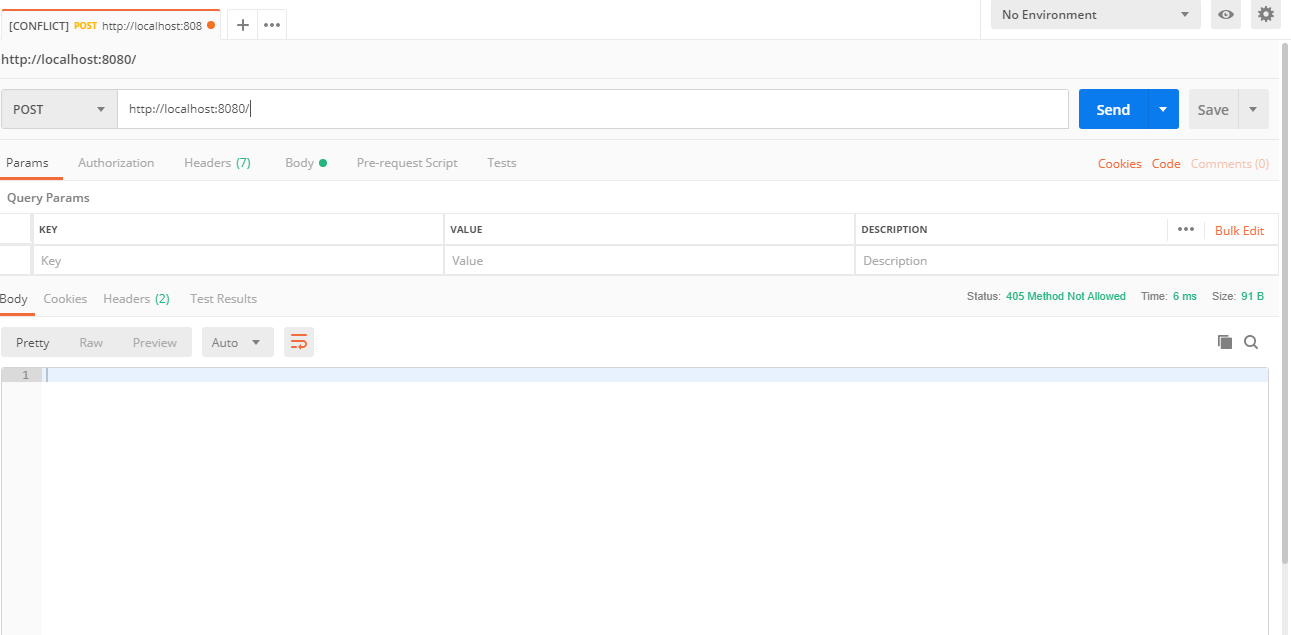
    log.Println("")

    fmt.Println("----- Het aantal servers online momenteel: ----- ", aantalServers(token), "servers")

}

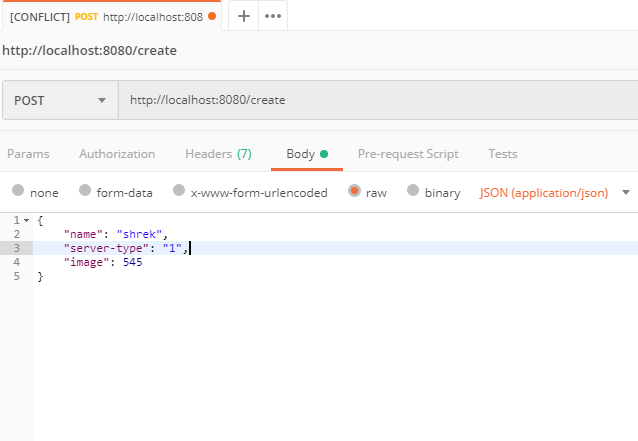
### API - Demo

De volgende stap voor de opdracht was de laatste en de belangrijkste. Nu was het zaaks alle kennis die ik had verworven de voorbije weken bij een te bundelen en samen te voegen in 1 programma/API. Het programma maakt een server aan via HetznerCloud met configuraties van Ansible en cloud-init en voegt deze toe aan een Kubernetes cluster. Het programma is geschreven in Golang. Als de Api runt kan er een http request worden gedaan om een server aan te maken of een server te verwijderen. Die http requests kunnen gebeuren in een webbrowser. Maar mijn stagementor gaf me een handige tool om dit te doen. Met de tool krijg je meer informatie en ook foutboodschappen als de Api faalt, de tool is Postman.



#### Aanmaken en verwijderen

Eerste dat ik moet doen is een functie in de API steken om een server aan te maken en te verwijderen. Dit was niet zo veel werk. Doordat ik in Golang al een programma had geschreven om een server aan te maken en te verwijderen. Ik kon daardoor veel van deze code kopiëren naar de API. Een moeilijkheid was om overal de errors te catchen. Ook moest er worden toegevoegd worden om input waardes toe te voegen met bijvoorbeeld: de naam, server-type en de image. De waardes werden toegevoegd aan de API via een JSON formaat.



Eerste stap was dat het programma zijn paramaters inlas. Met deze paramaters moest de server worden aangemaakt. Voor het aanmaken van de server maakte ik een aparte functie aan *CreateServer*, deze roepte ik dan aan. Een moeilijkheid was om cloud-init configuraties mee te geven als de servers werden aangemaakt. Dit kon meegeven worden als een waarde. De waarde werd toegevoegd door het bestand met de configuraties in te lezen via Golang en dit om te zetten naar een string.

Het verwijderen van een server was niet zo uitgebreid en moeilijk. Het deel voor de naam in te geven voor welke server er moet verwijderd worden kon worden gekopieerd van de functie voor het aanmaken van servers. Het effectieve verwijderen van een server had ik al eens geschreven in mijn test programma’s en kon ik daaruit kopiëren.

#### Kubernetes & Templates

Volgende stap voor de API was een server tijdens het aanmaken, toevoegen aan een Kubernetes cluster. Ook dat als een server verwijderd deze ook werd verwijderd uit het cluster.

Allereerst maakte ik een aparte functie *GetKubeJoinCommand*. Via deze functie vroeg ik een join command aan, aan de Kubernetes master. Het ip van de Kubernetes moest wel gekend zijn. Er werd dus een extra paramater toegevoegd aan de functie om in te vullen, namelijk: masterip. Er werd dan een ssh sessie gestart op de Kubernetes master via Golang. Daaruit kon ik het commando uitvoeren om een token aan te vragen. De output kon ik wegschrijven naar een string. Ook heb ik voor security maar een levensduur van 15 minuten aan de token toegevoegd.

Nadat het join commando verkregen was, moest dit worden toegevoegd worden aan het cloud-init script. Dit kon worden gedaan doormiddel van Templates. Via Golang is het mogelijk in een bestand of constante een waarde in te vullen waar dit is aangeduid Het cloud-init script dat ik al inlas om toe te voegen aan de server, las ik nu wat vroeger in. Zo kon ik het join commando er eerst aan toevoegen. Ik voegde dit als laatste commando toe onder de module *runcdm*. Via de accolades kon ik verwijzen waar het moest staan, hieronder een voorbeeld. Een environment variabel moest ik worden geëxporteerd om het commando uit te voeren.

runcmd:

- export KUBECONFIG=/etc/kubernetes/admin.conf && {{.Kubetoken}}

Na het werk van Golang en het invullen van het template krijg je dit.

runcmd:

- export KUBECONFIG=/etc/kubernetes/admin.conf && kubeadm join 116.203.159.186:6443 --token u2juj1.a021ht7gh6ju01v0 --discovery-token-ca-cert-hash sha256:029ce81b337983c25056df14677b4b31e4bf1decd27d0181e4fc9f48472fda71

Nu als de server werd aangemaakt was die ook lid van het Kubernetes cluster.

Het verwijderen van de server was niet zo moeilijk. Ook hier moest wel eerst een connectie gelegd worden met de master server. Dus ook hier moest dit ip worden meegegeven.

Voor het verwijderen van de server werd eerst een ssh connectie gelegd met master. Op de master werd het commando uitgevoerd om de server van het cluster te verwijderen. Meer moest er niet worden toegevoegd.

#### Informatie verkrijgen via API

De stagementor had een opmerking over de API. Het server-type en image-type kon worden meegeven maar er was geen optie om deze te bekijken via de API. Dus heb ik ook 2 kleine functies toegevoegd die een json weergeeft met alle images of server-types.

Ook heb ik een kleine help functie geschreven. Als je deze aanroept krijg je alle opties die mogelijk zijn met de API. Ook kan je hier bekijken welke paramaters je bij bepaalde requests moet ingeven.

#### Ansible

De volgende stap was de implementatie van Ansible voor het aanmaken van de servers. De eerste configuraties worden nog steeds gedaan door cloud-init. Het playbook wordt ook aangeroepen en uitgevoerd door cloud-init.

Het eerste dat werd gedaan/getest is het uitvoeren van een playbook. Zo een cloud-init script had ik al eens gemaakt. Dus gebruikte ik dit eerst gewoon. Dit werkte zoals verwacht werd maar er moesten nog wat aanpassing gebeuren. Het playbook werd gemaakt via de module *write\_files* en daar stand gewoon zo de b64 versie van het playbook. Maar via Golang kunnen teksten ook worden gecodeerd. Dus heb ik dit hierin geïmplementeerd. Het playbook werd ingelezen door Golang en gecodeerd. Via templating vulde ik de content dan in.

Het volgende was test roles pullen vanuit git en het playbook zo uitvoeren. Een moeilijkheid hierbij was dat als ze werden opgeroepen. Ansible moet weten dat dit roles zijn. Een oplossing hiervoor was ansible-galaxy. Via ansible-galaxy kunnen er rollen worden binnengehaald op het systeem en worde, deze ook in de juiste map gezet. Het is mogelijk om dit met 1 commando te doen om ze allemaal binnen te halen maar dit is wat te omslachtig. De mogelijkheid waarvoor ik heb gehanteerd is werken met een requirements bestand. Dit is een yaml bestand waar je de rollen aan toevoegt die moeten worden toegevoegd aan de server. Hier is een voorbeeld waar 1 rol mee wordt toegevoegd.

- src: git+git@bitbucket.org:be-mobile/ansible-role-bemobile.prep\_docker.git

version: master

scm: git

name: bemobile-docker

Een probleem dat wel opdook was dat de rollen hierna niet konden worden geüpdatet. Dit moest dus worden opgelost. Want als het playbook na aan het aanmaken van de servers weer werd gedraaid moest dit wel met de geüpdatete rollen gebeuren. Een probleem dat gevonden werd met ansible-galaxy hiermee is, er is geen optie om een update uit te voeren. Een optie zoals bij Github, een pull, bestaat hier niet. Dus moest er gezocht worden naar andere opties. Een optie was om de optie *--force*  toe te voegen aan het commando dat je draait om de rollen binnen te halen. De rollen worden dan altijd binnengehaald en overschreven. Zo is er altijd de nieuwe versie aanwezig. Een nadeel is wel dat dit even kan duren naargelang de hoeveelheid rollen. Want hij overschrijft alles. Andere opties heb ik niet gevonden dus werd er met de --*force* optie gewerkt.

Volgend probleem dat daarop volgde was hoe dit automatisch kon worden uitgevoerd. Er moest een optie worden toegevoegd zodat elke keer dat het playbook werd uitgevoerd dit gebeurde met de geüpdatete rollen. Meteen dacht ik aan de *pre\_tasks* module in Ansible, dit voert taken uit helemaal in het begin van het playbook. Daar werd gezet dat het commando dan moest worden uitgevoerd. Dit gebeurde allemaal maar er was nog een probleem. Ansible gebruikte niet de geüpdatete rol maar de oude rol. Ansible zoekt in de achtergrond eerst naar de rollen en voert dan pas de taken uit. Dit is uiteindelijk opgelost door i.p.v. *roles*, de module *tasks* te gebruikten. Daar werd dan de module *include\_roles* gezet. Deze gebruikte wel de geüpdatete rol.

#### Development test via Jenkins.

Als dat allemaal gedaan was kon de API voor het eerst worden online getest, wel nog in development. Mijn stagementor voegde de API toe aan Jenkins.

Allereest moest ik alle errors die Jenkins gaf eruit halen. Eerst en vooral moesten alle packages via vendor worden meegegeven in de API. Vendor zorgt ervoor dat de packages in de map van het Golang programma zitten. Zo geeft de API geen errors dat hij de packages niet vindt. Via documentatie van Be-Mobile kon ik dit vrij makkelijk doen.

Als die fouten eruit waren gaf hij andere errors. Elke error dat kon gebeuren moest er uit worden gecatcht. Dit was een vrij makkelijke taak omdat jenkins een link gaf met elke lijn waar een error kon gebeuren. Maar het was wel een taak die aardig wat tijd in beslag nam.

Eens alle errors eruit waren gehaald was het programma voor de eerste keer online. Om het op te zetten moest ik dit laten draaien via een docker container. Dit was het commando op de container met het commando te laten draaien:

**docker run -p** 8080:8080 **-e** CONFIG\_HETZNERTOKEN="h7cwcYE0ZP6QiuymDr1GoNwpwMXYH74SNPRHBsFg2ZnMIg9WaVSU1OZBwPPX4MRc" **-e** CONFIG\_DEFAULTIMAGEID=168855 **-e** CONFIG\_DEFAULTSERVERTYPEID=3 **-e** CONFIG\_PRIVATEKEYPATH= C:\Users\Maarten\.ssh\id\_rsa docker.be-mobile.biz:5000/kubernetes-worker-manager-api:0.0.1-develop-16

De waarden bij de optie *-e* zijn de config waarden. Dit zijn de standaard waarden van het programma. Er zijn er 4. De *hetznertoken* dit is de token waarmee je de server aanmaakt op HetznerCloud. De *DefaultImageID* dit het standaard id van de image. Als er geen image id wordt meegegeven bij de requests wordt dat id gebruikt, hetzelfde voor *ServerTypeID*. En het *PrivateKeyPath* dit is het pad naar de private sleutel die moet worden gebruikt bij de ssh verbindingen in het programma.

Toen ik een request uitvoerde kreeg ik wel nog een fout. Hij vond de private sleutel niet op de server. De sleutel moest dus worden gekopieerd naar de server of er moest een link worden gelegd. Dit heb gedaan door deze regel toe te voegen aan het commando ***-v*** *C:\Users\Maarten\.ssh\id\_rsa:/be-mobile/ssh-keys/key.pem.* Ook verander ik het *PrivateKeyPath* daarna naar */be-mobile/ssh-keys/key.pem.*

Als ik hierna een request uitvoerde werkte het ook online.

#### Vault

Voor de stap erna heb ik samengesproken met mijn stagementor en de rest van het devops team. We hebben toen besloten om ook Vault te integreren in de API. Via Vault kan je op een veilige manier secrets binnenhalen die je moet gebruiken. In de API moest het worden geïntegreerd in Ansible. Omdat de data op de Vault ook heel gevoelig is moest de verbindig worden gemaakt via een bastion server.

Vault werd gebruikt in 2 stappen.

#### API demo visueel ontwerp

Mss dit vanboven zetten?

#### 

### API deployment

Nadat de API demo werkt was het zaaks ze klaar te maken voor productie.

## Eindresultaat

Nog aan bezig (nog geen resultaat)

### Business doelstellingen

Nog aan bezig( nog geen resultaat)

### Persoonlijke doelstellingen

Nog aan bezig( nog geen resultaat)

# Eindreflectie

Nog bezig aan stage.

# Bibliografie

Cloud-init docs: <https://cloudinit.readthedocs.io/en/latest/topics/examples.html#disk-setup>

Cloud-init IBM informatie: <https://www.ibm.com/support/knowledgecenter/en/SSB27U_6.4.0/com.ibm.zvm.v640.hcpo5/instsubuntu.htm>

Hetzner Cloud & cloud-init: <https://wiki.hetzner.de/index.php/CloudServer/en>

Cloud-init & Ansible: <https://www.cloudsigma.com/an-introduction-to-server-provisioning-with-cloudinit/>

Git repo lokaal cloud-init: <https://github.com/tomkins/cloud-init-vagrant>

Codebeautify - Enoding bestanden: <https://codebeautify.org/base64-encode>

Stackoverflow - Probleem private key: <https://stackoverflow.com/questions/4565700/how-to-specify-the-private-ssh-key-to-use-when-executing-shell-command-on-git>

Installing kubernetes: <https://v1-12.docs.kubernetes.io/docs/setup/independent/install-kubeadm/>

Creating cluster: <https://v1-12.docs.kubernetes.io/docs/setup/independent/high-availability/>

# Verklarende woordenlijst

Devops

Ansible

Cloud-init

Linux

Kubernetes

Golang

Api

Ubuntu

Puppet

Chef

BitBucket

Json

playbook

Yaml

Jenkins

Vault

Bastion

# Bijlage: stagedagboek

<http://tinfbo1.hogent.be/Activiteitsverslag/MijnActiviteitsverslagen>