



Faculteit Bedrijf en Organisatie

Onderzoek naar de educatieve mogelijkheden om chaos engineering tools en experimenten toe te passen
om de weerbaarheid te testen van een Kubernetes cluster

Ken Bruggeman

Scriptie voorgedragen tot het bekomen van de graad van
professionele bachelor in de toegepaste informatica

Promotor:
Gertjan Bosteels
Co-promotor:
Bert Van Vreckem

Instelling: Hogeschool Gent

Academiejaar: 2021-2022

Tweede examenperiode

Faculteit Bedrijf en Organisatie

Onderzoek naar de educatieve mogelijkheden om chaos engineering tools en experimenten toe te passen
om de weerbaarheid te testen van een Kubernetes cluster

Ken Bruggeman

Scriptie voorgedragen tot het bekomen van de graad van
professionele bachelor in de toegepaste informatica

Promotor:
Gertjan Bosteels
Co-promotor:
Bert Van Vreckem

Instelling: Hogeschool Gent

Academiejaar: 2021-2022

Tweede examenperiode

Woord vooraf

Deze bachelorproef is het sluitstuk van de opleiding Bachelor in de Toegepaste Informatica aan de Hogeschool Gent en is gericht op de specialisatie Systeem- en Netwerkbeheer. De inspiratie voor Chaos Engineering te onderzoeken vond ik in het verhaal van Netflix die doelbewust hun applicatie en systemen ging aanvallen om te leren hoe zij vervolgens verbeteringen konden aanbrengen om de weerbaarheid te verhogen. Tijdens mijn opleiding leerde ik alreeds hoe moderne applicaties via de microservices softwarearchitectuur worden opgesplitst in meerdere containers en hoe deze applicaties via een CI/CD pipeline kunnen getest worden om alreeds vroegtijdig fouten op te sporen. Hoe deze applicaties kunnen groeien via containerorkestratie zoals Kubernetes en de uitdagingen die dit met zich meebrengt was dan ook een logisch vervolg in mijn leertraject. Door zowel Kubernetes als Chaos Engineering te onderzoeken leerde ik meer bij over de interne werking van Kubernetes en hoe deze acties onderneemt wanneer applicaties of onderdelen van de cluster getroffen worden.

Gedurende dit onderzoek kon ik meermaals rekenen op hulp en advies van mijn promotor Gertjan Bosteels en co-promotor Bert Van Vreckem. Mijn oprechte dank gaat uit naar hen voor alle ondersteuning en tijd die zij in mij geïnvesteerd hebben om dit onderzoek tot een goed eind te brengen.

Verder wil ik ook mijn oneindige dank betuigen aan mijn vrouw, familie en vrienden die me ondersteund hebben en het mogelijk maakten de opleiding Bachelor in de Toegepaste Informatica als afstandsstudent succesvol te combineren met een vaste job en een gezinsleven met twee jonge kinderen. Zonder hun onvoorwaardelijke steun was ik nooit tot dit punt in de opleiding kunnen raken.

Samenvatting

Kubernetes is een bekende vorm van containerorkestratie en heeft alreeds een enorme impact gehad op het IT-landschap. Om het inzicht in de werking ervan te verruimen kan gebruik gemaakt worden van Chaos Engineering, die toelaat proactief te experimenteren op een omgeving zodoende een systeem beter te leren kennen. Dit onderzoek richt zich op de educatieve mogelijkheden om chaos engineering tools en experimenten toe te passen in een Kubernetes cluster. Eerst is onderzocht als het voordelig is een cluster lokaal op te zetten via automatisatietools. Drie proof of concepts zijn uitgewerkt nl. een single-node cluster via Minikube en een multi-node cluster via Kubeadm en Kubespray.

Door de tijdrovende procedure om lokale clusters tot stand te brengen drong de vergelijking zich op met een cloudomgeving. Als gevolg werd een cluster via Google Kubernetes Engine (GKE) opgezet. De conclusie is dat een cluster in de cloud opzetten veel sneller verloopt en eveneens extra functionaliteiten bevat die handig zijn in het verder verloop van het onderzoek naar een geschikte chaos engineering tool.

Vervolgens werden in deze cloudomgeving de chaos engineering tools Chaos Toolkit, Chaos Mesh en Litmus vergeleken. Met de nodige voorzichtigheid werd Chaos Mesh gekozen aangezien deze een ruim aanbod aan experimenten bevat, goed gedocumenteerd is, weinig resources vereist in een cluster, en experimenten relatief makkelijk op te zetten en uit te voeren zijn zowel in de terminal als via de browser.

De experimenten in dit onderzoek werden uitgevoerd op twee demo-applicaties en konden aantonen hoe Kubernetes actie onderneemt wanneer het geconfronteerd wordt met realistische problemen die een applicatie in productie kunnen treffen. Eveneens kwamen door het uitvoeren van deze experimenten enkele oplossingen tot stand om de beschikbaarheid van de applicatie(s) te vergroten.

Inhoudsopgave

1	Inleiding	13
1.1	Probleemstelling	14
1.2	Onderzoeksvraag	14
1.3	Onderzoeksdoelstelling	14
1.4	Opzet van deze bachelorproef	15
2	Stand van zaken	17
2.1	Virtualisatie	17
2.1.1	Voordelen van een hypervisor	18
2.1.2	Types hypervisors	18
2.2	Automatisatietools	19
2.2.1	Vagrant	19
2.2.2	Ansible	19

2.3	Evolutie in softwarerachitectuur	19
2.4	Containerisatie	20
2.4.1	Docker	21
2.5	Containerorkestratie	21
2.5.1	Kubernetes	22
2.5.2	Kubernetes architectuur	22
2.5.3	kubectl commandline tool	23
2.5.4	Workloads	24
2.5.5	Services	24
2.5.6	Componenten van een Kubernetes object	25
2.5.7	Kubernetes Networking	26
2.5.8	Helm	26
2.5.9	k9s monitoring tool	27
2.6	Chaos Engineering	28
2.6.1	Geschiedenis van Chaos Engineering	28
2.6.2	Chaos Engineering tools	28
2.6.3	Chaos Engineering experimenten	29
3	Methodologie	31
4	Lokale Kubernetes Clusters	33
4.1	Minikube	33
4.1.1	Vereisten	34
4.1.2	Opzetten van de virtuele machine	34
4.1.3	Minikube installatie	35
4.1.4	Minikube opstarten	36

4.1.5	Conclusie Minikube	36
4.2	KubeAdm	37
4.2.1	Vereisten	37
4.2.2	Opzetten van een multi-node cluster met kubeadm	37
4.2.3	Conclusie Kubeadm	42
4.3	Kubespray	42
4.3.1	Vereisten	42
4.3.2	Een multi-node cluster opzetten met Kubespray	43
4.3.3	Conclusie Kubespray	46
5	Cloud Kubernetes clusters	47
5.1	Google Cloud Platform	47
5.1.1	Opzetten van een GKE cluster via Google Cloud	47
5.1.2	Extra functionaliteit in GKE	49
5.1.3	Conclusie GKE cluster	50
6	Chaos Engineering Tools	51
6.1	Demo-applicaties opzetten	52
6.1.1	Demo-applicatie 1: Nginx/Apache webserver pods	52
6.1.2	Demo-applicatie 2: PodTato-Head	53
6.2	Chaos Toolkit	54
6.2.1	Installatie Chaos Toolkit	54
6.2.2	Een experiment opzetten	56
6.2.3	Experiment 1: Een random pod vernietigen	56
6.2.4	Experiment 2: uitbreiding experiment 1 met steady state	59

6.2.5	Experiment 3: uitbreiding vorige experimenten	60
6.2.6	Experiment 4: bereikbaarheid van Podtato-Head applicatie testen ...	60
6.2.7	Referenties toepassen in een experiment	62
6.2.8	Experiment 5: Een nodefaling simuleren	62
6.2.9	Experiment 6: Rollback toepassen op experiment 5	64
6.2.10	Conclusie ChaosToolkit	65
6.3	Chaos Mesh	65
6.3.1	Installatie Chaos Mesh	66
6.3.2	Toegang voorzien tot het Chaos Dashboard	67
6.3.3	Overzicht van het Chaos Dashboard	68
6.3.4	Pod kill experiment in Chaos Dashboard via New Experiment	69
6.3.5	Een experiment plannen via New schedule	71
6.3.6	Meerdere experimenten opzetten via New workflow	71
6.3.7	Experimenten opzetten in de terminal	72
6.3.8	Conclusie Chaos Mesh	76
6.4	Litmus	77
6.4.1	Installatie	78
6.4.2	Litmus experimenten uitvoeren via de terminal	80
6.4.3	Experiment 1: Nginx pod delete	80
6.4.4	Experiment 2: Nginx pod delete met iteratie	81
6.4.5	Experiment 3: Nginx pod delete met iteratie en probe	82
6.4.6	Experiment 4: Geheugen belasten van nginx pods	83
6.4.7	Experimenten uitvoeren via Litmus ChaosCenter	85
6.4.8	Conclusie Litmus	87

7	Conclusie	89
----------	------------------------	-----------

A	Onderzoeksvoorstel	93
A.1	Introductie	93
A.2	State-of-the-art	94
A.3	Methodologie	94
A.4	Verwachte resultaten	94
A.5	Verwachte conclusies	95
B	Bijlagen	99
B.1	Litmus experimenten opzetten via de terminal	99
B.2	Stappenplan experiment 3 opzetten in ChaosCenter	101
	Bibliografie	103

1. Inleiding

Moderne applicaties maken steeds meer gebruik van een microservices softwarearchitectuur. Hierbij wordt de applicatie opgebouwd als een set van los gekoppelde maar samenwerkende services. Dit biedt verschillende voordelen, maar zorgt ook voor een verhoogde complexiteit.

Containervirtualisatie, een technologie waar Docker een bekend voorbeeld van is, biedt een oplossing om tegemoet te komen aan deze nieuwe vorm van applicatieontwikkeling. Hierbij wordt elke service in een aparte container opgebouwd, waarbij enkel de nodige resources voor deze service gebruikt worden.

Om deze containerapplicaties beter te kunnen schalen, en er onder andere voor te zorgen dat deze continu beschikbaar zijn, wordt containerorkestratie zoals Kubernetes toegepast. Orkestratie is kortweg het automatiseren van containermanagement. Containers van een applicatie kunnen onderverdeeld zijn op één of meerdere nodes (fysieke server/virtuele machine), die gegroepeerd worden in een Kubernetes cluster.

Netflix was het eerste bedrijf die begon te experimenteren op deze clusters om de weerbaarheid ervan te testen. Zij ontwikkelden verschillende tools die foutinjecties van allerlei soorten konden simuleren, om zo te leren hoe ze proactief problemen konden aanpakken. Deze vorm van experimenteren werd later bekend als chaos engineering en wordt alreeds toegepast door enkele van de grootste bedrijven ter wereld.

1.1 Probleemstelling

Containerorkestratie, waarvan Kubernetes een bekend voorbeeld is, wint laatste jaren enorm aan populariteit. Deze nog relatief jonge technologie zal komende jaren ongetwijfeld zijn plaats krijgen in het curriculum van systeem-en netwerkbeheer. Het toepassen van chaos engineering experimenten kan tegelijkertijd een meerwaarde bieden in het leerproces van Kubernetes.

Dit onderzoek is zowel gericht op studenten als lectoren in systeem- en netwerkbeheer en biedt inzicht in welke kennis vereist is om met Kubernetes aan de slag te kunnen gaan. Hierbij zal eerst de nodige aandacht geschonken worden aan de setup van een Kubernetes cluster, zowel lokaal als in een cloudomgeving.

Chaos engineering experimenten toepassen op applicaties in een Kubernetes cluster stimuleert het begrip hoe Kubernetes acties onderneemt wanneer deze geconfronteerd wordt met realistische problemen die een applicatie in een productieomgeving kunnen treffen. Ook deze technologie staat echter nog in de kinderschoenen waardoor de juiste tool vinden in combinatie met de geschikte Kubernetes setup het nodige onderzoek vereist.

1.2 Onderzoeksvraag

Via dit onderzoek wordt een antwoord gezocht op meerdere vragen zoals:

1. Biedt het een meerwaarde om een lokale Kubernetes cluster op te zetten m.b.v. automatisatietools ten opzichte van een Kubernetes cluster via Google Cloud op te zetten?
2. Welke chaos engineering tool die in dit onderzoek aan bod komt (Chaos Toolkit, Chaos Mesh of Litmus) is het meest geschikt om experimenten uit te voeren op een applicatie in een Kubernetes cluster?
3. Welke chaos engineering experimenten zijn relevant om een beter inzicht te creëren in de werking van Kubernetes?

1.3 Onderzoeksdoelstelling

Een eerste doelstelling in dit onderzoek is om meerdere proof-of-concepts te bekomen in het opzetten van een Kubernetes cluster, dit zowel lokaal als in de cloud. Qua lokale setups is gekozen om Minikube, Kubeadm en Kubespray te onderzoeken. Nadien is ook een cluster opgezet via Google Cloud.

Met de kennis hoe deze lokale clusters opgezet worden tegenover een setup in de cloud kan nadien een vergelijking gemaakt worden welke manier best geschikt is om een Kubernetes cluster op te zetten.

De tweede doelstelling is om eenvoudige applicaties en tools te vinden om de brug te

maken tussen Kubernetes en Chaos Engineering. Extra aandacht wordt hierbij vooral geschonken aan de eenvoud van de demo-applicatie(s) en tools, zodat deze de leercurve niet negatief beïnvloeden.

De derde doelstelling is om verschillende chaos engineering tools te testen in de Kubernetes setup die als beste optie uit doelstelling 1 naar boven kwam. Hierbij zal de nodige aandacht geschonken worden aan volgende factoren:

1. de eenvoud om de tool op te zetten.
2. de documentatie die deze tool biedt.
3. de vereiste leercurve om met deze tool aan de slag te kunnen.
4. de eenvoud om experimenten op te zetten en uit te voeren.

Deze drie doelstellingen zullen resulteren in een aanbeveling naar een geschikte Kubernetes setup en chaos engineering tool om experimenten uit te voeren met als ultieme doelstelling het begrip te vergroten in de algemene werking van Kubernetes.

1.4 Opzet van deze bachelorproef

De rest van deze bachelorproef is als volgt opgebouwd:

In Hoofdstuk 2 wordt een overzicht gegeven van de stand van zaken binnen het onderzoeksdomein, op basis van een literatuurstudie.

In Hoofdstuk 3 wordt de methodologie toegelicht en worden de gebruikte onderzoekstechnieken besproken om een antwoord te kunnen formuleren op de onderzoeksvragen.

In Hoofdstuk 4 komt de setup aan bod van drie lokale Kubernetes clusters nl. Minikube, Kubeadm en Kubespray. Hierbij zal steeds een conclusie gevormd worden over elke setup.

In Hoofdstuk 5 wordt de setup besproken van een Kubernetes cluster via Google Cloud en zal nadien ook een conclusie gevormd worden.

In Hoofdstuk 6 zal eerst de nodige aandacht geschonken worden aan het opzetten van de twee demo-applicaties en de monitoring tool k9s. Vervolgens zullen drie verschillende chaos engineering tools genaamd Chaos Toolkit, Chaos Mesh en Litmus onderzocht worden.

In Hoofdstuk 7, tenslotte, wordt de conclusie gegeven en een antwoord geformuleerd op de onderzoeksvragen. Daarbij wordt ook een aanzet gegeven voor toekomstig onderzoek binnen dit domein.

2. Stand van zaken

In deze stand van zaken worden de gebruikte technologieën stapsgewijs besproken om het nodige inzicht te creëren in het onderwerp chaos engineering. Eerst zal uitgelegd worden wat virtualisatie is en welke voordelen dit met zich meebrengt. Vervolgens wordt toegelicht hoe softwareontwikkeling geëvolueerd is van een monolitische naar een microservices architectuur en hoe containerisatie hier een rol in speelt. Nadien zal uitleg gegeven worden over wat de containerorkestratie Kubernetes is, hoe het er voor zorgt dat applicaties dynamisch kunnen schalen en hoe er gemikt wordt naar een zo hoog mogelijke uptime ervan. Tot slot wordt besproken wat chaos engineering inhoudt en wat de link is met containerorkestratie.

2.1 Virtualisatie

Virtualisatie maakt het mogelijk om op een fysieke computer meerdere virtuele machines uit te voeren, elk met zijn eigen besturingssysteem, geheugen, processorkernen en opslagcapaciteit.

Het essentiële onderdeel om de virtuele machines te binden aan de hardware van de fysieke host, en wat de dynamische toewijzing van resources zoals geheugen en CPU mogelijk maakt, is een softwarelaag die men de hypervisor noemt.

2.1.1 Voordelen van een hypervisor

Het gebruik van een hypervisor om virtuele machines op een host aan te maken heeft enkele voordelen (Yfantis, 2020) :

- Snelheid: virtuele machines opzetten verloopt snel in vergelijking met de tijd benodigd om een fysieke server op te zetten.
- (Kost)efficiëntie: men maakt optimaal gebruik van de beschikbare resources op het hostsysteem door deze dynamisch te verdelen over meerdere virtuele machines.
- Overdraagbaarheid: virtuele machines worden bewaard als bestanden op de host en zijn makkelijk overdraagbaar naar andere systemen.

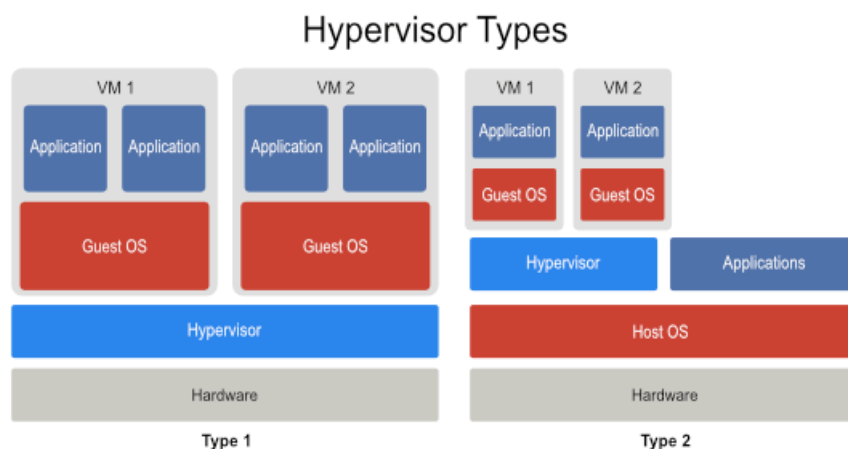
2.1.2 Types hypervisors

Er bestaan twee hypervisor types:

- Type 1: bare-metal / native
- Type 2: hosted

Een bare-metal of native type 1 hypervisor is virtualisatiesoftware die rechtstreeks geïmplementeerd is op de hardware van een host en zich gedraagt zoals een lichtgewicht besturingssysteem. Het voordeel bij dit type hypervisor is dat geen volwaardig besturingssysteem geïnstalleerd wordt en dat het hiermee gevrijwaard blijft van de kwetsbaarheden die dit met zich meebrengt. Voorbeelden van type 1 hypervisors zijn VMware ESXi, Microsoft Hyper-V, Citrix Xen ...

Een hosted type 2 hypervisor is software die bovenop het bestaande besturingssysteem op een host aanwezig is. Het nadeel bij een type 2 hypervisor is de verhoogde latentie doordat de communicatie tussen hardware en hypervisor eerst nog het besturingssysteem moet passeren. Bekende voorbeelden hiervan zijn de softwarepakketten VirtualBox of VMware.



Figuur 2.1: hypervisor types (Vembu, 2019)

2.2 Automatisatietools

2.2.1 Vagrant

Vagrant is open-source software ontworpen door HashiCorp met als doel virtuele omgevingen op te zetten en te beheren. (Vagrant, 2022b) Met behulp van een Vagrantfile, een configuratiebestand waarin specificaties van de omgeving gedefinieerd zijn, kan men via het **vagrant up** commando op een geautomatiseerde manier één of meerdere virtuele machines creëren.

De software VirtualBox dient hierbij alreeds op het systeem aanwezig te zijn, aangezien Vagrant standaard de virtuele machines achterliggend via deze hypervisor type 2 zal creëren. (Vagrant, 2022a)

2.2.2 Ansible

Ansible is open-source configuratiemanagementsoftware ontworpen door RedHat. Het doel van Ansible is om repetitieve en manuele taken te automatiseren via verschillende scripts die gebundeld worden in een Ansible playbook. (RedHat-Ansible, 2022)

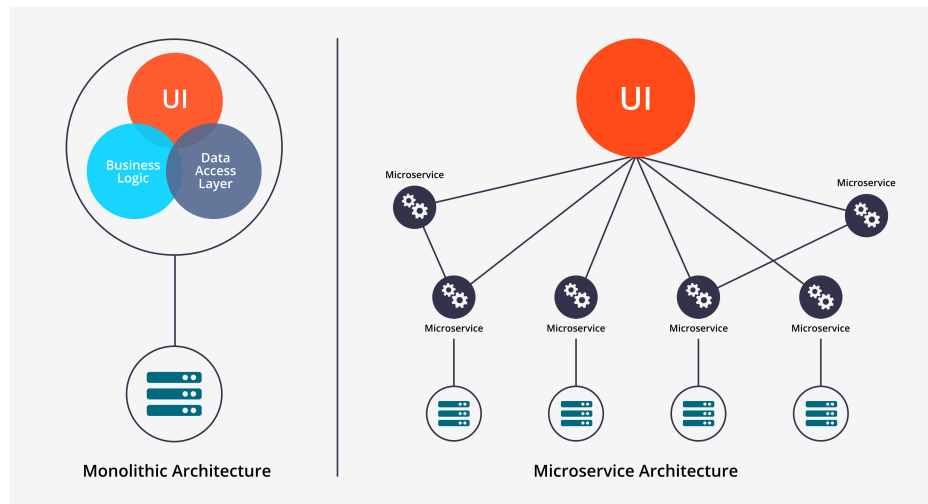
2.3 Evolutie in softwarerachitectuur

Klassieke softwareontwikkeling leverde applicaties op die gebaseerd waren op een monolitische architectuur. Hierin was alle code aanwezig van de applicatie, zowel front- als backend. Dit bracht verschillende voordelen met zich mee waaronder:

- makkelijk qua ontwikkeling omdat IDEs/tools gefocust waren op het bouwen van 1 applicatie.
- vlotter om te testen gezien de code zich allemaal in 1 applicatie bevindt.
- simpel om uit te rollen in productie omdat men de verpakte applicatie slechts moest kopiëren op een server.
- schalen van de applicatie eenvoudig door meerdere kopieën uit te voeren achter een load balancer.

Verskillende nadelen van deze manier van softwareontwikkeling kwamen echter aan het licht wanneer succesvolle applicaties met de tijd begonnen te groeien. Wanneer ontwikkelaars nieuwe functionaliteit wouden toevoegen werd steeds meer code toegevoegd, waardoor de complexiteit van de applicatie steeds verhoogde. Het resultaat hiervan was dat bugs oplossen en features toevoegen steeds moeizamer verliep. Elke keer een functionaliteit veranderde aan de applicatie moest de hele applicatie opnieuw uitgerold worden, wat voor continuous deployment een enorm obstakel is. De betrouwbaarheid van een monolitische applicatie is ook een minpunt aangezien een bug ervoor kan zorgen dat de beschikbaarheid van de volledige applicatie in het gedrang komt. (Richardson, 2015)

Om deze problemen met monolitische applicaties aan te pakken gebruikten grote bedrijven zoals Amazon, eBay, en Netflix een nieuwe softwareontwikkeling op basis van een microservices architectuur. Het idee achter deze nieuwe manier van softwareontwikkeling is om de applicatie op te splitsen in verschillende kleine geconnecteerde services die elk een set van features of specifieke functionaliteit bevatten.



Figuur 2.2: evolutie in softwarearchitectuur (Sanjaya, 2020)

2.4 Containerisatie

Applicaties gebouwd via een microservices architectuur brengen services onder in containers. Een container op zich bevat geen besturingssysteem maar verpakt en isoleert de code van één applicatie inclusief de gerelateerde configuratiebestanden en afhankelijkheden die deze nodig heeft. Het voordeel van containers is dat ze snel opgestart kunnen worden doordat ze weinig resources vereisen, en makkelijk overdraagbaar zijn naar verschillende omgevingen. Meerdere containers kunnen actief zijn op een systeem en maken ook allemaal gebruik van hetzelfde besturingssysteem. (Singh, 2020)

Ondanks dat containers net zoals virtuele machines een vorm van virtualisatie zijn, is er toch een belangrijk onderscheid te maken. Containers zijn namelijk een vorm van besturingssysteemvirtualisatie, dit tegenover virtuele machines die aan hardwarevirtualisatie doen. Hierdoor verbruiken containers dus minder resources ten opzichte van virtuele machines. (Holt & Huang, 2018)

Containers en virtuele machines kunnen gecombineerd worden om virtuele omgevingen te maken waarin software ontwikkeld en getest kan worden. Containers blijven wel afhankelijk van het besturingssysteem. Men kan geen Windows containers in een Linux omgeving opstarten of omgekeerd. Om de interactie met het besturingssysteem op de host mogelijk te maken is een container runtime op het systeem nodig zoals Docker Engine, CRI-O, LXC ...

2.4.1 Docker

De oorsprong van containers kan herleid worden naar 1979 toen men voor het eerst een proces kon isoleren via de system call `chroot` in Unix V7. Pas vanaf 2000 kwamen bedrijven zoals Open Virtuoso en Google met nieuwe ontwikkelingen zoals `cgroups` en `LXC` (Linux containers) die de concepten rond containerisatie meer vorm begonnen te geven. Het was pas echter toen Docker in 2013 op de markt kwam dat het gebruik van containers qua populariteit explodeerde. Docker maakte het onderscheid door een volledig ecosysteem aan te bieden voor container management. (Osnat, 2020)

Door Docker Engine op het systeem te installeren kan men gecontaineriseerde applicaties op gelijk welke infrastructuur uitvoeren. Voordien kon men hinder ondervinden doordat afhankelijkheden van applicaties ervoor zorgden dat het uitvoeren ervan in een andere omgeving problemen kon geven. Dit probleem wordt vaak omschreven als de 'dependency hell'.

Met Docker bundelt een ontwikkelaar een applicatie en zijn afhankelijkheden in een container die overal uitvoerbaar is. Om een applicatie in een container te plaatsen is een `Dockerfile` nodig. Dit bestand plaatst men bij de applicatie en is in essentie een set instructies waarin de applicatiecode inclusief de benodigde afhankelijkheden gekopieerd worden naar de container en hoe de applicatie opgestart wordt. De `Dockerfile` zal vervolgens gebruikt worden om een image te maken, die opgeslagen wordt in een publieke repository zoals Docker Hub of een private repository.

Wanneer men de applicatie op een ander systeem wil opbouwen, waar Docker ook geïnstalleerd is, kan men via het `docker run` commando de image van de applicatie uit een repository halen en hiermee een container op het systeem lanceren die de applicatie bevat.

2.5 Containerorkestratie

In een productieomgeving moet men ervoor zorgen dat applicaties continu beschikbaar blijven en deze fouttolerant zijn. Wanneer een container faalt, moet een andere container automatisch opgestart worden om dit op te vangen. Wanneer meer gebruik gemaakt wordt van een applicatie, moet deze automatisch kunnen schalen om deze extra vraag op te vangen. Om deze redenen en nog veel meer wordt containerorkestratie toegepast.

Containerorkestratie is het volledige proces rond het uitrollen, beheren en schalen van gecontaineriseerde applicaties. De bekendste speler in containerorchestratie is ongetwijfeld Kubernetes (K8s). Andere aanbieders in deze markt zijn o.a. Nomad, Docker Swarm, Amazon Elastic Kubernetes Service (EKS) ... Bij het opzetten van de verschillende virtuele omgevingen die later in hoofdstuk 3 uitvoerig besproken worden werd enkel gebruik gemaakt van Kubernetes.

2.5.1 Kubernetes

Ongeveer 1 jaar nadat de containerisatiesoftware Docker in 2013 op de markt verscheen, werd Kubernetes aan het grote publiek voorgesteld. Oorspronkelijk werd het ontwikkeld door Google, maar nu wordt het project onderhouden door de Cloud Native Computing Foundation (<https://www.cncf.io/>).

Kubernetes wordt op hun eigen website omschreven als een overdraagbaar, uit te breiden, open-source platform voor het beheer van gecontaineriseerde applicaties en services, dat zowel declaratieve als geautomatiseerde configuratie toelaat.

2.5.2 Kubernetes architectuur

Het grootste object in de Kubernetes architectuur noemt men een cluster. Hierin worden fysieke of virtuele machines als nodes gegroepeerd. Men kan afhankelijk van het aantal nodes twee types clusters onderscheiden:

- een single-node cluster
- een multi-node cluster

Wanneer men meerdere nodes in een cluster onderbrengt kan het onderscheid gemaakt worden tussen:

- de control-plane node(s), in oudere versies nog de master node genoemd.
- de worker node(s)

In een single-node cluster is de enige node zowel de control-plane als worker node. Later in hoofdstuk 3 zullen verschillende Kubernetes distributies gebruikt worden om een lokale cluster op te zetten. Hierbij zal zowel een single-node cluster via Minikube, alsook multi-node clusters via Kubeadm en Kubespray opgezet worden. Ook zal aangetoond worden hoe men via Google Kubernetes Engine (GKE) een cluster in Google Cloud kan opzetten.

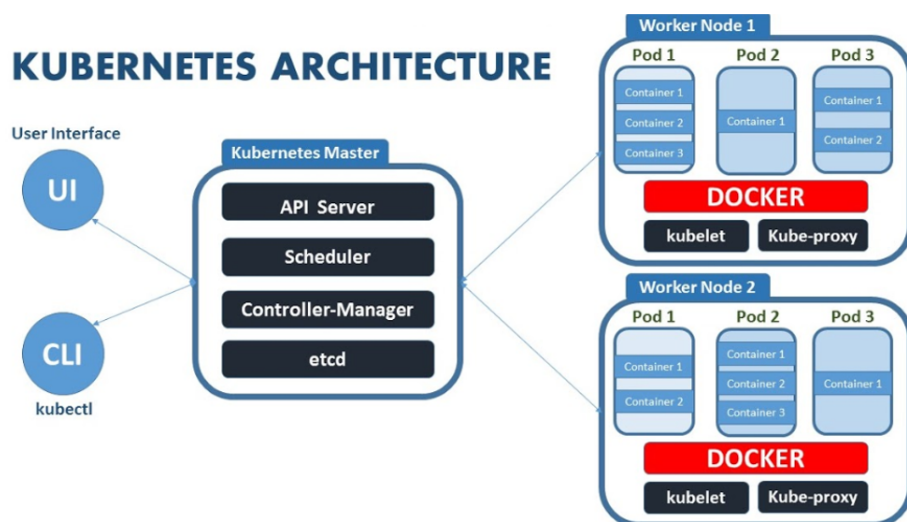
De infrastructuur van een Kubernetes cluster bevat een aantal componenten die afhankelijk van het type node zullen geïnstalleerd worden. Op deze manier zijn de verantwoordelijkheden in een cluster duidelijk afgelijnd. De volgende componenten kan men in elke cluster terugvinden:

- **API server:** de front-end voor de control-plane en laat interactie met een cluster toe.
- **etcd:** een key-value store die gebruikt wordt om clusterdata op te slaan.
- **scheduler:** verantwoordelijk voor de distributie van containers naar de nodes.
- **controller:** verantwoordelijk voor het monitoren en reageren wanneer nodes/pods uitvallen en beslist als nieuwe nodes/pods gecreëerd moeten worden.
- **container runtime:** de onderliggende software die het mogelijk maakt om containers uit te voeren bv. Docker.
- **kubelet:** een agent die op elke node actief is en verantwoordelijk is voor de correcte uitvoering van de containers.

Elk infrastructuurcomponent hierboven opgesomd zal steeds ondergebracht worden in een pod, wat het kleinste Kubernetes object is. Meer over pods kan u later lezen in sectie 2.5.4. De API-server, etcd, scheduler en controller zijn de componenten die verantwoordelijk zijn voor het beheer van de cluster en zijn samen gebundeld in de control-plane node. Deze staat in voor het beheer van de worker nodes en de pods in de cluster.

Een worker node bevat de pods van een workload, of anders geformuleerd de containers van een applicatie. De kubelet en een kube-proxy zijn de enige infrastructuurcomponenten benodigd op een worker node. De kubelet is verantwoordelijk voor de communicatie met de control-plane node en zal deze eveneens informeren over de interne staat van de worker node. De kube-proxy staat in voor de netwerkcommunicatie.

Een overzicht van de verschillende infrastructuurcomponenten en de onderverdeling over de verschillende nodes kan u zien in figuur 2.3.



Figuur 2.3: k8s clusterarchitectuur (Collabnix, 2019)

2.5.3 kubectl commandline tool

Wanneer men acties wil uitvoeren in een cluster dan communiceert men steeds via de API-server op de control-plane node. Men kan dit doen via een user interface ofwel in een terminal via de kubectl commandline tool. Deze tool wordt gebruikt om applicaties uit te rollen en te beheren binnenin een Kubernetes cluster. Elk commando dat men via deze tool uitvoert zal intern omgevormd worden tot een HTTP-request richting de API-server. (Biradar, 2019)

2.5.4 Workloads

Een workload is een gecontaineriseerde applicatie in Kubernetes. Containers worden nooit rechtstreeks op een worker node ondergebracht maar worden door Kubernetes in pods geplaatst. Een pod kan meer dan 1 container bevatten van een applicatie indien de functionaliteit van deze containers verschillend is. Meerdere instanties van dezelfde applicatie zullen dus nooit binnenin één pod actief zijn. Elke container binnen een pod zal gebruik maken van dezelfde systeembronnen. Het één-container-per-pod model is de meest gebruikte Kubernetes use case. (Kubernetes, 2022c)

De werking van een pod kan op vele manieren verstoord worden en dit manueel beheren zou een tijdrovende taak zijn. Om deze redenen bestaan volgende ingebouwde workload resources in Kubernetes, die deze manuele taken kunnen automatiseren:

- Deployments en ReplicaSets
- StatefulSets
- DaemonSets
- Jobs en CronJob

In dit onderzoek wordt enkel gebruik gemaakt van Deployments en ReplicaSets om pod-creatie te automatiseren. Kubernetes kan ook nog verder uitgebreid worden via Custom Resource Definitions (CRDs) om extra workload resources te creëren.

Een Deployment wordt gebruikt om de creatie van pods of aanpassingen aan pods in een gecontaineriseerde applicatie te automatiseren. Via een Deployment kan men de pods van een applicatie schalen, gecontroleerd updates uitvoeren zonder downtime, en indien noodzakelijk terugrollen naar vorige versies van de applicatie.

Het schalen van de pods gebeurt via een ReplicaSet, wat als doel heeft om continu een vooraf gedefinieerd aantal pods van een applicatie (op basis van de geconfigureerde labels) actief te houden. Men hoeft deze echter niet manueel te configureren aangezien een ReplicaSet steeds automatisch gecreëerd zal worden tijdens de creatie van een Deployment.

2.5.5 Services

Elke pod in een cluster heeft zijn eigen IP-adres, gedeeld door de onderliggende containers. Pods kunnen echter falen of naar een andere node verplaatst worden, waarop ze een nieuw IP-adres toegewezen krijgen. Om een oplossing te bieden voor dit probleem gebruikt men in Kubernetes **Services**. Dit zijn Kubernetes objecten (zoals pods, Deployments, ReplicaSets ...) die communicatie tussen verschillende componenten binnen en buiten een applicatie mogelijk maken. Meerdere pods die op verschillende nodes aanwezig zijn kunnen samen vertegenwoordigd worden door een Service.

Een Service krijgt net zoals de pods eronder eveneens een IP-adres toegewezen. Kubernetes bevat een ingebouwde (eenvoudige) load balancer die het verkeer kan leiden richting

de verschillende pods onder een Service.

Er bestaan vier verschillende types Services:

- **ClusterIP**: maakt de Service enkel binnen de cluster bereikbaar zodat communicatie tussen verschillende Services mogelijk is.
- **NodePort**: maakt de Service buiten de cluster bereikbaar zodat men van buitenaf de applicatie kan bereiken.
- **LoadBalancer**: idem zoals NodePort maar maakt gebruik van een load balancer bij een cloud provider.
- **ExternalName**: mapt de Service aan het geconfigureerde externalName veld.

Om te controleren als een Service succesvol gecreëerd is kan men het commando **kubectl get svc** gebruiken.

2.5.6 Componenten van een Kubernetes object

Wanneer men via een declaratieve methode een Kubernetes object wil creëren dient men eerst een YAML-configuratiebestand te schrijven. De naam van dit bestand kiest men vrij bv. pod-definition.yaml.

In elke definitie van een Kubernetes object dienen 4 verplichte velden aanwezig te zijn:

- **apiVersion**: gebruikte versie van de Kubernetes API om het object te creëren
- **kind**: type object die gecreëerd wordt
- **metadata**: data waarmee het object uniek geïdentificeerd kan worden bv. naam, namespace ...
- **spec**: de gewenste staat van het object

Zie onderstaand voorbeeld van een eenvoudige pod-definitie:

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
    type: front-end
spec:
  containers:
    - name: nginx-container
      image: nginx
```

Wanneer de configuratie geschreven is slaat men het bestand vervolgens op en kan men het object creëren via het commando **kubectl apply -f [YAML-configuratiebestand]**. Met behulp van het commando **kubectl get pods** of het algemenere commando **kubectl get all** kan het zopas gecreëerde object opgevraagd worden.

Om het overzicht te bewaren over de Kubernetes objecten zoals pods, deployments, services ... die in een cluster actief zijn wordt gebruik gemaakt van labels. Verder kan men ook nog onderscheid creëren door gebruik te maken van namespaces, wat een manier is om een aparte groep te vormen waaronder Kubernetes objecten actief zijn. Het toepassen van namespaces zal later in hoofdstuk 2.6 Chaos Engineering gebruikt worden om de scope van de experimenten te bepalen.

2.5.7 Kubernetes Networking

Elke pod in een Kubernetes cluster krijgt een IP-adres toegewezen vanuit een intern gebruikt private netwerk vb. 10.244.x.x. Deze range verschilt met de range van het IP-adres van de node waarop deze pods aanwezig zijn vb. 192.168.1.x

In een multi-node cluster zal dit bovenstaande voorbeeld echter een probleem vormen want elke node wordt opgezet met hetzelfde intern gebruikte private netwerk. Hierdoor zouden pods op verschillende nodes in een cluster hetzelfde IP-adres toegewezen kunnen krijgen wat zal leiden tot conflicten.

Wanneer een cluster opgezet wordt zal Kubernetes géén oplossing bieden voor dit probleem maar wordt verwacht dat we zelf een gepaste netwerkoplossing installeren die beantwoordt aan enkele fundamentele vereisten:

- Alle pods in een cluster moeten met elkaar kunnen communiceren zonder Network Address Translation (NAT) te configureren.
- Alle nodes moeten kunnen communiceren met alle pods en vice-versa zonder NAT te configureren.

Om een netwerkoplossing te bieden gebruikt men een Container Network Interface (CNI), wat een specificatie is voor het configureren van netwerkinterfaces voor Linux containers. Een CNI installeert men via een plugin. Meerdere CNI's kunnen operationeel zijn in een cluster en ondertussen verschillende functionaliteiten invullen bv. een CNI die de routing voorziet terwijl een andere CNI zich ontfert over de netwerkbeveiliging. Er zijn heel wat CNI plugins waaruit men kan kiezen, waarvan enkele bekende o.a. Calico, WeaveNet, Cilium en Istio zijn. (Power, 2019)

In dit onderzoek zal op verschillende manieren een lokale omgeving voor een Kubernetes cluster opgezet worden. In deze omgevingen zal gebruik gemaakt worden van WeaveNet en Calico. Een cluster opzetten in de Google Cloud (GKE) zal eveneens aan bod komen. Een voordeel van deze cloudomgeving is dat er alreeds standaard gebruik gemaakt wordt van een CNI genaamd kubenet na het opzetten van de cluster.

2.5.8 Helm

Een besturingssysteem maakt vaak standaard gebruik van een pakketbeheerder (of package manager). Denk maar aan apt bij de Linux distributie Debian, of yum bij distributie

Red Hat Enterprise Linux. Dit maakt het mogelijk om software te (de)installeren en softwareupgrades uit te voeren.

Ook Kubernetes maakt gebruik van een eigen package manager genaamd Helm, die het mogelijk maakt om Kubernetes applicaties te beheren. Via Helm charts kan men de meest complexe Kubernetes applicaties definiëren, installeren en upgraden.

Later in dit onderzoek zal één van de chaos engineering tools genaamd 'Chaos Mesh' in een cluster in de Google Cloud geïnstalleerd worden met behulp van Helm.

2.5.9 k9s monitoring tool

De tool **k9s** is een terminal-gebaseerde UI waarmee men makkelijker de objecten in een Kubernetes cluster kan beheren en observeren. K9s kan gebruikt worden voor monitoring doeleinden aangezien het continu de Kubernetes cluster afspeurt voor wijzigingen. Met behulp van deze tool zullen enkele van de chaos engineering experimenten die later in dit onderzoek aan bod komen opgevolgd worden.

De k9s website biedt heel wat verschillende opties om de tool te installeren. (K9s, 2022) In dit onderzoek is gekozen om k9s te installeren door een archiefbestand van de k9s binary in de Google Cloud Shell te downloaden en vervolgens uit te pakken. Om dit proces te herhalen kan men volgende stappen uitvoeren:

```
# k9s binary downloaden in de terminal
$ wget https://github.com/derailed/k9s/releases/download/v0.25.18/k9s_Linux_x86_64.tar.gz

# Uitpakken .gz archiefbestand
$ tar -xvf k9s_Linux_x86_64.tar.gz

# k9s binary verplaatsen naar /usr/bin
$ sudo mv k9s /usr/bin/

# Verwijder het gedownload .gz archiefbestand
$ rm k9s_Linux_x86_64.tar.gz

# Pods in een specifieke namespace weergeven
$ k9s -n [namespace]
```

Deze tool bevat heel wat functionaliteit die in dit onderzoek echter niet aan bod komt. Indien men deze tool verder wil ontdekken kan deze bron gebruikt worden: <https://medium.com/flant-com/k9s-terminal-ui-for-kubernetes-ae8ad8b0303f>

2.6 Chaos Engineering

Wanneer men de term chaos engineering online opzoekt zal men ongetwijfeld de zin 'breaking things on purpose' (opzettelijk dingen breken) tegenkomen. Deze zin dekt maar een fractie van de lading wat chaos engineering werkelijk is. Om een meer volledige beeld te vormen kan men de volgende definitie in acht nemen: Chaos Engineering is de discipline van het experimenteren op een systeem met de intentie om vertrouwen te krijgen in de capaciteiten van het systeem om te volharden tijdens turbulente omstandigheden in een productieomgeving. (Eliot, 2019)

Met behulp van chaos engineering experimenten kunnen we proactief problemen oplossen, falingen en storingen voorkomen, en nieuwe inzichten ontwikkelen over de werking van een applicatie of systeem.

2.6.1 Geschiedenis van Chaos Engineering

Netflix is een pionier wat betreft chaos engineering. In 2007 lanceerden zij, bovenop hun bestaande DVD-via-mail service, hun streamingdienst die vandaag de dag wereldwijd gekend is. Door het groeiende succes in de beginjaren van deze dienst ondervonden zij ook de nadelen van hun monolitische applicatie toen in 2008 hun service drie dagen lang onderbroken werd door een grootschalige databasecorruptie. Daarop besloten zij om hun monolitische applicatie die actief was op lokale server racks te herbouwen naar een applicatie met een microservices architectuur in de AWS cloud. Deze migratie duurde maar liefst zeven jaar en werd pas in 2016 afgerond.

Door de complexiteit die een microservices architectuur met zich meebracht ontstond een nieuwe aanpak om falingen te voorkomen. Door tools te ontwikkelen zoals Chaos Monkey begonnen zij doelbewust en proactief fouten te injecteren in hun applicatie, door willekeurige instanties en services te vernietigen. De bedoeling van Chaos Monkey was om de engineers bij Netflix aan te moedigen om software services te maken die falingen kunnen weerstaan van individuele instanties. (Basiri e.a., 2016)

Later werden nog extra tools ontworpen die elk op zich een bepaalde functionaliteit hadden qua foutinjectie. Deze set tools noemde men de 'Simian Army'. Zo wist Netflix via hun proactieve aanpak een weerbare architectuur te creëren door middel van deze tools die nu bekend zijn als chaos engineering tools.

2.6.2 Chaos Engineering tools

Vooraleer men van start kan gaan met chaos experimenten moet men eerst een geschikte tool kiezen. Chaos engineering is nog steeds relatief nieuw, waardoor sommige tools die vandaag de dag beschikbaar zijn nog onderontwikkeld of slecht gedocumenteerd zijn.

In dit onderzoek worden volgende tools getest die toegepast kunnen worden op een Kubernetes cluster:

- Chaos Toolkit
- Chaos Mesh
- Litmus

2.6.3 Chaos Engineering experimenten

Chaos Engineering experimenten verlopen volgens 4 theoretische principes:

- Definieer het normale, verwachte gedrag van het systeem (= de steady state)
- Vorm een hypothese dat het normale gedrag overeind zal blijven
- Introduceer een fout-injectie in het systeem
- Probeer de hypothese te ontkrachten

In de praktijk worden deze experimenten in YAML-bestanden opgeslaan en variëren deze qua inhoud afhankelijk van de gekozen tool. Er zijn experimenten die bv. willekeurig de werking van pods verstoren, pods vernietigen, communicatie tussen pods tijdelijk verstoren, nodes vernietigen ... In hoofdstuk 3 Methodologie worden enkele van deze uitgevoerde experimenten en het resultaat ervan bestudeerd.

De essentie van het uitvoeren van chaos engineering experimenten is het bestuderen hoe Kubernetes reageert en hoe men uit de resultaten ervan vervolgens structureel en proactief zaken kan verbeteren om de beschikbaarheid en fouttolerantie van een applicatie te verhogen.

Om de impact van een experiment steeds te beperken zal gebruik gemaakt worden van namespaces, waarin de demo-applicatie actief zal zijn. Op deze manier voorkomt men dat experimenten een onverwachte en negatieve impact hebben op andere Kubernetes objecten buiten deze namespace.

Het soort experimenten die uitgevoerd worden zullen ook afhankelijk zijn van de opgezette clusteromgeving. Men kan bijvoorbeeld geen experimenten uitvoeren die een node zullen uitschakelen op een single-node cluster, omdat hierdoor de enige node geraakt wordt en dit zonder twijfel kwalijke gevolgen zal hebben.

3. Methodologie

Bij de start van dit onderzoek moet eerst een basiskennis Kubernetes verworven worden. Hiervoor is via het online leerplatform Udemy de cursus 'Kubernetes for the absolute beginners' geraadpleegd. (Mannambeth, 2021)

Vooraleer men van start kan gaan met het uitvoeren van chaos engineering experimenten is een Kubernetes cluster nodig waarin een demo-applicatie actief is. In schoolcontext werd steeds gebruik gemaakt van een type 2 hypervisor zoals VirtualBox om virtuele machines op te zetten, aangevuld met automatisatietools zoals Vagrant en Ansible.

Vanuit dit standpunt zijn volgende Kubernetes distributies eerst onderzocht om een lokale Kubernetes cluster op te zetten:

- **Minikube:** een single-node Kubernetes cluster opgezet op een Ubuntu Linux VM.
- **Kubeadm:** een multi-node Kubernetes cluster opgezet m.b.v. Vagrant, die 1 control-plane (master) node en 2 worker nodes bevat.
- **Kubespray:** een multi-node Kubernetes cluster opgezet m.b.v. Ansible, die 2 control-plane en 2 worker nodes bevat.
- **KinD:** een multi-node Kubernetes cluster opgezet op één Ubuntu Linux VM. Alle nodes bestaan in de vorm van Docker containers, waardoor men een multi-node cluster kan simuleren binnen 1 VM. Deze cluster bevat 1 control-plane en 2 worker nodes.

Door problemen met de bereikbaarheid van de Kubernetes cluster na de setup via KinD is besloten deze setup verder niet te beschrijven in dit onderzoek.

Om een basis te verwerven in chaos engineering is de online Udemy cursus 'Kubernetes Chaos Engineering With Chaos Toolkit And Istio' geraadpleegd. Deze cursus is geba-

seerd op het 'boek The DevOps Toolkit: Kubernetes Chaos Engineering' en wordt gegeven door de auteur Victor Farcic. (Farcic & Pope, 2020)

Tijdens het opzetten van lokale clusters zoals Minikube en Kubeadm werden alreeds de eerste experimenten uitgevoerd met behulp van de Chaos Engineering tool Chaos Toolkit. Hierbij werd oorspronkelijk gebruik gemaakt van de demo-applicatie 'go-demo-8' die ook gebruikt werd in de Udemy cursus 'Kubernetes Chaos Engineering With Chaos Toolkit And Istio'.

Het nadeel aan deze demo-applicatie is dat deze gebruik maakte van een achterliggende database die nood had aan het Kubernetes object 'PersistentVolume (PV)'. Dit opslag-gerelateerde object is gekoppeld aan andere gerelateerde Kubernetes objecten zoals een 'PersistentVolumeClaim (PVC)' en 'StorageClass (SC)'.

Een single-node Minikube cluster heeft een default StorageClass die gebruikt kan worden door de 'go-demo-8' demo-applicatie, maar is tegelijkertijd als single-node te beperkt om een goed inzicht te krijgen in de werking van Kubernetes.

Andere lokale multi-node setups zoals Kubeadm en Kubespray bevatten géén default StorageClass waardoor de 'go-demo-8' applicatie niet werkte. Hierdoor kwamen de eerste nadelen aan het licht van een lokale Kubernetes setup. Pogingen om de benodigde Kubernetes objecten achteraf te installeren bleken geen succes waardoor andere opties onderzocht werden.

Eens het nodige onderzoek verricht werd hoe men lokale Kubernetes clusters kon opzetten, en welke nadelen dit met zich meebracht, is vervolgens onderzocht welke meerwaarde een cluster in de Google Cloud kon bieden.

Vanaf dit punt is gekozen om af te stappen van de 'go-demo-8' applicatie en onderzoek te verrichten naar een andere applicatie die geen nood had aan achterliggende Kubernetes objecten. De eenvoud van de applicatie alsook het visuele aspect was hierbij de belangrijkste drijfveer in de zoektocht naar een geschikte applicatie. Uiteindelijk werd gekozen voor twee demo-applicaties, waarvan de eerste applicatie simpelweg opgezet wordt door zelf twee Deployments te creëren die resulteren in enkele Nginx en Apache pods inclusief Services. Anderzijds werd ook gekozen voor de applicatie Podtato-Head, die meer beantwoordt aan het visuele aspect doordat deze applicatie een aardappelmannetje voorstelt waarin elke ledemaat een pod is.

Rond deze tijd kwam ook de monitoring tool k9s ter sprake, die in het verder verloop van het onderzoek een meerwaarde bleek te zijn doordat de experimenten via deze tool live opgevolgd konden worden. Tergelijkertijd werd het onderzoek gestart naar de twee andere chaos engineering tools genaamd Chaos Mesh en Litmus. Deze tools bieden eveneens een grafische userinterface (GUI) aan waardoor men experimenten ook in de browser kan opzetten i.p.v. via de terminal. Om de werking van deze twee tools beter te begrijpen is gekozen om zowel experimenten uit te voeren via de GUI als via de terminal.

Alle bevindingen en experimenten uitgevoerd via deze drie verschillende tools werden gefilterd om enkel de relevante info en experimenten over te nemen in dit onderzoek.

4. Lokale Kubernetes Clusters

Vooraleer men chaos engineering experimenten kan uitvoeren is er eerst een Kubernetes cluster nodig. De bedoeling van dit onderzoek is om deze omgeving eveneens te kunnen reproduceren voor educatieve doeleinden, in de veronderstelling dat deze geïntegreerd kan worden in een toekomstig curriculum.

Vandaar start dit onderzoek met het opzetten en testen van Kubernetes clusters in een lokale omgeving. Eerst wordt beschreven hoe men een single node Minikube cluster kan opzetten, om vervolgens de setup van multi-node clusters via KubeAdm en Kubespray te bespreken. Voor elk van deze lokale setups zal een conclusie geformuleerd worden waarin de voor- en/of nadelen kort aan bod komen.

Later zal in Hoofdstuk 5 besproken worden hoe men een cluster kan opzetten via het Google Cloud Platform.

4.1 Minikube

Minikube is een snelle en makkelijke manier om een lokale Kubernetes cluster op te zetten, dit voor zowel Windows, MacOS en Linux omgevingen. (Minikube, 2022) Men kan zowel een single-node als multi-node cluster opzetten via Minikube, maar beiden zullen op één fysieke machine bestaan.

In dit onderzoek is gekozen om Minikube als een single-node cluster te installeren op een Linux VM met distributie Ubuntu (Bionic) 18.04. Er is voor een setup op een Linux virtuele machine gekozen om praktische redenen. Minikube kwam aan bod tijdens het volgen van de online cursus 'Kubernetes for the absolute beginners', waarbij een single-node

Minikube cluster oorspronkelijk opgezet werd als een afzonderlijke virtuele machine. De `kubect`l commando's, gebruikt om te communiceren met deze cluster, werden vanuit Powershell uitgevoerd. Nadien, toen de eerste chaos engineering tool 'Chaos Toolkit' aan bod kwam en deze via Python geïnstalleerd moest worden, kwamen de nadelen van deze setup in een Windowsomgeving naar boven.

Op het eind van de online cursus 'Kubernetes for the absolute beginners' werd de tool Kubeadm geïntroduceerd, die in Hoofdstuk 4.2 besproken wordt.

4.1.1 Vereisten

Voor de Linux virtuele machine op te zetten, waar men nadien de Minikube Kubernetes cluster op kan installeren, heeft men de tool Vagrant en een Vagrantfile met de beschrijving van de Ubuntu Linux virtuele machine nodig.

Minikube vereist qua resources minimum 2 CPU, 2 GB RAM en 20 GB schijfruimte. In de setup van de Ubuntu Linux virtuele machine is gekozen om dubbel zoveel CPU en RAM toe te wijzen, om een veilige buffer te voorzien.

4.1.2 Opzetten van de virtuele machine

Volgend stappenplan kan u volgen om de virtuele machine op te zetten:

- Maak een directory aan op uw host systeem met een gepaste naam vb. `ubuntu-bionic`. Hierin zullen alle configuratiebestanden van de virtuele machine in de toekomst bewaard worden.
- Maak een bestand 'Vagrantfile' aan in deze directory (zonder een extensie).
- Plaats de onderstaande configuratie in de zopas gecreëerde Vagrantfile en sla op.

Dit is een aangepaste versie van de oorspronkelijke 'gusztavvargadr/ubuntu-desktop' Vagrantfile (Varga, 2022), waarmee een virtuele machine opgezet wordt met de naam 'ubuntu-desktop' en qua resources 4 GB RAM-geheugen en 4 CPU's bevat.

```
-----
Vagrant.configure("2") do |config|
  config.vm.box = "gusztavvargadr/ubuntu-desktop"
  config.vm.hostname = "ubuntu-desktop"
  config.vm.define "ubuntu-desktop" do |node|
    node.vm.provider "virtualbox" do |vb|
      vb.name = "ubuntu-desktop"
      vb.memory = "4096"
      vb.cpus = 4
    end
  end
end
-----
```

Via een terminalsessie te openen in de directory waar de Vagrantfile staat, en vervolgens het commando **vagrant up** uit te voeren, zal de installatieprocedure voor de setup van de virtuele machine opstarten. Na afloop kan men via VirtualBox inloggen op de virtuele machine met de credentials vagrant/vagrant.

4.1.3 Minikube installatie

Eens men ingelogd is op de virtuele machine opent men een terminal sessie. Volg deze stappen om Minikube te installeren: (Simic, 2020)

1. Update het systeem en installeer vereiste packages:

```
# Update het systeem
$ sudo apt-get update -y

# Upgrade het systeem
$ sudo apt-get upgrade -y

# Installeer de curl package
$ sudo apt-get install curl

# Installeer de apt-transport-https package
$ sudo apt-get install apt-transport-https
```

2. Installeer Minikube:

```
# Download de minikube binary
$ wget https://storage.googleapis.com/minikube/releases/\
latest/minikube-linux-amd64

# Kopieer binary naar /usr/local/bin/
$ sudo cp minikube-linux-amd64 /usr/local/bin/minikube

# Verander de rechten van de binary
$ sudo chmod 755 /usr/local/bin/minikube

# Verifieer de installatie
$ minikube version
```

3. Installeer de kubectl tool om de communicatie met- / en het beheer van de Minikube cluster mogelijk te maken:

```
# Download de kubectl tool binary
$ curl -LO \
https://storage.googleapis.com/kubernetes-release/release/ \
curl -s https://storage.googleapis.com/kubernetes-release/ \
release/stable.txt ' /bin/linux/amd64/kubectl

# Maak de binary uitvoerbaar
```

```
$ chmod +x ./kubectl

# Verplaats de binary naar /usr/local/bin/
$ sudo mv ./kubectl /usr/local/bin/kubectl

# Verifieer de installatie
$ kubectl version -o json
```

4.1.4 Minikube opstarten

Op dit moment zijn alle benodigde zaken geïnstalleerd maar is de Minikube cluster nog niet actief. De cluster start men op m.b.v. commando **minikube start**. Hou er rekening mee dat dit commando steeds zal uitgevoerd moeten worden wanneer men de virtuele machine herstart. Om de werking van de geïnstalleerde Kubernetes componenten te controleren kan men commando **minikube status** gebruiken. Zie onderstaand voorbeeld ter illustratie:

```
$ minikube status

minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured
```

4.1.5 Conclusie Minikube

Het opzetten van de virtuele machine en de Minikube cluster vereisen eenvoudige manuele stappen die eventueel via scripting of andere vormen van automatisatie versneld kunnen worden.

De setup van Minikube is eenvoudig maar de eenvoud van een single-node cluster kan als nadeel aanzien worden wanneer men meer inzicht wil verwerven in Kubernetes. Men kan de principes rond control-plane (master) en worker nodes moeilijker vatten aangezien alles zich op dezelfde node bevindt. Sommige van de chaos engineering experimenten die aan bod komen in dit onderzoek zullen daardoor ook niet toegepast kunnen worden in deze omgeving.

Vandaar besloten is om verder onderzoek te verrichten naar tools zoals Kubeadm en Kubespray om een multi-node cluster op te zetten.

4.2 KubeAdm

Via de Kubeadm tool kan men acties uitvoeren om een simpele, operationele en beveiligde Kubernetes cluster op te zetten op een alreeds bestaande infrastructuur. De scope van deze tool is gelimiteerd tot de lokale node en de Kubernetes API, en is bedoeld om toegepast te worden als bouwblok door andere, meer complete tools. (Kubeadm, 2021) Er zijn alreeds enkele Kubernetes distributies, waaronder de later besproken tool Kubespray, die de Kubeadm tool gebruiken om een cluster te bootstrappen.

In dit onderzoek is gekozen om eerst via Vagrant drie virtuele machines op te zetten en nadien via de Kubeadm tool een multi-node Kubernetes cluster te creëren bestaande uit één control-plane node en twee worker nodes. De kubeadm commando's **kubeadm init** die de control-plane node van de cluster zal opzetten, en **kubeadm join** die de worker nodes nadien zullen toevoegen, zullen ervoor zorgen dat een multi-node cluster tot stand komt.

4.2.1 Vereisten

Net zoals voordien bij de setup van Minikube zal opnieuw gebruik gemaakt worden van Vagrant en een Vagrantfile om in dit geval meerdere Linux virtuele machines op te zetten.

Elke virtuele machine heeft minimum 2 GB RAM en 2 CPU's nodig om over voldoende resources te beschikken voor applicaties die nadien in de cluster actief zullen zijn. Ook moet swap uitgeschakeld worden op elke machine, om een goede werking van de kubelet te verzekeren. (Kubernetes, 2022b)

4.2.2 Opzetten van een multi-node cluster met kubeadm

Volgend stappenplan beschrijft eerst in grote lijnen de volledige setup van de multi-node clusteromgeving, die nadien in detail besproken zal worden:

1. Creëer drie virtuele machines die later onderdeel zullen uitmaken van de cluster.
2. Pas de Linux firewall aan zodat bridged netwerkverkeer correct verwerkt wordt.
3. Installeer een Container Runtime (vb. Docker) op elke virtuele machine.
4. Installeer de Kubeadm tool op elke virtuele machine.
5. Stel één virtuele machine in met de rol van master node.
6. Zet een Container Networking Interface (CNI) op om een netwerk tussen nodes/pods te voorzien.
7. Voeg de worker nodes toe aan de cluster.

Meerdere virtuele machines opzetten via Vagrant

Maak een nieuwe directory (met naam naar wens) op uw hostsysteem aan waar de configuratiebestanden van de virtuele machines zullen bewaard worden. De Vagrantfile die gebruikt wordt om de drie virtuele machines op te zetten is terug te vinden op de Github pa-

gina van Kodekloudhub. Open een Git Bash terminal in de zojuist gecreëerde directory en kloon de repository via commando **git clone <https://github.com/kodekloudhub/certified-kubernetes-administrator-course>**

Optioneel: men kan de inhoud van de Vagrantfile bekijken/aanpassen alvorens tot de installatie van de virtuele machines over te gaan.

Via het commando **vagrant status** kan u zien dat er drie virtuele machines zijn met de status 'not created'. Wanneer men vervolgens commando **vagrant up** uitvoert zal het proces starten om de drie virtuele machines op te zetten.

Na afloop van de installatie kan men nogmaals de status opvragen en zal men volgende output zien:

```
$ vagrant status
Current machine states:

kubemaster           running (virtualbox)
kubenode01           running (virtualbox)
kubenode02           running (virtualbox)
```

Wanneer men VirtualBox opent kan men eveneens zien dat deze drie virtuele machines operationeel zijn.

Controles uitvoeren en Linux firewall configureren

Men kan via m.b.v. een Git Bash terminal inloggen via SSH op elke virtuele machine via commando **vagrant ssh [nodenaam]**.

Let op: Wanneer men op een Windows host Powershell zou gebruiken om een SSH-verbinding te maken met deze virtuele machines kan dit leiden tot de foutmelding 'Permission denied', omdat Powershell eerst de native ssh.exe op het hostsysteem zal proberen aanspreken in plaats van het ingebouwde **vagrant ssh**.

Volgende stappen worden uitgevoerd op elke virtuele machine alvorens men kan overgaan tot het installeren van de kubectl tool:

1. Verifieer dat MAC-adres en product_uuid uniek zijn voor elke node.

```
# MAC-adres controle op enp0s8 interface
$ ip link OF ipconfig -a

# product_uuid controle
$ sudo cat /sys/class/dmi/id/product_uuid
```

2. Laad de module_netfilter zodat iptables bridged netwerkverkeer kan zien. Het programma iptables wordt gebruikt om regels te configureren m.b.v. verschillende Netfilter modules zodat bepaalde IP pakketten toegestaan/geblokkeerd worden op de Linux firewall.


```
# Controleer als de module br_netfilter ingeladen is
$ lsmod | grep br_netfilter
```

```
# Laad de module br_netfilter in
$ sudo modprobe br_netfilter
```

3. **Pas de 'sysctl' configuratie aan zodat iptables het bridged netwerkverkeer correct kan zien.** Gebruik volgende commando's om dit te bekomen:

```
$ cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
br_netfilter
EOF
```

```
$ cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
EOF
$ sudo sysctl --system
```

Een Container Runtime op elke VM installeren

Men heeft verschillende opties qua Container Runtime, maar in dit geval wordt gekozen voor Docker. Deze kan men installeren door gebruik te maken van het 'convenience script' die te vinden is op de officiële Docker webpagina. (Docker, 2021)

1. **Installeer Docker op de nodes m.b.v. het convenience script.**

```
$ curl -fsSL https://get.docker.com -o get-docker.sh
$ sudo sh get-docker.sh
```

2. **Controleer als Docker succesvol is geïnstalleerd.**

```
$ systemctl status docker
```

Kubeadm, kubelet en kubectl installeren + cgroup driver configureren

Volgende packages worden geïnstalleerd op alle virtuele machines:

- **kubeadm:** tool waarmee de cluster op het systeem wordt geïnstalleerd.
- **kubelet:** verantwoordelijk voor het opstarten van pods/containers.
- **kubectl:** cmdline-tool om te communiceren met de cluster.

Gebruik volgende commando's om deze drie te installeren op de nodes:

```
# Update apt package index en installeer nodige packages
$ sudo apt-get update
$ sudo apt-get install -y apt-transport-https ca-certificates curl

# Download de Google Cloud public signing key
```

```
$ sudo curl -fsSLo \
  /usr/share/keyrings/kubernetes-archive-keyring.gpg \
  https://packages.cloud.google.com/apt/doc/apt-key.gpg

# Voeg de Kubernetes apt repository toe
$ echo "deb \
  [signed-by=/usr/share/keyrings/kubernetes-archive-keyring.gpg] \
  https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee \
  /etc/apt/sources.list.d/kubernetes.list

# Update apt package index
$ sudo apt-get update

# Installeer kubelet, kubeadm en kubectl
$ sudo apt-get install -y kubelet kubeadm kubectl
$ sudo apt-mark hold kubelet kubeadm kubectl
```

Nota: De kubelet zal nu in een herstart-loop terecht komen aangezien het wacht op orders van kubeadm.

Het configureren van een 'cgroup driver' is eveneens van belang aangezien deze er voor zorgt dat de Container Runtime (= Docker) en de kubelet cgroup drivers overeenkomen. De cgroup driver voor zowel Docker als kubelet moeten beiden systemd zijn! Normaal gezien zou dit voor Docker die eerder geïnstalleerd is nog niet het geval zijn. Men kan dit controleren via het commando **docker info** als root uit te voeren. Daar zal men kunnen zien dat de cgroup driver momenteel nog 'cgroupfs' is. Om dit te veranderen naar 'systemd' voeren we volgend commando's uit op elke node:

```
$ echo '{"exec-opts": ["native.cgroupdriver=systemd"]}' \
  >> /etc/docker/daemon.json
$ systemctl restart docker
```

De master node configureren

In deze stap wordt een cluster opgezet op de master node. De stappen in deze procedure zijn afgeleid van de officiële Kubernetes documentatie en kan men in volgende bron raadplegen. (Kubernetes, 2022a)

Om een cluster op te starten gebruikt men commando **kubeadm init**, aangevuld met extra parameters om het POD-netwerk en het IP-adres van de Kubernetes API Server te specificeren. Dit is het IP-adres van de master node, en wordt gebruikt om in de cluster te adverteren wie de API Server is. Pas dit commando ENKEL toe op de master node:

```
$ kubeadm init --pod-network-cidr 10.244.0.0/16 \
  --apiserver-advertise-address=[IP-master node]
```

Het verwerken van dit commando zal even tijd nodig hebben. Als dit succesvol is zal

men onderaan in de output de melding 'Your Kubernetes control-plane has initialized successfully!' zien. Onder deze melding krijgt men ook enkele commando's te zien die noodzakelijk zijn om gebruik te kunnen maken van de cluster. Ondanks er nog meer commando's verder in de output te zien zijn is het voorlopig enkel nodig onderstaande uit te voeren:

```
$ mkdir -p $HOME/.kube
$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
$ sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

De resterende commando's die te zien zijn in de output worden uitgevoerd om worker nodes toe te voegen aan de cluster. Dit lukt pas nadat een Container Network Interface is geconfigureerd in volgend hoofdstuk. Bewaar de output voorlopig in een tekstbestand zodat men later kan terug grijpen naar de benodigde commando's.

Een Container Network Interface configureren

Een CNI opzetten doet men om een netwerk te voorzien waardoor pods/nodes onderling kunnen communiceren. Er zijn verschillende keuzes qua CNI, en de installatie verloopt steeds door een plugin te installeren. In het opzetten van een cluster via Kubeadm is gekozen voor de CNI plugin 'Weavenet'. (Weaveworks, 2022)

Voer volgend commando uit om de Weavenet CNI te installeren in de cluster:

```
$ kubectl apply -f \
"https://cloud.weave.works/k8s/net?\
k8s-version=$(kubectl version | base64 | tr -d '\n')"
```

De worker nodes toevoegen aan de cluster

De bewaarde output die men kreeg op het eind van sectie 'De master node configureren' kan men nu toepassen om de worker nodes toe te voegen aan de cluster. Men gebruikt hiervoor het **kubeadm join** commando, aangevuld met een token om authenticatie te voorzien. Voer onderstaand commando uit op de worker nodes zodat deze toegevoegd worden aan de cluster:

```
$ kubeadm join 192.168.56.2:6443 --token [TOKEN]
--discovery-token-ca-cert-hash
sha256:[HASH]
```

Indien de toevoeging succesvol is zal men onderaan de gegenereerde output de melding 'This node has joined the cluster' zien. Men kan dit bevestigen via commando **kubectl get nodes**.

Nota: Indien men de error krijgt 'invalid discovery token CA certificate hash' kan men nog steeds een andere token creëren en het proces herhalen. Volgende bron kan gebruikt worden om dit te bekomen. (Mukul, 2020)

4.2.3 Conclusie Kubeadm

Het opzetten van een multi-node cluster via Kubeadm is tijdrovend door de vele manuele stappen die ondernomen moeten worden op de verschillende nodes. Ondanks het een tijdrovende taak is geeft het wel enig inzicht in wat op het allemaal systeem moet gebeuren om een multi-node Kubernetes cluster operationeel te krijgen. Zo ziet men o.a. dat firewall regels moeten toegepast worden om het bridged netwerkverkeer mogelijk te maken, cgroup drivers correct geconfigureerd moeten worden, nodes aan de cluster toegevoegd worden o.b.v. tokens, een Container Network Interface (CNI) moet opgezet worden ...

Zoals eerder aangegeven is de Kubeadm tool ontworpen om toegepast te worden als een bouwblok in andere, completere tools. Zo gebruikt Kubespray, de tool die in volgend hoofdstuk besproken wordt, de Kubeadm tool om een Kubernetes multi-node cluster te creëren op een volledig geautomatiseerde manier m.b.v. een Ansible playbook.

4.3 Kubespray

Kubespray is een tool waarmee men het opzetten van Kubernetes clusters kan automatiseren m.b.v. Ansible. Er is nog steeds nood om vooraf een infrastructuur op te zetten waarop de cluster actief zal zijn, inclusief een Ansible provisioning node. Dit is het systeem waarop de Ansible playbook wordt uitgevoerd resulterend in een cluster met een aantal vooraf gedefinieerde control-plane nodes en worker nodes. Kubespray is een wrapper rond Kubeadm en voorziet een set van scripts en templates dat men kan gebruiken om een Kubernetes cluster op te zetten. (Matykevich, 2018)

Nota: De provisioning node maakt géén deel uit van de cluster en hoeft dus ook niet in dezelfde IP-range te zitten als de virtuele machines waarop de cluster actief is.

4.3.1 Vereisten

In dit onderzoek is gekozen om voor de Ansible provisioning node een Linux Debian Bullseye virtuele machine met 4 GB RAM en 2 CPU's te gebruiken. De drie nodes die deel zullen uitmaken van de cluster zijn Linux Ubuntu (bionic64) virtuele machines met 2 GB RAM en 2 CPU's. Volgens de officiële documentatie moet men aan een control-plane node minimum 1.5 GB RAM, en aan een worker node minimum 1 GB RAM toewijzen. Om deze setup snel te verkrijgen is opnieuw gebruik gemaakt van Vagrant en een Vagrantfile.

Nota: Men kan indien de resources op het host systeem beperkt zijn ervoor kiezen om in de Vagrantfile de toegewezen resources van de nodes te verlagen tot het minimum.

Alle vereiste packages die nodig zijn op de Ansible provisioning node, vb. de specifieke versies van packages zoals Ansible, Jinja, python-netaddr ..., kan men terugvinden in het 'requirements.txt' bestand in de kubespray repository. (Kubespray, 2022b) Het klonen

van de kubespray repository en de installatie van de benodigde packages komt aan bod tijdens de setup van de Ansible provisioning node in hoofdstuk 4.3.2.

Let op wanneer men gebruik zou maken van een virtuele machine waar alreeds Ansible op geïnstalleerd is. Dit kan als gevolg hebben dat de packages die geïnstalleerd worden uit het 'requirements.txt' bestand in een verkeerde directory terecht komen, wat mogelijk kan leiden tot een falende uitvoer van de playbook (Kubespray, 2022a)

4.3.2 Een multi-node cluster opzetten met Kubespray

Net zoals in voorgaand beschreven tools dient men eerst zelf een infrastructuur op te zetten. Dit wordt opnieuw via Vagrant gedaan. Volgend stappenplan beschrijft hoe men deze infrastructuur kan bekomen en kan inloggen op de Ansible provisioning node.

Standaard is Kubespray geconfigureerd om twee control-plane nodes te creëren. Aangezien in deze setup slechts drie virtuele machines gebruikt zullen worden zal in het stappenplan beschreven staan hoe men dit in de Kubespray configuratie kan aanpassen. Wil men toch een cluster opzetten met twee control-plane nodes, zorg dan voor een extra virtuele machine door de Vagrantfile aan te passen, zodat eveneens twee worker nodes bestaan in de cluster.

1. Gebruik de Vagrantfile in volgende link en plaats deze in een nieuwe directory genaamd 'kubespray':
Open een Git Bash terminal in deze directory en voer het commando **vagrant up** uit. Hierdoor worden de vier virtuele machines gecreëerd: 1 provisioning node + 3 cluster nodes.
2. Maak een SSH-verbinding met de Ansible provisioning node via commando **vagrant ssh provisioningnode**.

Eens ingelogd op de provisioning node moet men o.a. de lijst van beschikbare packages in de repositories updaten, de kubespray Github repository op het systeem klonen, nodige packages installeren ... Volg volgende stappen om de provisioning node klaar te maken om de cluster te creëren:

```
# packages updaten en Git installeren
$ sudo apt update
$ sudo apt install git -y

# Kubespray repository klonen op het systeem
$ git clone https://github.com/kubernetes-sigs/kubespray.git

# Python-pip3 installeren
$ sudo apt-get install python3-pip -y

# Ga naar directory kubespray + installeer benodigde packages
$ cd kubespray && sudo pip3 install -r requirements.txt
```

```
# Kopiëer directory inventory/sample naar inventory/mycluster
$ cp -rfp inventory/sample inventory/mycluster
```

Nota: Indien de installatie van de benodigde packages faalt kan men proberen een oudere versie van requirements.txt aan te spreken vb. requirements-2.11.txt. Voer commando **ls** uit in de kubescape directory om alle mogelijkheden op te lijsten.

Nu alle nodige packages geïnstalleerd zijn kan men een SSH-keypair aanmaken op de provisioning node. De public key zal vervolgens toegevoegd worden op elke overige virtuele machine waarop de cluster actief zal zijn. Voer volgende stappen uit op de provisioning node om dit te realiseren:

```
# SSH-keypair maken (alles default laten)
$ ssh-keygen

# Output toont directory waar public key bewaard wordt.
# Open dit bestand en kopieer de inhoud.
$ cat /home/vagrant/.ssh/id_rsa.pub

# Open nieuwe Git Bash terminal en maak SSH-verbinding met
# resterende virtuele machine(s)
$ vagrant ssh [master1, worker1, worker2]

# Ga naar de .ssh directory en open het authorized_keys
# bestand met een texteditor naar keuze.
# Plaats daar de public key van de provisioning node en
# sla vervolgens op.
# Herhaal dit proces op alle nodes.
$ nano .ssh/authorized_keys
```

Op dit moment kan de Ansible provisioning node connecteren via SSH met de (toekomstige) cluster nodes. In volgende stap creëert men via de inventory_builder automatisch een inventory file genaamd hosts.yaml. Het enige wat vooraf nodig is zijn de IP-adressen van de clusternodes. Om deze snel op te vragen kan men volgend commando uitvoeren in een nieuwe Git Bash terminal:

```
for f in $(VBoxManage list runningvms | awk -F\" '{print $2}');
do
echo "$f:"
VBoxManage guestproperty enumerate "$f" | grep IP
done
```

Keer terug naar de terminal sessie op de provisioning node. Gebruik hier de IP-adressen verkregen in vorige output om via volgend commando's de inventory file te creëren. Eerst zal men nog een kleine aanpassing moeten uitvoeren aan het bestand 'contrib/inventory_builder/inventory.py' om ervoor te zorgen dat slechts één control-plane node geconfigureerd wordt in de inventory file. Standaard zal dit namelijk op twee staan, maar deze

vorm van high availability is voor deze testdoeleinden niet noodzakelijk. Zoek volgende regel in het inventory.py bestand en pas de waarde 2 aan naar 1 en sla op.

```
# Remove the reference of KUBE_MASTERS after ...
KUBE_CONTROL_HOSTS = int(os.environ.get("KUBE_CONTROL_HOSTS",
os.environ.get("KUBE_MASTERS", 2)))
```

Vervolgens kan men overgaan tot het creëren van de inventory file:

```
# Een inventory file hosts.yaml maken
$ declare -a IPS=(10.10.2.2 10.10.2.3 10.10.2.4)
$ CONFIG_FILE=inventory/mycluster/hosts.yaml python3 \
contrib/inventory_builder/inventory.py ${IPS[@]}

# (Optioneel) Controleer de zojuist gecreëerde inventory file
$ cat inventory/mycluster/hosts.yaml
```

Nu kan men overgaan tot het installeren van de Kubernetes cluster op de nodes geconfigureerd in de inventory file. Dit geautomatiseerd proces zal, afhankelijk van de toegewezen resources, ongeveer een half uur tijd nodig hebben om alles op te zetten. Gebruik volgend commando om de Ansible playbook uit te voeren:

```
ansible-playbook -i inventory/mycluster/hosts.yaml \
--become --become-user=root cluster.yml
```

Eens de clusterconfiguratie afgerond is dient men eerst nog volgende commando's uit te voeren op de control-plane node, alvorens men kan communiceren met de cluster via de kubectl commandline tool.

```
# Connecteer via SSH naar de control-plane node
$ vagrant ssh master1

# Kopieer kubectl configuratiebestanden naar home directory
$ mkdir -p $HOME/.kube
$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
$ sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

De drie virtuele machines horen nu in een Kubernetes cluster. Dit kan men controleren door op de control-plane node het commando **kubectl get nodes** uit te voeren. Men zal zien dat de nodenamen node1, node2 en node3 gebruikt worden i.p.v. de hostnames van de virtuele machines.

Zoals men uit de output van dit commando ook kan afleiden maakt Kubespray standaard gebruik van de Container Runtime containerd. Deze Container Runtime wordt ook in gebruik genomen door Docker. Dit is relevant voor later in bepaalde chaos experimenten, waar het vergeten definiëren van de correcte Container Runtime tot problemen kan leiden bij de uitvoer.

In deze cluster zal ook alreeds de CNI Calico geïnstalleerd zijn, zodat netwerkcommunicatie tussen pods/nodes correct kan verlopen. Men heeft ook de mogelijkheid om via de Ansible playbook een andere CNI te installeren, volgens de documentatie op de Kubespray website.

Wanneer ter illustratie een simpele Deployment gelanceerd wordt op de master node o.b.v. de Nginx image, en waarin 3 replicas aanwezig moeten zijn, dan kan men zien via commando **kubectl get pods -o wide** dat deze gelijk verdeeld werden over de verschillende nodes:

```
$ kubectl get pods -o wide
```

NAME	READY	STATUS	IP	NODE
nginx-85b98978db-4t792	1/1	Running	10.233.96.3	node2
nginx-85b98978db-78gcd	1/1	Running	10.233.90.5	node1
nginx-85b98978db-n9tb2	1/1	Running	10.233.92.1	node3

De default configuratie van Kubespray heeft een cluster opgezet waarbij de Scheduler applicatie-pods kan onderbrengen op de control-plane node. Om de resources van de control-plane node onder controle te houden, maar ook eveneens om veiligheidsredenen, is het echter afgeraden dit in een cluster in productie toe te passen. Meer info over dit onderwerp kan men in de bron raadplegen. (Bailey, 2016)

4.3.3 Conclusie Kubespray

De manier hoe met Kubespray een lokale Kubernetes cluster werd opgezet maakt zowel gebruik van de automatisatietools Vagrant als Ansible. Dit heeft het voordeel dat veel van de tijdrovende taken die in voorgaande tool Kubeadm aan bod kwamen alreeds geautomatiseerd zijn. Zo is elke node van de multi-node cluster alreeds toegevoegd aan de cluster, is er alreeds een CNI aanwezig, is de Container Runtime containerd op elke node geïnstalleerd ... De Ansible playbook had wel bijna een half uur nodig om deze cluster tot stand te brengen. Hierdoor kan men nog steeds concluderen dat een lokale cluster opzetten een tijdrovende taak is ondanks het wel het nodige inzicht met zich meebrengt welke zaken nodig zijn om een Kubernetes cluster operationeel te krijgen.

De procedure op de Kubespray website om een cluster tot stand te brengen is vrij summier en mocht duidelijker omschreven zijn. Het uitvoeren van de Ansible playbook bracht een cluster tot stand waarin de control plane node eveneens user pods toegewezen kreeg. Hoe dit precies kon aangepast worden via de playbook was eveneens onduidelijk gedocumenteerd.

Het opzetten van een monitoring platform voor het opvolgen van experimenten die het geheugen of CPU belasten zou eveneens nog moeten gebeuren in deze lokale omgeving, maar is vanwege de tijdsdruk niet verder onderzocht.

In volgend hoofdstuk zal een Kubernetes cluster via Google Cloud opgezet worden, om enerzijds te vergelijken als de setup sneller verloopt en anderzijds de extra functionaliteit in kaart te brengen die deze omgeving met zich meebrengt.

5. Cloud Kubernetes clusters

5.1 Google Cloud Platform

In het voorgaande hoofdstuk 4 lag de focus op het in stand brengen van een Kubernetes cluster in een lokale omgeving. Initieel werd de voorkeur gegeven aan een lokale omgeving omdat deze dichter aanleunt bij de manier hoe men in schoolverband een virtuele omgeving tot stand brengt, vaak met behulp van automatisatie. Ondanks de automatisatie bleek dit nog steeds een tijdrovende taak te zijn.

In dit hoofdstuk zal besproken worden hoe men een Kubernetes cluster kan opzetten via Google Cloud. Hiervoor zal Google Kubernetes Engine (GKE) gebruikt worden, een Kubernetes beheer- en orkestratieplatform van Google. GKE voorziet een beheerde omgeving voor het uitrollen, beheren en schalen van gecontaineriseerde applicaties.

Na de installatieprocedure zal ook kort ingegaan worden op de extra functionaliteit die het opzetten van een GKE cluster met zich meebrengt.

5.1.1 Opzetten van een GKE cluster via Google Cloud

Om een cluster op te zetten via Google Cloud doet men beroep op Google Kubernetes Engine (GKE). Nieuwe klanten kunnen GKE eerst uittesten door een account te creëren waarbij een gratis krediet van driehonderd dollar gegeven wordt, geldig voor een periode van negentig dagen. Men zal bij de creatie van een account een kredietkaart of andere betalingswijze moeten opgeven, om een Cloud Billing account op te zetten en de identiteit te bevestigen. Indien het gratis krediet eerder op is of de negentig dagen zijn verstreken, dan zal men uitdrukkelijk moeten upgraden naar een betaalde Cloud Billing account om

verder gebruik te kunnen maken van de diensten die GKE aanbiedt. (Google-Cloud, 2022)

Via volgende stappenplan kan men een account op GKE creëren en een cluster opzetten:

1. Ga naar GKE via volgende URL: <https://cloud.google.com/kubernetes-engine>
2. Vul uw gegevens in + activeer uw account.
3. Kies voor de optie **Enable** bij de Kubernetes Engine API en wacht vervolgens één a twee minuten tot het startscherm van het Google Cloud Platform tevoorschijn komt. Bovenaan deze pagina zal u het huidige saldo en de resterende dagen steeds kunnen opvolgen.
4. Centraal op deze pagina ziet men **Kubernetes Engine - Kubernetes Clusters**. Kies hier eerst voor de optie **Create**, en vervolgens de optie **GKE Standard** om verder te gaan.
5. De cluster die aan de hand van dit stappenplan wordt opgezet zal bestaan uit drie nodes. Voldoende resources toewijzen aan de cluster is cruciaal, aangezien men later zowel pods zal installeren toebehorend aan een chaos engineering tool als twee demo-applicaties. **Cluster Autoscaling** zal ook toegepast worden op de cluster en zal o.b.v. de vraag naar resources extra nodes toevoegen of nodes afbouwen. (GKE, 2022)

Nota: Hierdoor is het mogelijk dat in het verloop van de experimenten er meer of minder dan drie nodes actief zullen zijn.

Volgende configuraties kan men toepassen om de cluster op te zetten:

- (a) **Een Machine Type definiëren:** Ga in het linkermenu eerst naar **Node pools** en vervolgens onder **default-pool** naar het tabblad **Nodes**. Een goede richtlijn is per node 2 vCPU en 4GB RAM toe te wijzen. Dit kan men bekomen door in de dropdownlist **Machine Type** te kiezen voor **e2-medium (2vCPU, 4 GB memory)**.
 - (b) **Enable cluster autoscaler activeren:** Vink deze optie aan om er voor te zorgen dat GKE de nodes in de cluster automatisch zal schalen.
 - (c) Klik onderaan op **Create** om de cluster te creëren. Dit proces zal ongeveer één a twee minuten tijd nodig hebben om af te ronden. Eens de cluster operationeel is zal men naast de naam bij Status een groene vink zien staan. Men bevindt zich nu in het Kubernetes Engine menu in het Google Cloud Platform. Dit menu kan u terugvinden onder het dropdownmenu in de linkerbovenhoek (naast Google Cloud Platform).
6. Klik op de naam van de cluster om het overzicht te openen. Daar ziet men algemene configuraties van deze cluster.
 7. Om een terminalsessie in de cluster te openen kiest men rechtsbovenaan in de taakbalk voor de optie **Connect**. Een nieuw scherm **Connect to the cluster** zal hierdoor geopend worden.
 8. Kies voor de optie **Run in Cloud Shell**. Dit zal automatisch het bovenstaande commando meenemen naar een nieuwe terminalsessie. Daar kan men vervolgens het commando uitvoeren.
 9. Een nieuw scherm **Authorize Cloud Shell** wordt hierdoor geopend, waar men via de optie **Authorize** de toegang krijgt tot de cluster.

Op een identieke manier als bij het opzetten van de lokale Kubernetes clusters, kan men

hier ook snel een eerste demo-applicatie lanceren om de verdeling van de pods over de nodes te controleren. Gebruik hiervoor commando **kubectl create deploy nginx –image=nginx –replicas=3**.

Uit de output van het commando **kubectl get pods -o wide** kan men zien dat de drie applicatie-pods verdeeld zijn over de drie beschikbare nodes. Dit was eveneens het geval bij de laatste lokale cluster setup via Kubespray, maar met het fundamentele verschil dat in deze cloudomgeving de control-plane node beheerd wordt door Google zelf, waardoor bij de creatie van de cluster enkel worker nodes in de node pool aanwezig zijn. Men kan via commando **kubectl cluster-info** informatie omtrent de control plane node opvragen.

5.1.2 Extra functionaliteit in GKE

De Deployments en pods van de demo-applicaties, alsook enkele gemonitorde metrics zoals CPU- en geheugengebruik, kan men opvolgen door in het linkermenu naar Workloads te gaan. Wanneer men een Deployment selecteert in dit menu kan men bovenaan in de menubalk enkele handige opties zien die het toelaten om configuraties rechtstreeks vanuit de browser uit te voeren i.p.v. via de terminal. De opties hier zijn o.a.:

- **Edit:** De YAML-configuratie van de Deployment aanpassen.
- **Delete:** De Deployment verwijderen.
- **Actions:**
 - Autoscale:** Een HorizontalPodAutoscaler configureren. (zie Hoofdstuk 6.3.7)
 - Expose:** Een Service creëren.
 - Rolling Update:** De versie van de container image vernieuwen.
 - Scale:** Instellen hoeveel pods de ReplicaSet van de Deployment actief moet houden. Men kan hier eveneens een limiet instellen op de resources die pods in de Deployment mogen gebruiken.
- **Kubectl:** Het benodigde commando om de YAML-configuratie van de Deployment op te vragen rechtstreeks inladen in de Cloud Shell terminal.

Nota: Ondanks de aanwezigheid van deze functionaliteit zal gedurende dit onderzoek voornamelijk vanuit de Cloud Shell terminal gewerkt worden om soortgelijke configuraties te bekomen. De reden hiervoor is om bekend te raken met de commando's waarmee Kubernetes objecten geconfigureerd worden.

GKE monitoring

In het Workloads menu kan men alreeds enkele metrics opvolgen, maar Google Kubernetes Engine (GKE) heeft ook een volwaardig monitoring platform. Wanneer men in GKE in het Clusters menu de cluster selecteert gaat men naar tabblad **Details**. Scroll in dit overzicht tot aan de sectie **Features** waar men in de lijst **Cloud Monitoring** kan terugvinden. Men kan naar het monitoring platform gaan door te klikken op de link **View GKE Dashboard**.

In de pagina die geopend wordt kan men heel wat functionaliteit opmerken. In het linker-

menu ziet men **Metrics Explorer**. Via deze kan men verschillende metrics opvragen van de nodes in de cluster. Dit zal later van pas komen in het chaos engineering experiment waarbij node resources zoals geheugen/CPU belast worden.

Nota: Een alternatief pad naar dit platform kan ook genomen worden door in één van de opgevolgde metrics in het Workloads menu de optie rechtsbovenaan 'More chart options' te selecteren en vervolgens voor 'View in Metrics Explorer' te kiezen.

Voor het opvolgen van andere experimenten zal de terminal-gebaseerde UI **k9s** gebruikt worden, waarvan de setup alreeds besproken is in Hoofdstuk 2.5.9.

5.1.3 Conclusie GKE cluster

Het opzetten van een GKE cluster via Google Cloud is heel eenvoudig en verloopt veel sneller ten opzichte van een lokale Kubernetes setup op te zetten via de eerder besproken opties in dit onderzoek. Ondanks een cluster opzetten in GKE niet gratis is kan men deze omgeving wel eerst uittesten via het gratis krediet van 300 dollar die men krijgt bij de creatie van een account. Dit krediet is geldig gedurende negentig dagen waardoor dit platform in aanmerking komt om te gebruiken voor educatieve doeleinden. Het enige nadeel om een account te bekomen is dat een betaalwijze moet meegegeven worden, ondanks men wel uitdrukkelijk verduidelijkt dat géén bedrag zal gefactureerd worden zonder men eerst aangeeft over te gaan naar een betalend abonnement.

GKE biedt veel functionaliteit, waaronder de mogelijkheid tot het automatisch schalen van nodes via de autoscaling-optie, de mogelijkheid om configuraties vanuit de browser uit te voeren, de aanwezigheid van een monitoring platform om metrics op te volgen ... Dit is echter nog maar een kleine greep uit het enorme aanbod waardoor men wel de nodige tijd moet investeren om wegwijs te raken in GKE.

6. Chaos Engineering Tools

Het toepassen van chaos engineering is relatief nieuw, maar er zijn ondertussen al heel wat tools op de markt verschenen om chaos experimenten te kunnen uitvoeren. De Cloud Native Computing Foundation (CNCF), een Linux Foundation project gestart in 2015, is de thuisbasis van heel wat open source projecten die vandaag het landschap definiëren in de IT-sector. CNCF projecten kunnen drie niveaus van maturiteit hebben nl. sandbox, incubating en graduated. Deze niveau's geven aan hoe ver een project geëvolueerd is. (CNCF, 2022b)

Men kan een overzicht van deze projecten, gerangschikt per categorie, terugvinden in het CNCF Cloud Native Interactive Landscape. Eén van deze categorieën is chaos engineering, waar men een lijst kan terugvinden van chaos engineering tools. (CloudNative-Landscape, 2022)

Pavlos Ratis, Site Reliability Engineer bij RedHat OpenShift, onderhoudt een Github repository waar men heel wat zaken omtrent chaos engineering kan terugvinden. In deze repository vindt men in de sectie 'Notable Tools' een lijst van chaos engineering tools die bruikbaar zijn voor verschillende doeleinden. Deze lijst werd eveneens geraadpleegd in de zoektocht naar een geschikte tool om chaos engineering experimenten toe te passen op een Kubernetes cluster. (Ratis, 2022).

De eerste tool die onderzocht werd is Chaos Toolkit, via een cursus op het online leerplatform Udemy. Nadien kwamen uit de eerder vernoemde bronnen nog twee andere tools naar boven genaamd Chaos Mesh en Litmus.

Vooraleer men experimenten kan beginnen uitvoeren heeft men eerst een demo-applicatie nodig. Volgend hoofdstuk beschrijft hoe men twee demo-applicaties kan opzetten waar men later experimenten op kan toepassen. Daarna komt de installatie van de verschillende

chaos engineering tools en het uitvoeren van de experimenten aan bod.

6.1 Demo-applicaties opzetten

Aangezien dit onderzoek gericht is om experimenten uit te voeren voor educatieve doeleinden, is gekozen om eerst enkele experimenten uit te voeren op Nginx en Apache web-server pods. Door twee aparte Deployments en Services op te zetten kan de onderlinge communicatie tussen beiden aangetoond en getest worden. Dit is praktisch om experimenten uit te voeren waarbij de communicatie verstoord wordt tussen verschillende Services, of experimenten enkel te richten op specifieke pods in een namespace.

Het visuele aspect ontbreekt hierbij echter nog om de impact van een experiment te verduidelijken op de applicatie in een browser. Vandaar is gekozen om enkele experimenten te herhalen op een tweede demo-applicatie genaamd PodTato-Head, een applicatie die een aardappelmanneltje toont en waarbij de lichaamsdelen bestaan uit verschillende pods, opgezet via afzonderlijke Deployments en Services. Met behulp van deze applicatie kan o.a. aangetoond worden dat een applicatie nog steeds bereikbaar kan zijn desondanks bepaalde pods getroffen worden.

Men kan alle experimenten die hieronder beschreven staan ook toepassen op de PodTato-Head applicatie door simpelweg de namespace aan te passen waar nodig naar 'demoapp2'.

6.1.1 Demo-applicatie 1:Nginx/Apache webserver pods

De eerste demo-applicatie kan men onderbrengen in een aparte namespace genaamd 'demoapp1'. Op deze manier kan men de impact van de experimenten (= de blast radius) beperken en voorkomen dat andere pods ongewenst mee betrokken worden in een experiment. Om de eerste demo-applicatie tot stand te brengen voert men volgende commando's uit:

```
# Creëer een nieuwe namespace voor de experimenten
$ kubectl create ns demoapp1

# Creëer een Deployment met 3 Apache pods
# in de namespace litmusexperiments
$ kubectl create deploy apache --image=bitnami/apache \
--replicas=3 -n demoapp1

# Creëer een Service voor de Apache pods
# Apache luistert in container op poort 8080
$ kubectl expose deploy apache --port=80 \
--target-port=8080 -n demoapp1

# Creëer een Deployment met 3 Nginx pods
# in de namespace demoapp1
```

```
$ kubectl create deploy nginx --image=nginx \
--replicas=3 -n demoapp1

# Creëer een LoadBalancer Service voor de Nginx pods
$ kubectl expose deploy nginx --port=80 --type=LoadBalancer \
-n demoapp1
```

Wanneer men vervolgens via commando **kubectl get svc -n demoapp1** de Services opvraagt zal men een extern IP-adres bij de Nginx LoadBalancer Service zien. Via de browser kan men nu surfen naar dit IP-adres en zal men de nginx default webpagina te zien krijgen.

6.1.2 Demo-applicatie 2: PodTato-Head

De tweede demo-applicatie PodTato-Head kan men onderbrengen in een aparte namespace genaamd 'demoapp2'. De manier hoe deze geïnstalleerd wordt is opnieuw via het eerder gebruikte Helm, alhoewel men ook o.a. kubectl kan gebruiken om deze te installeren. (Gavant, 2022)

Met behulp van volgende commando's kan men de Podtato-Head applicatie installeren op het systeem:

```
# Creëer de namespace demoapp2
$ kubectl create ns demoapp2

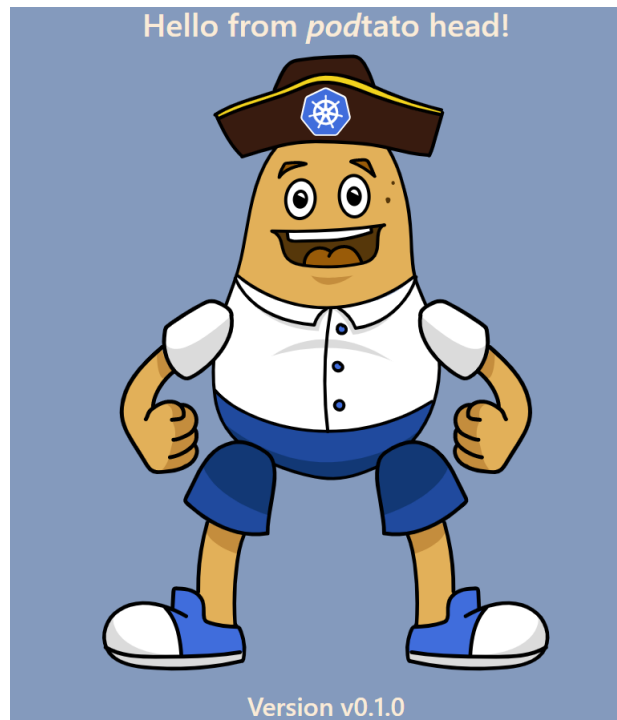
# Kopieer de Podtato-Head repository op het systeem
$ git clone https://github.com/podtato-head/podtato-head.git

# Ga verder naar de podtato-head directory
$ cd podtato-head/

# Installeer de podtato-head app in namespace demoapp2
$ helm install podtato-head ./delivery/chart -n demoapp2

# Verifieer als de app pods + services operationeel zijn
$ kubectl get pods -n demoapp2
$ kubectl get svc -n demoapp2
```

Via de output van het **helm install ...** commando ziet men enkele commando's hoe de URL verkregen kan worden om toegang tot de applicatie te bekomen via de browser. Zie figuur 6.1 als voorbeeld hoe de applicatie er in de browser uitziet.



Figuur 6.1: Podtato-Head applicatie

6.2 Chaos Toolkit

De eerst besproken chaos engineering tool in dit onderzoek is Chaos Toolkit. Deze tool kwam aan bod in de Udemmy cursus 'Kubernetes Chaos Engineering With Chaos Toolkit and Istio' en was volgens **Viktor Farcic**, de lesgever en auteur van het gelijkaardig genaamd boek, de beste chaos engineering tool op dat moment (maart 2020) beschikbaar.

Er zullen in dit onderzoek enkele eenvoudige experimenten aan bod komen, zowel uitgevoerd op de applicatie in namespace demoapp1 als demoapp2, die de verschillende componenten van Chaos Toolkit en de algemene werking ervan zullen omschrijven. Vooraleer hiermee van start te gaan zal eerst beschreven worden hoe deze tool geïnstalleerd wordt.

6.2.1 Installatie Chaos Toolkit

De enige vereiste wanneer men ChaosToolkit wil installeren is dat Python alreeds op het systeem aanwezig is. (ChaosToolkit, 2022d)

Voor een Linux Debian/Ubuntu distributie kan men volgend commando gebruiken om de nodige packages te installeren: **sudo apt-get install python3 python3-venv**

ChaosToolkit zal geïnstalleerd worden in een Python virtuele omgeving. Dit is een geïsoleerde omgeving die toelaat om afzonderlijke afhankelijkheden te beheren voor verschillende Python projecten. (UniversityOfMinnesota, 2022)

Gebruik volgend stappenplan om Chaos Toolkit te installeren in de Google Cloud Shell. Dezelfde werkwijze kan men ook toepassen in een lokale omgeving.

```
# Maak een virtuele omgeving aan
$ python3 -m venv ~/.venvs/chaostk

# Activeer deze virtuele omgeving
$ source ~/.venvs/chaostk/bin/activate

# Installeer de CLI
$ pip install -U chaostoolkit

# Verifieer succesvolle installatie CLI
$ chaos --version
```

Chaos Toolkit is nu geïnstalleerd in de virtuele omgeving op het systeem. Men zal links naast de prompt nu (**chaostk**) zien staan.

Let op: Het beëindigen van de sessie door de terminal te sluiten zal eveneens resulteren in het verlaten van de virtuele omgeving. Wanneer men opnieuw naar de virtuele omgeving wil gaan dient men het commando **source ~/.venvs/chaostk/bin/activate** nogmaals uit te voeren. Vergeet men dit echter te doen dan zal het 'chaos' commando niet gevonden worden wanneer men dit in de terminal probeert uit te voeren, aangezien dit commando enkel in de geïsoleerde virtuele omgeving bestaat.

Chaos Toolkit bevat na installatie voorlopig weinig functionaliteit. Men kan namelijk op dit moment nog geen experimenten opzetten die toepasbaar zijn op een Kubernetes cluster. Om dit mogelijk te maken dient men de **chaostoolkit-kubernetes** plugin te installeren. (ChaosToolkit, 2022b). Via volgende stappen kan men eerst de chaostoolkit-kubernetes plugin installeren op het systeem en vervolgens de verschillende experimentconfiguraties opvragen die deze plugin bevat.

```
# Installeer chaostoolkit-kubernetes plugin
$ pip install chaostoolkit-kubernetes

# Creëer bestand 'discovery.json' met overzicht qua opties
$ chaos discover chaostoolkit-kubernetes

# Ontdek alle opties voor experimenten in Kubernetes
$ cat discovery.json
```

Andere extensies/plugins waarmee men de functionaliteit van Chaos Toolkit verder kan uitbreiden kan men via volgende link raadplegen: Repository Chaos Toolkit extensies. Via deze extensies/plugins is het o.a. mogelijk experimenten op te zetten m.b.t. specifieke cloudarchitecturen en platformen, netwerkgerelateerde experimenten uit te voeren ...

6.2.2 Een experiment opzetten

Chaos Toolkit is CLI-gebaseerd, wat wil zeggen dat experimenten enkel via de terminal uitgevoerd kunnen worden. Voorgedefinieerde experimenten worden niet aangeboden bij deze tool, maar de bedoeling is om de experimenten modulair op te bouwen door de afzonderlijke componenten hieronder beschreven in een YAML- of JSON-bestand te definiëren (ChaosToolkit, 2022a):

1. **Beschrijving experiment:** versie, titel, tags.
2. **De 'steady-state-hypothesis':** De hypothese waarin men een controle (= probe) definieert om het normale gedrag (= de steady state) te valideren.
3. **Een 'method':** De activiteiten van een chaos experiment. Deze kunnen acties en probes bevatten die omschrijven hoe het experiment moet uitgevoerd worden.
4. **Een 'rollback':** Het chaos experiment ongedaan maken en terugkeren naar een normale staat. Nota: Een Kubernetes cluster bevat alreeds enkele zelfhelende eigenschappen waardoor een rollback configureren hierbij soms onnodig is.

Men kan zelf kiezen om deze YAML- of JSON-definitie te creëren, maar via commando **chaos init** kan men ook een experiment opzetten waarbij de verschillende stappen in het proces met begeleiding van een wizard afgerond kunnen worden.

Experimenten die via de wizard opgezet worden leiden tot de creatie van een **experiment.json** bestand. Wanneer men op dezelfde manier een nieuw experiment wil opzetten zal dit JSON-bestand overschreven worden. Men kan dit echter voorkomen door zelf een naam mee te geven in dit commando. Eveneens heeft men de optie om als output een YAML-bestand te bekomen. Zowel de naam van het experiment als de keuze qua configuratietaal bekomt men via commando **chaos init --experiment-path [experiment naam].[json | yaml]**.

Er zal ook een **journal.json** bestand gecreëerd worden na de uitvoer van een experiment opgezet via het 'chaos init' commando. Dit toont o.a. de definitie van het uitgevoerde experiment, welke pods getroffen zijn, wanneer en hoelang het experiment uitgevoerd is ... Alleen het laatst uitgevoerde experiment wordt bewaard in dit bestand m.a.w. ook dit wordt overschreven bij elke uitvoer van een nieuw experiment.

Zowel de setup m.b.v. de wizard via het **chaos init** commando, alsook het zelf creëren van experiment-definities zullen in komende experimenten beschreven worden. Vooraleer men van start gaat met het experimenteren maakt men een nieuwe directory aan waarin deze experimenten zullen bewaard worden genaamd 'chaostoolkit-experiments' m.b.v. commando **mkdir chaostoolkit-experiments**.

6.2.3 Experiment 1: Een random pod vernietigen

In dit experiment zal een willekeurige nginx pod van de demoapplicatie in namespace **demoapp1** vernietigd worden. De bedoeling is om te zien hoe het systeem zal reageren wanneer een pod vernietigd wordt. Aangezien de pods in de **demoapp1** namespace via een

Deployment zijn opgezet, zou het vernietigen van een pod steeds de ReplicaSet moeten triggeren om een nieuwe pod te creëren.

Zet het experiment op volgende manier op:

1. Ga naar de chaostoolkit-experiments directory en voer commando **chaos init** uit.
2. Geef een naam aan het experiment vb. random-pod-kill.
3. Geef deze keer géén steady-state hypothese op. Dit zal in volgende experimenten aan bod komen.
4. Antwoord met 'y' op de vraag 'Do you want to define an experimental method?' Dit zal een lijst weergeven met alle mogelijke acties die uitgevoerd kunnen worden.
5. Kies de actie 'terminate_pods' in deze lijst en bevestig de keuze met 'y'.
6. Vervolgens zal men enkele parameters moeten opgeven voor deze actie:
 - (a) **label_selector**: geef hier 'app=nginx' in om het experiment op de nginx te richten.
 - (b) **name_pattern**: via enter kiest men de defaultwaarde 'null'.
 - (c) **all**: via enter kiest men de defaultwaarde 'False' om aan te geven dat niet alle pods moeten vernietigd worden.
 - (d) **rand**: geef 'True' in, om een willekeurige pod te selecteren.
 - (e) **mode**: via enter kiest men de defaultwaarde 'Fixed'.
 - (f) **qty**: via enter kiest men de defaultwaarde '1' (= het aantal pods die vernietigd moeten worden).
 - (g) **grace_period**: via enter kiest men de defaultwaarde '-1'.
 - (h) **ns**: geef hier de namespace 'demoapp1' op.
 - (i) **order**: via enter kiest men de defaultwaarde 'Alphabetic'.
7. Antwoord met 'N' op de vraag 'Do you want to select another activity?' om aan te geven dat geen extra acties meer ondernomen moeten worden in dit experiment.

De wizard is afgelopen en het bestand 'experiment.json' zal vervolgens gecreëerd worden in deze directory. Men kan dit experiment uitvoeren via commando **chaos run experiment.json** De uitvoer van het experiment in onderstaande output toont aan dat dit succesvol verlopen is:

```
$ chaos run experiment.json
[INFO] Validating the experiment's syntax
[INFO] Experiment looks valid
[INFO] Running experiment: random-pod-kill
[INFO] Steady-state strategy: default
[INFO] Rollbacks strategy: default
[INFO] No steady state hypothesis defined.
[INFO] Playing your experiment's method now...
[INFO] Action: terminate_pods
[INFO] Let's rollback...
[INFO] No declared rollbacks, let's move on.
[INFO] Experiment ended with status: completed
```

Men ziet in de uitvoer dat dit experiment geen steady-state probeert te bevestigen of een

rollback uitvoert, maar louter één actie 'terminate_pods' doorloopt. Indien bovenstaande regels enkel groen gekleurd zijn kan men concluderen dat het experiment zonder fouten doorlopen is. De kleur van deze regels kan ook oranje of rood zijn bij afwijkingen tijdens de uitvoer van het experiment, maar dit zal vaker voorkomen bij het valideren van de steady state bv. door een probe die faalt.

Via het commando **cat journal.json** kan men de details opvragen van het uitgevoerde experiment zoals de definitie van het experiment, de start- en eindtijd, de getroffen pod, de status ...

Tijdens de uitvoer van dit experiment kan men via commando **k9s -n demoapp1** zien hoe één van de nginx pods in de namespace demoapp1 vernietigd wordt (zie kolom Status) en hoe direct daarna een nieuwe pod gecreëerd wordt om terug tot drie actieve pods te komen.

NAME↑	PF	READY	RESTARTS	STATUS
apache-5bcb8b58cc-2wlxf	●	1/1	0	Running
apache-5bcb8b58cc-b8hft	●	1/1	0	Running
apache-5bcb8b58cc-j5p4t	●	1/1	0	Running
nginx-6799fc88d8-9xkkm	●	1/1	0	Running
nginx-6799fc88d8-822h9	●	0/1	0	ContainerCreating
nginx-6799fc88d8-g26ld	●	1/1	0	Running
nginx-6799fc88d8-sr9bp	●	0/1Δ	0	Terminating

Figuur 6.2: k9s uitvoer van Chaos Toolkit experiment 1

Men hoeft de configuratie niet noodzakelijk te bekomen via het **chaos init** commando maar kan een experiment ook opzetten via een vooraf gedefinieerd YAML-bestand. Via volgende stappen zet men dit op:

1. Creëer een nieuw bestand genaamd 'random-pod-kill.yaml' m.b.v. een teksteditor naar keuze.
2. Kopieer de YAML-definitie van dit experiment die men kan terugvinden via volgende link: [Experiment 1: random-pod-kill.yaml](#)
Men ziet in dit bestand eerst een beschrijving van het experiment, gevolgd door de configuratie van de 'method'. Ook hier ontbreekt dus de configuratie van andere componenten zoals een steady state of een rollback, die in volgende experimenten nog aan bod zullen komen.
3. Neem de inhoud van dit bestand over in het nieuw gecreëerde bestand random-pod-kill.yaml en sla op.
4. Herhaal het experiment m.b.v. deze YAML-definitie via commando **chaos run random-pod-kill.yaml**.

6.2.4 Experiment 2: uitbreiding experiment 1 met steady state

In dit experiment zal experiment 1 aangevuld worden met een Steady State Hypothesis, die voor en na het verwijderen van een willekeurige nginx pod het aantal nginx pods zal controleren.

Eerst zal men de stappen via de wizard nogmaals doorlopen om dit experiment op te zetten maar deze keer zal gevraagd worden een YAML-definitie als output te bekomen. Dit doet men via commando **chaos init --experiment-path random-pod-kill-ssh.yaml** (ssh = steady state hypothesis) Start de setup van dit experiment via commando **chaos init**. Configureer volgende zaken via de wizard:

1. **Experiment's title:** random-pod-kill-ssh
2. Antwoord met 'y' op de vraag 'Do you want to define a steady state hypothesis now?'
3. **Hypothesis's title:** Er zijn drie nginx pods aanwezig
4. **Add an activity:** geef het nummer op van de optie 'count_pods'.
5. Antwoord met 'y' op de vraag 'Do you want to use this probe?'
6. **What is the tolerance for this probe?:** 3 (= het aantal nginx pods in de applicatie)
7. **label_selector:** app=nginx
8. **phase:** via enter kiest men de default [].
9. **ns:** demoapp1
10. Antwoord met 'N' op de vraag 'Do you want to select another activity?'. Hiermee vraagt men namelijk als nog een andere controle in een tweede 'steady state hypothesis' moet gecreëerd worden.
11. Antwoord met 'y' op de vraag 'Do you want to define an experimental method?'
12. Resterende configuratie vanaf **label_selector** is dezelfde als bij experiment 1.

Let op: Wanneer men de zopas gecreëerde 'random-pod-kill-ssh.yaml' gaat bekijken zal men zien dat onder de configuratie bij **probes** bij de parameter **tolerance** single quotes rond de waarde 3 zullen staan. De kans is groot dat hierdoor de uitvoer van het experiment zal falen dus verwijder deze quotes en sla het bestand terug op. Het experiment uitvoeren doet men vervolgens via commando **chaos run random-pod-kill-ssh.yaml**.

In figuur 6.3 ziet men dat zowel voor als na de Action 'terminate-pod' de probe uitgevoerd wordt. Wanneer men dit experiment reproduceert zal de probe hoogstwaarschijnlijk na de actie terug falen met de melding: 'Steady state probe 'count_pods' is not in the given tolerance so failing this experiment'.

Dit komt omdat de probe direct na de actie uitgevoerd werd en er dus onvoldoende tijd werd gegeven aan Kubernetes om een nieuwe pod op te starten.

Als oplossing kan men een parameter **pauses** aan de configuratie toevoegen zodat na de actie enkele seconden gewacht wordt vooraleer opnieuw de controle uit te voeren. Deze parameter **pauses** kan men echter niet configureren via de chaos init wizard maar zal manueel toegevoegd moeten worden. Men kan gebruik maken van volgend voorgedefinieerd YAML-bestand om de configuratie over te nemen en het experiment te herhalen: Experiment 2: random-pod-kill-ssh.yaml

```

INFO] Validating the experiment's syntax
INFO] Experiment looks valid
INFO] Running experiment: random-pod-kill-ssh
INFO] Steady-state strategy: default
INFO] Rollbacks strategy: default
INFO] Steady state hypothesis: Er zijn drie nginx pods aanwezig
INFO] Probe: count_pods
INFO] Steady state hypothesis is met!
INFO] Playing your experiment's method now...
INFO] Action: terminate_pods
INFO] Steady state hypothesis: Er zijn drie nginx pods aanwezig
INFO] Probe: count_pods
CRITICAL] Steady state probe 'count_pods' is not in the given tolerance so failing this experiment
INFO] Experiment ended with status: deviated
INFO] The steady-state has deviated, a weakness may have been discovered

```

Figuur 6.3: Probe faalt bij uitvoer Chaos Toolkit experiment 2

6.2.5 Experiment 3: uitbreiding vorige experimenten

Door de tekortkomingen die in vorig experiment bewezen werden bij de experimentconfiguratie via het 'chaos init' commando zal in volgende experimenten louter nog gebruik gemaakt worden van een voorgedefinieerd YAML-bestand.

De parameter **qty** wordt in dit experiment geïntroduceerd waarmee men vorig experiment kan uitbreiden zodat meerdere pods in de namespace `demoapp1` vernietigd kunnen worden. Eveneens zal gebruik gemaakt worden van een andere probe genaamd **pods_in_phase** die de status van de pods zal controleren i.p.v. het aantal pods te tellen. De defaultwaarde van deze probe is ingesteld op 'Running' dus hoeft dit niet noodzakelijk in de configuratie opgenomen te worden. (ChaosToolkit, 2022f)

Volgende stappen kan men toepassen om het experiment op te zetten en uit te voeren:

1. Creëer een nieuw YAML-bestand genaamd 'multiple-random-pod-kill-ssh.yaml'.
2. Kopieer de definitie van dit experiment die men kan terugvinden via volgende link:
Experiment 3: multiple-random-pod-kill-ssh.yaml
3. Voer dit experiment uit m.b.v. commando **chaos run multiple-random-pod-kill-ssh.yaml**

De YAML-definitie van dit experiment ontbreekt een waarde bij parameter **label_selector** waardoor nu alle pods, inclusief de Apache pods, in de namespace `demoapp1` getroffen kunnen worden. De slaagkans van de probe na de actie 'terminate_pods' is afhankelijk van de snelheid waarmee Kubernetes de nieuwe pods kan creëren.

6.2.6 Experiment 4: bereikbaarheid van Podtato-Head applicatie testen

In dit experiment zal de demo-applicatie Podtato-Head in namespace `demoapp2` gebruikt worden. Deze applicatie is alreeds opgezet in hoofdstuk 6.1.2 en men kon daar alreeds de URL te zien krijgen waarop de applicatie bereikbaar is. Deze URL zal men nodig hebben in de configuratie van het experiment.

Dit experiment zal het gevolg testen van het vernietigen van pod 'podtato-head-entry', die

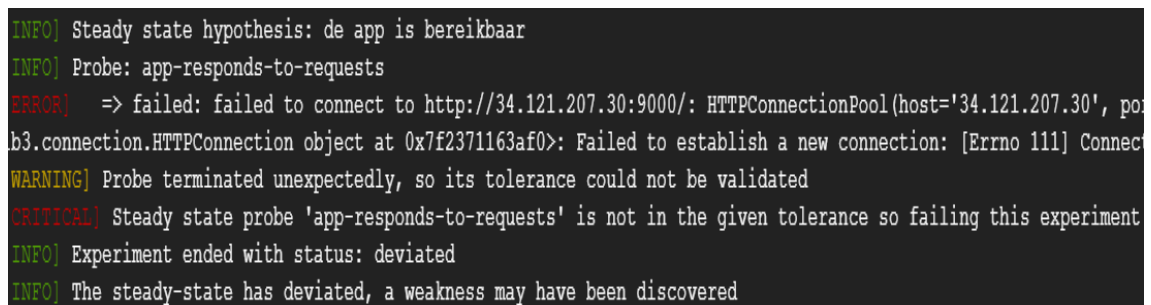
verantwoordelijk is voor de toegang tot de applicatie. Voor en na de vernietiging van deze pod zal via een **http probe** gecontroleerd worden als de applicatie terug bereikbaar is. (ChaosToolkit, 2022e)

Men kan in deze probe een parameter **timeout** configureren en een waarde meegeven hoelang het maximum mag duren om een antwoord op de GET-request te krijgen. Als antwoord wordt een HTTP-response status code 200 verwacht, wat wil zeggen dat de applicatie bereikbaar is. Alternatief kunnen ook twee waarden als JSON array geconfigureerd worden waarbij de eerste waarde de connection timeout weergeeft, en de tweede waarde de request timeout. Meer info over deze twee types timeout kan men in volgende link terugvinden: [what-is-a-connection-timeout-during-a-http-request](#)

Gebruik volgend stappenplan om dit experiment op te zetten:

1. Creëer een nieuw YAML-bestand genaamd 'podtato-entry-kill-httpprobe.yaml'.
2. Kopieer de definitie in volgende link om dit bestand op te vullen: Experiment 4: podtato-entry-kill-httpprobe.yaml
3. Pas de parameter **url** aan naar deze die men bij de setup van de Podtato-Head applicatie in hoofdstuk 6.1.2 alreeds heeft bekomen.
4. Sla de wijzigingen op.
5. Voer het experiment uit via commando **chaos run podtato-entry-kill-httpprobe.yaml**.

Het resultaat van dit experiment ziet men in figuur 6.4. Ondanks een pauze van tien seconden faalt de probe na de actie nog steeds. Op deze manier kan men zien dat de applicatie meer tijd nodig heeft vooraleer deze terug bereikbaar is.



```
INFO] Steady state hypothesis: de app is bereikbaar
INFO] Probe: app-responds-to-requests
ERROR] => failed: failed to connect to http://34.121.207.30:9000/: HTTPConnectionPool(host='34.121.207.30', port=9000): Max retries exceeded with url: / HTTPConnectionPool: Failed to establish a new connection: [Errno 111] Connection refused
WARNING] Probe terminated unexpectedly, so its tolerance could not be validated
CRITICAL] Steady state probe 'app-responds-to-requests' is not in the given tolerance so failing this experiment
INFO] Experiment ended with status: deviated
INFO] The steady-state has deviated, a weakness may have been discovered
```

Figuur 6.4: http probe faalt bij uitvoer Chaos Toolkit experiment 4

Oplossing: Een applicatie die onbereikbaar is wil men ten allen tijde vermijden. Wat men ziet in de Podtato-Head applicatie is dat elke pod slechts één keer voorkomt. In deze applicatie wordt elke pod voorzien door een aparte Deployment.

Een voordeel van pods die opgezet worden via een Deployment is dat deze makkelijk te schalen zijn. Als oplossing op voorgaand gefaald experiment kan men dus de Deployment die verantwoordelijk is voor de 'podtato-head-entry' pod schalen zodat deze steeds twee pods actief heeft. Als één pod in de problemen komt zal de andere de bereikbaarheid van de applicatie toch nog verzekeren. Men kan dit controleren via commando **kubectl get deploy -n demoapp2**. In de output van dit commando ziet men dat de Deployment eveneens 'podtato-head-entry' noemt.

Om deze te schalen zodat steeds twee pods actief zijn past men volgend commando toe: **kubect1 scale deploy podtato-head-entry --replicas=2 -n demoapp2**. Als gevolg kan men zien via de k9s tool dat een extra podtato-head-entry pod gecreëerd is in namespace demoapp2.

Nota: Zoals alreeds vermeld in Hoofdstuk 5.1.2 is het eveneens mogelijk deze actie uit te voeren vanuit het Workloads menu in het Google Cloud Platform.

Herhaal vervolgens het experiment en als resultaat zal men nu zien dat de applicatie bereikbaar blijft wanneer één van de podtato-head-entry pods getroffen wordt. Als test kan men de parameter pauses in commentaar plaatsen in de YAML-definitie, zodat de http probe rechtstreeks na de actie uitgevoerd wordt.

6.2.7 Referenties toepassen in een experiment

Wanneer men een actie of probe meermaals wil uitvoeren dan kan men via een referentie verwijzen naar de alreeds bestaande configuratie van die actie of probe. Zie als voorbeeld de onderstaande code uit de YAML-definitie van experiment 4 waarin een probe herhaald wordt door ernaar te refereren via parameter **ref**:

```
steady-state-hypothesis:
  title: Applicatie is bereikbaar
  probes:
    - name: response-OK
      type: probe
      tolerance: 200
      provider:
        type: http
        timeout: [1,2]
        verify_tls: false
        url: http://34.121.207.30:9000/
    - type: probe
      tolerance: 200
      ref: response-OK
```

Nota: Men hoeft een probe niet noodzakelijk in een steady-state-hypothesis te definiëren. Wil men een probe tijdens de actie van het experiment uitvoeren dan kan men dit in de 'method' definitie alreeds toepassen. Hou rekening dat deze controle dan direct na de actie uitgevoerd wordt waardoor deze vaak niet het gewenste resultaat zal geven.

6.2.8 Experiment 5: Een nodefaling simuleren

Pods blijven niet voor eeuwig op dezelfde node bestaan. Er zijn enkele redenen wanneer de locatie van een pod kan wijzigen o.a. wanneer autoscaling geactiveerd is en hierdoor nodes op- en afgebouwd kunnen worden, of men een onderhoud moet uitvoeren aan een

bepaalde node in een Kubernetes cluster waardoor de aanwezige pods op deze node moeten verhuizen naar een andere node.

Het proces waarbij alle pods op één van de nodes verwijderd worden en verhuizen richting een andere node noemt men **node draining**. Dit proces kan zowel automatisch door Kubernetes toegepast worden alsook manueel door een administrator.

In volgend experiment zal m.b.v. de actie **drain_nodes** een nodefaling gesimuleerd worden. Tijdens dit experiment zal men live deze verhuis kunnen opvolgen in de k9s tool. Het experiment zal ook de bereikbaarheid van de applicatie controleren voor en na de verhuis om te bevestigen dat deze nog steeds toegankelijk is. (ChaosToolkit, 2022c)

De nodes in de cluster kan men opvragen via commando **kubectl get nodes**:

```
$ kubectl get nodes
NAME                                     STATUS
gke-cluster-1-default-pool-1a61ddf4-mcs9 Ready
gke-cluster-1-default-pool-1a61ddf4-z4rx Ready
```

In bovenstaande output ziet men dat er momenteel twee nodes actief zijn. Het experiment wordt toegepast op de pods van de Podtato-Head applicatie. Hoe de pods over deze twee nodes verdeeld zijn kan men bekijken via commando **kubectl get pods -n demoapp2 -o wide**.

In de YAML-definitie van dit experiment zullen ook enkele argumenten opgenomen worden met als doel het experiment te richten op een node waarop specifiek pods van de demoapp2 namespace actief zijn (= **node_label**), alsook een node te kiezen waarop specifiek de pod 'podtato-head-entry' aanwezig is (= **pod_label_selector**).

Om een waarde toe te kennen aan parameter **node_label** is de output van het commando **kubectl describe nodes** nodig. Wanneer men naar de eerste regels van deze output gaat ziet men enkele labels gekoppeld aan de nodes in de cluster. Men heeft dus verschillende opties qua labels om te koppelen aan de parameter. Aangezien deze labels kunnen variëren op basis van de cluster die opgezet is (vb. andere cloud provider of lokale clusters) wordt buiten de YAML-definitie eerst een omgevingsvariabele gecreëerd op het systeem.

In het geval van de Google Cloud cluster is deze waarde 'beta.kubernetes.io/os=linux'.

Pas volgende stappen toe om dit experiment op te zetten:

1. Creëer de omgevingsvariabele via commando
export NODE_LABEL="beta.kubernetes.io/os=linux"
2. Creëer een nieuw YAML-bestand genaamd 'drain-node.yaml'.
3. Kopieer de experimentdefinitie in volgende link naar het nieuwe bestand: Experiment 5: drain-node.yaml
4. Pas de parameter **url** aan naar deze die men bij de setup van de Podtato-Head applicatie in hoofdstuk 6.1.2 alreeds heeft bekomen en sla de wijzigingen op.
5. Voer het experiment uit via commando **chaos run drain-node.yaml**.

Men kan via de k9s tool opvolgen dat alle pods van één van de nodes verhuizen richting een andere node. In de configuratie van het experiment kreeg het hiervoor twee minuten tijd. Soms blijkt dit echter niet genoeg waardoor de uitvoer van het experiment toch nog foutmeldingen geeft. Het experiment bevestigde eveneens m.b.v. een http probe dat de Podtato-Head applicatie ondertussen bereikbaar blijft.

Als men de nodes terug opvraagt via commando **kubectl get nodes** zal men opmerken dat bij kolom **Status** de term **SchedulingDisabled** te zien is.

```
$ kubectl get nodes
NAME                                     STATUS
gke-cluster-1-default-pool-1a61ddf4-mcs9 Ready
gke-cluster-1-default-pool-1a61ddf4-z4rx Ready, SchedulingDisabled
```

Dit is het gevolg van de node drain actie en hierdoor zal de Kubernetes Scheduler deze node als onplanbaar zal zien m.a.w. er zullen geen nieuwe pods op deze node geplaatst worden. Aangezien dit geen gewenst scenario is zal men dit proces deze keer wel moeten terugdraaien via een **rollback**. Het proces waarbij de Status veranderd wordt richting 'SchedulingDisabled' en de node als gevolg onplanbaar wordt noemt men **node cordoning**. (DeFabia, 2022)

6.2.9 Experiment 6: Rollback toepassen op experiment 5

Het vorige experiment heeft een node onbruikbaar gemaakt. Dit proces kan via ChaosToolkit eveneens omgedraaid worden door het implementeren van de rollback-actie **uncordon_node**. (Chaostoolkit, 2022)

Pas volgende stappen toe om de rollback toe te passen:

1. Creëer een nieuw YAML-bestand genaamd 'drain-node-with-rollback.yaml'.
2. Kopieer de YAML-definitie uit volgende link naar dit nieuwe bestand: Experiment 6: drain-node-with-rollback.yaml
3. Pas de parameter **url** aan naar deze die men bij de setup van de Podtato-Head applicatie in hoofdstuk 6.1.2 alreeds heeft bekomen.
4. Sla de wijzigingen op.
5. Voer het experiment uit via commando **chaos run drain-node-with-rollback.yaml**.
6. Vraag de status van de nodes op via commando **kubectl get nodes**.

Resultaat: Men kan zien dat de Status van de nodes nu terug naar Ready veranderd is. Dit experiment is uitgevoerd in een Google Cloud cluster en het gevolg van deze experimenten was vaak dat eveneens een nieuwe node werd opgezet om het verlies van de getroffen node op te vangen.

GKE cluster resizing

Aangezien Google Cloud via het 'pay per node' principe werkt zullen de gratis credits dus sneller verbruikt worden indien meer nodes dan nodig actief zijn. Via volgend commando kan men de nodes in de cluster terugschalen naar een gewenst niveau: **gcloud container clusters resize [cluster naam] --zone [cluster zone] --num-nodes=[aantal nodes]**.

Voorbeeld: `gcloud container clusters resize cluster-1 --zone us-central1-c --num-nodes=2`

Nota: dit resizing proces zal ongeveer een minuut tijd in beslag nemen.

6.2.10 Conclusie ChaosToolkit

ChaosToolkit is eenvoudig te installeren op het systeem. Men kan de functionaliteit van deze chaos engineering tool uitbreiden via verschillende extensies/plugins. Vooraf gedefinieerde experimenten worden niet aangeboden, maar afzonderlijke configuraties voor verschillende Kubernetes objecten, probes, rollbacks ... kan men op de website <https://chaostoolkit.org/> terugvinden.

Op deze manier krijgt men de vrijheid zelf een experiment naar wens samen te stellen. Om de eerste experimenten op te zetten kan men via het **chaos init** commando een wizard raadplegen. Zowel JSON als YAML-definities worden aanvaard als experimentconfiguratie. Deze tool is goed gedocumenteerd.

Enkele nadelen aan Chaos Toolkit zijn:

- het ontbreken van een GUI waarmee via de browser experimenten opgezet kunnen worden.
- het ontbreken van de optie om experimenten in te plannen zodat deze periodiek uitgevoerd kunnen worden. Dit zou eventueel via het configureren van een cronjob opgevangen kunnen worden.
- Sommige experimenten zijn enkel beschikbaar voor bepaalde omgevingen bv. het belasten van de CPU is enkel beschikbaar via de Azure extensie. Deze stresstesten kunnen nochtans een waardevol inzicht bieden in hoe een applicatie omgaat met belasting, wat een realistisch scenario is voor een applicatie in productie.
- de moeilijkheidsgraad om netwerkgerelateerde experimenten uit te voeren. Hiervoor moeten extra plugins en packages geïnstalleerd worden zoals Istio, die op zich ook de nodige technische kennis vereisen.

6.3 Chaos Mesh

Het Chaos Mesh project bestaat sinds begin 2020 en was oorspronkelijk bedoeld als test-platform voor de open-source NewSQL-database TiDB. Chaos Mesh is een veelzijdig chaos engineering platform om chaos experimenten uit te voeren in Kubernetes omgevingen. Het project werd in juli 2020 geaccepteerd als Cloud Native Computing Foundation

(CNCF) Sandbox project en is sinds begin 2022 geëvolueerd naar een CNCF Incubating project. (CNCF, 2022a)

De algemene documentatie op de Chaos Mesh website toont aan dat voor enkele lokale omgevingen een oneliner script kan toegepast worden om deze tool te installeren. (ChaosMesh-documentation, 2022a) Bij de start van dit onderzoek werd de installatie van Chaos Mesh oorspronkelijk uitgevoerd op een single-node Minikube cluster. Hierin werden enkele experimenten succesvol uitgevoerd via de terminal en de Chaos Mesh GUI genaamd Chaos Dashboard.

Nadien is ChaosMesh geïnstalleerd in de lokale multi-node cluster opgezet via Kubespray. Enkele experimenten via de terminal konden in deze omgeving ook uitgevoerd worden. De GUI Chaos Dashboard toonde echter verschillende foutmeldingen in de browser bij het openen van de Developer Tools (F12). Pas later in het onderzoek is bij herinstallatie van Chaos Mesh een correcte werking van het Chaos Dashboard vastgesteld.

Aangezien alreeds enkele lokale setups uitgeprobeerd waren is besloten de installatie van Chaos Mesh uit te voeren in een cluster die opgezet is via GKE, waar het Chaos Dashboard wel opgestart kon worden. Volgende beschrijving hoe men Chaos Mesh kan installeren m.b.v. de package manager Helm en hoe experimenten vervolgens uitgevoerd kunnen worden, zal zich specifiek op deze Google Cloud omgeving richten. (ChaosMesh-documentation, 2022b)

Vereisten

Om Chaos Mesh via de package manager **Helm** te kunnen installeren moet Helm vooraf geïnstalleerd worden op het systeem. In volgende link kan men verschillende manieren terugvinden om Helm te installeren: <https://helm.sh/docs/intro/install/>

6.3.1 Installatie Chaos Mesh

Pas volgend stappenplan toe om ChaosMesh te installeren:

1. Voeg Chaos Mesh toe aan de helm repositories via volgend commando **helm repo add chaos-mesh <https://charts.chaos-mesh.org>**.
2. Bekijk welke versie van Chaos Mesh in de helm repositories beschikbaar is via commando **helm search repo chaos-mesh**. Deze info zal men in stap 4 nodig hebben.
3. Creëer de namespace ‘chaos-testing’ via commando **kubectl create namespace chaos-testing**.
4. Installeer de nodige Chaos Mesh componenten in de chaos-testing namespace.
Let op: GKE maakt gebruik van de container runtime containerd en NIET van Docker. Daardoor moet men dit specificeren in het installatiecommando om errors in de toekomst te vermijden. Om de Chaos Mesh componenten te installeren gebruikt men onderstaand commando:

```
$ helm install chaos-mesh chaos-mesh/chaos-mesh \
--n=chaos-testing --set chaosDaemon.runtime=containerd --set \
chaosDaemon.socketPath=/run/k3s/containerd/containerd.sock \
--version 2.2.0
```

- Controleer als de Chaos Mesh componenten in de namespace zijn geïnstalleerd en operationeel zijn via commando **kubectl get pods -n chaos-testing**. Een correcte installatie in GKE zou volgende output moeten tonen:

```
kubectl get pods -n chaos-testing
```

NAME	READY	STATUS
chaos-controller-manager-7dc9bf54c6-4bp7q	1/1	Running
chaos-controller-manager-7dc9bf54c6-srgbw	1/1	Running
chaos-controller-manager-7dc9bf54c6-vwv5l	1/1	Running
chaos-daemon-bvpgd	1/1	Running
chaos-daemon-k6pz5	1/1	Running
chaos-daemon-stqvz	1/1	Running
chaos-dashboard-7f7bc7cdfb-pbj52	1/1	Running

6.3.2 Toegang voorzien tot het Chaos Dashboard

Na installatie van Chaos Mesh is alreeds een NodePort service opgezet voor het Dashboard. Bevestig dit via commando **kubectl get svc -n chaos-testing**:

```
$ kubectl get svc -n chaos-testing
```

NAME	TYPE	PORT(S)
chaos-daemon	ClusterIP	31767/TCP,31766/TCP
chaos-dashboard	NodePort	2333:30061/TCP,2334:30554/TCP
chaos-mesh-controller	ClusterIP	443/TCP,10081/TCP,10082/TCP

Met behulp van volgende stappen kan men via de browser connectie maken met het Chaos Dashboard:

- Men zal eerst een firewall regel moeten toevoegen in de Google Cloud Shell die verkeer toelaat op de poort verbonden aan de Nodeport service van het Chaos Dashboard die in vorig commando opgevraagd is. Gebruik één van de poorten NA de dubbelpunt en creëer een firewall regel via volgend commando:
gcloud compute firewall-rules create chaos-dashboard-rule --allow tcp:30061
- Men kan vervolgens het extern IP-adres van één van de nodes gebruiken in combinatie met de poort die in vorige stap gebruikt werd om via de browser naar het Chaos Dashboard te gaan.
Tip: om het extern IP-adres van een node te bekomen gebruikt men commando **kubectl get nodes -o wide**.

Wanneer men voor het eerst gebruik maakt van het Chaos Mesh Dashboard zal men een pop up venster zien waarin een token moet ingegeven worden. Kies voor 'Click here to generate' om een nieuwe token te maken. De bedoeling van deze token is om een RBAC

autorisatie in te stellen waarmee men definieert wat de rechten zijn van een gebruiker van het Chaos Dashboard ten opzichte van de Kubernetes cluster. (ChaosMesh, 2022a)
In volgende stappen wordt stapsgewijs de procedure uitgelegd om deze token te creëren:

1. Vink bovenaan de checkbox 'Cluster scoped' aan om de scope van de experimenten over heel de Kubernetes cluster mogelijk te maken.
2. Kies vervolgens in de dropdownlist voor 'Manager' om uzelf alle rechten te geven omtrent het maken, updaten, uitvoeren en vernietigen van Chaos experimenten.
3. De configuratie wordt dynamisch aangepast ten opzichte van de keuzes uit stap 5 en 6. Kies rechtsbovenaan voor 'Copy' om deze configuratie te kopiëren.
4. Ga naar de Cloud shell terminal en maak een nieuwe directory 'chaosmesh-experiments'.
5. Maak in deze directory een nieuwe YAML-definitie genaamd 'rbac.yaml'. Open dit bestand en kleef de inhoud van de configuratie uit voorgaande stappen hierin en sla vervolgens op.
6. Voer dit bestand uit via commando **kubectl apply -f rbac.yaml**. Dit zal o.a. een Service Account aanmaken en de RBAC autorisatie op de cluster instellen.
7. Haal de token op via het commando beschreven in het oorspronkelijke pop-up venster. Kopieer de token in de output van dit commando en plaats deze in het tekstvak van het pop-up venster.
8. Geef een betekenisvolle naam aan deze token vb. 'token-fullscope-manager' en klik op Submit. De toegang tot het Dashboard is nu afgehandeld.

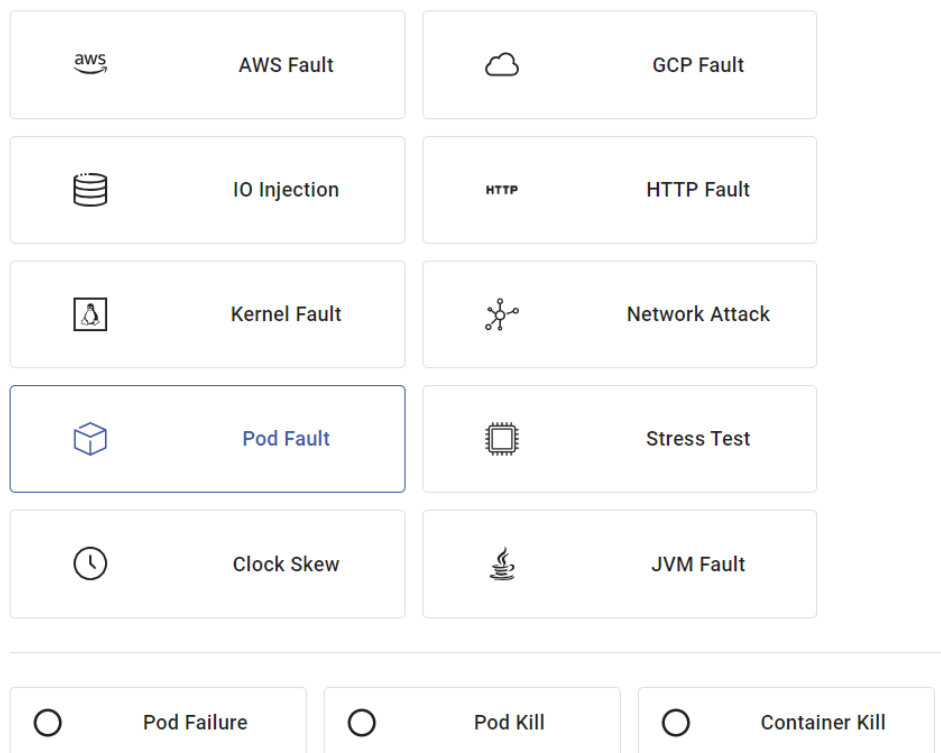
6.3.3 Overzicht van het Chaos Dashboard

Via de beschrijving in vorig hoofdstuk 6.3.2 kon men alreeds de toegang bekomen tot het Chaos Dashboard. Men komt hierdoor terecht op de default startpagina in het Dashboard menu. In dit overzicht ziet men verschillende mogelijkheden om een experiment op te zetten, waaronder:

- **New Experiment:** Link naar het Experiments menu waar men een experiment kan opzetten voor eenmalige uitvoer.
- **New Workflow:** Link naar het Workflows menu waar men meerdere experimenten in sequentie kan opzetten (= workflow).
- **New Schedule:** Link naar het Schedules menu waar men experimenten kan opzetten die herhaald worden op vaste tijdstippen.

Aanbod van experimenten

Wanneer men via 'New Experiment' een experiment wil opzetten zal eerst een keuze moeten gemaakt worden uit twee groepen nl. Kubernetes of Hosts. Afhankelijk van de keuze zal men daaronder een lijst experimenten te zien krijgen. Deze opties zijn gegroepeerde experimenten, d.w.z. dat elke optie in de lijst een aantal specifiekere experimenten bevat. Zie onderstaand voorbeeld in figuur 6.5 waarbij men de groep 'Pod Fault' selecteert en vervolgens drie keuzes onderaan de lijst krijgt qua experimenten die uitgevoerd kunnen worden.



Figuur 6.5: Chaos Mesh Experimenten per groep

6.3.4 Pod kill experiment in Chaos Dashboard via New Experiment

Het eerste experiment zal één willekeurige nginx pod uit de demo applicatie in namespace demoapp1 vernietigen. Volgende stappen omschrijven hoe men dit kan opzetten in het Chaos Dashboard:

1. Kies in de Dashboards pagina voor New Experiment.
2. In tabblad **Experiment Type** kiest men bovenaan voor 'Kubernetes' en vervolgens in de lijst voor 'Pod Fault'.
3. Men krijgt als reactie op voorgaande keuze drie nieuwe keuzes te zien van experimenten die onder 'Pod Fault' beschikbaar zijn. Kies voor 'Pod Kill' en klik vervolgens op Submit om de keuze te bevestigen.
4. In tabblad **Experiment Info** configureert men vervolgens het experiment. Gebruik volgende configuraties om het Pod Kill experiment toe te passen op de nginx pods uit de demo applicatie:
 - **Scope:** Bij namespace stelt men in dat het PodChaos experiment in 'demoapp1' mag geplaatst worden.
 - **Metadata:** Geef zelf een naam aan het experiment.
 - **Label Selectors:** specificeer welke pods getroffen moeten worden o.b.v. de label, in dit geval 'app=nginx'.
 - **Mode:** Hier configureert men hoeveel pods getroffen moeten worden. Het experiment is gericht op één pod dus men kiest men 'Random One'. Alternatieve keuzes zijn een vast aantal, procentueel, allemaal ...

- **Preview of Pods to be injected:** (Optioneel) Dit toont een lijst o.b.v. voorgaand gemaakte keuzes waarin men kan zien welke pods getroffen zullen worden door het experiment. Hier kan men eventueel nog pods de-selecteren die niet door het experiment getroffen mogen worden.
 - Klik op Submit om de configuraties te bevestigen.
5. Wanneer zowel het Experiment Type als de Experiment Info via Submit bevestigd zijn zal nog een derde keer bevestigd moeten worden via Submit om het experiment daadwerkelijk te starten.

Figuur 6.6: Configureren van experiment in Chaos Dashboard

Na de bevestiging ziet men het experiment verschijnen in het menu Experiments met als status 'Injecting'. Wanneer men hierop klikt zal extra info weergegeven worden waaronder de configuratie, de Events (die de fases van het experiment beschrijven) en de YAML-definitie van het experiment.

Men kan een (statisch) overzicht van de pods opvragen in de Cloud Shell terminal m.b.v. commando **kubectl get pods -n demoapp1**, maar nog een betere optie is om dit experiment op te volgen via k9s. Gebruik hiervoor het commando **./k9s -n demoapp1** om specifiek de pods in de namespace van de demo-applicatie te openen. Op deze manier kan men live volgen hoe na het starten van het experiment één willekeurige nginx pod uit de demo-applicatie vernietigd wordt. In de kolom 'Age' zal na afloop te zien zijn dat één van de nginx pods nog maar enkele seconden actief is.

In bovenstaand stappenplan werd elke parameter van het experiment geconfigureerd in de GUI. In figuur 6.6 ziet men bovenaan ook nog twee andere tabbladen nl.:

- **Load from:** bedoeld om configuratie van een alreeds uitgevoerd experiment te herladen en aan te passen. Let op: men moet de naam van het experiment wijzigen

aangezien anders een foutmelding op het scherm zal verschijnen die zal aangeven dat het experiment alreeds bestaat.

- **By YAML:** bedoeld om zelf een YAML-definitie te schrijven of een alreeds bestaande vanop het systeem in te laden.

6.3.5 Een experiment plannen via New schedule

In vorige sectie werd beschreven hoe een enkelvoudig experiment opgezet werd. Een meer praktische benadering van chaos engineering zou zijn om een experiment meerdere malen te herhalen om de applicatie op verschillende tijdstippen te testen. In het Chaos Dashboard kan men dit bekomen via de optie 'New schedule' in het Dashboard menu, of door direct naar het 'Schedules' menu te gaan.

Op een gelijkaardige manier als voordien zal men het experiment kunnen configureren. Enkele extra parameters zullen hier ook geconfigureerd moeten worden nl.:

- **newS.basic.historyLimit:** het aantal logbestanden (= records) dat er moeten bewaard worden.
- **newS.basic.concurrencyPolicy:** toestaan dat het experiment gelijktijdig met andere experimenten wordt uitgevoerd.
- **Schedule:** via een cron schedule aangeven op welke tijdstippen het experiment moet uitgevoerd worden. Een link naar de website <https://crontab.guru/> wordt meegegeven om deze parameter op een correcte manier te configureren.

Geplande experimenten kunnen opgevolgd worden in het 'Schedules' menu, waar het experiment de status 'Running' krijgt. Deze status wordt behouden tot men beslist het experiment te archiveren.

6.3.6 Meerdere experimenten opzetten via New workflow

Wanneer men meerdere experimenten wil uitvoeren in serie, parallel, in combinatie met een taak ... dan doet men dit d.m.v. een workflow te creëren. In het Chaos Dashboard kan men dit bekomen via de optie 'New workflow' in het Dashboard menu, of door direct naar het 'Workflows' menu te gaan.

Een praktische benadering in een workflow zou zijn om parallel een experiment uit te voeren waarbij willekeurige pods van een applicatie getroffen worden en ondertussen ook de bereikbaarheid van de applicatie herhaaldelijk te controleren gedurende de tijdspanne van het experiment. Dit zou men kunnen doen door een GET-request uit te sturen en te controleren als een HTTP-response met code 200 terugkeert, wat wil zeggen dat de request succesvol was. (MDN, 2022)

Jammer genoeg is het opzetten van dit scenario in een workflow op het eerste zicht niet mogelijk. Men kan wel opteren voor één HTTP-request uit te sturen via de task type 'HTTP Request'. Na onderzoek bleek deze functionaliteit zich nog in een experimentele

fase te bevinden. (ChaosMesh, 2022b)

Ook op de Chaos Mesh Q&A pagina kan men lezen dat 'probe support' nog niet gepland staat om toegevoegd te worden aan de functionaliteiten van Chaos Mesh. (ChaosMesh, 2021)

Het iteratief herhalen van een experiment is wel mogelijk via een workflow, maar is vrij omslachtig. Men moet hierbij hetzelfde experiment meerdere malen configureren zodat dit achtereenvolgens uitgevoerd kan worden. Een voorbeeld van de YAML-definitie die ontstaan is bij het testen van het iteratief vernietigen van een willekeurige pod van de applicatie in namespace demoapp2 kan men in volgende link terugvinden: `iteration-test.yaml`

6.3.7 Experimenten opzetten in de terminal

Op de Chaos Mesh website vindt men heel wat voorbeelden van YAML-definities van experimenten terug. Open volgende link naar de website en ga in het linkermenu naar 'Types of Chaos Experiments': <https://chaos-mesh.org/docs/>.

Hieronder krijgt men opnieuw de keuze tussen 'Kubernetes' en 'Physical Nodes' te zien. Als men verder gaat in optie Kubernetes ziet men dezelfde oplijsting als voordien in het Chaos Dashboard. In de documentatie van elk experiment ziet men o.a. hoe men dit kan opzetten via de GUI, maar ook een voorbeeld YAML-definitie die men via de terminal kan uitproberen.

Experiment 1: pod-failure

Een eerste experiment om via de terminal uit te proberen is het pod-failure experiment. Hierbij zal over een periode van dertig seconden één willekeurige pod van de PodTato-Head applicatie uit de namespace demoapp2 falen.

Een alreeds aangepaste YAML-definitie van het hieronder beschreven pod-failure experiment kan men vinden via volgende link: `pod-failure.yaml`

Om manueel het pod-failure experiment te configureren gebruikt men volgend stappenplan. Hierbij wordt gebruik gemaakt van de voorbeeld YAML-definitie op de Chaos Mesh website.

1. Ga op de Chaos Mesh website in het linkermenu naar Types of Chaos Experiments. Kies voor Kubernetes en vervolgens voor 'Simulate Pod Faults'.
2. Kopieer de inhoud van de YAML-definitie in sectie 'pod-failure example'.
3. Ga naar de Cloud Shell terminal en maak een nieuw bestand aan die 'pod-failure.yaml' noemt in de directory 'chaosmesh-experiments'.
4. Plaats de inhoud van het experiment in dit bestand en pas volgende zaken aan:
 - wijzig de waarde bij parameter namespace (onder metadata) naar 'demoapp2'.
 - verwijder onder selector de parameter labelSelector en de toegewezen waarde.
 - voeg onder selector een parameter 'namespaces' toe met de waarde 'demoapp2'. Als voorbeeld kan men kijken naar de configuratie van het pod-kill.yaml

bestand verderop in de documentatie.

5. Sla de wijzigingen op.
6. Voer het experiment uit via commando **kubectl apply -f pod-failure.yaml**. Men zal hierbij volgende melding te zien krijgen: 'podchaos.chaos-mesh.org/pod-failure-example created'

Nota: Indien men hier de foutmelding 'Admission webhook "vauth.kb.io" denied the request' te zien krijgt kan men een workaround toepassen via volgend commando uit te voeren en nadien het experiment te herhalen:

kubectl delete validatingwebhookconfigurations.admissionregistration.k8s.io validate-auth (Keao, 2021)

Men kan bij het uitvoeren van het pod-failure experiment zien via k9s dat een restart afgedwongen wordt bij één van de pods in de Podtato-Head applicatie. Wanneer men via k9s op de getroffen pod staat kan men de beschrijving openen door op 'D' te duwen. Onderaan deze pagina in sectie Events zal men zien hoe de pod gedwongen wordt om te herstarten doordat de definitie van de container gewijzigd is door de foutinjectie.

Type	Reason	Message
----	-----	-----
Normal	Started	Started container podtato-head-entry
Normal	Killing	Container podtato-head-entry definition changed, will be restarted
Normal	Created	Created container podtato-head-entry
Normal	Pulled	Container image "ghcr.io/podtato-head/entry:0.2.7" already present on machine

Figuur 6.7: opvragen van Events van getroffen pod via k9s

Het starten van het pod-failure experiment creëerde een 'podchaos' object in de demoapp2 namespace. Dit kan men controleren via commando **kubectl get podchaos -n demoapp2**. Indien men Events wil opvragen van het bovenstaand experiment dan spreekt men het 'podchaos' object aan via commando **kubectl describe podchaos pod-failure-example -n demoapp2**.

Nota: Afhankelijk van het type experiment zal dit object variëren vb. bij een later experiment waarbij CPU-belasting zal opgewekt worden noemt dit object 'stresschaos'. Deze info kan men afleiden uit de parameter 'kind' in de YAML-definitie van het experiment. Men zal eveneens zien dat dit experiment toegevoegd is in het Experiments menu van het Chaos Dashboard.

Een experiment herhalen via de terminal

Wanneer men het pod-failure experiment zou willen herhalen in de terminal, dan is dit niet mogelijk door gewoon opnieuw het **kubectl apply -f pod-failure.yaml** commando uit te voeren. Dit zal namelijk volgende melding genereren: 'podchaos.chaos-mesh.org/pod-failure-example unchanged'.

Men moet steeds eerst het bestaande podchaos object verwijderen via commando **kubectl delete podchaos pod-failure-example -n demoapp2** alvorens een experiment opnieuw te kunnen uitvoeren.

Experiment 2: NetworkChaos experiment

In dit experiment zal gedurende zestig seconden een communicatiestoring uitgelokt worden tussen de Apache en Nginx pods van de demo-applicatie in namespace demoapp1. Vooraf kan men al testen als de communicatie tussen beiden succesvol is. Dit doet men op volgende manier:

```
# Bewaar eerste Apache pod in environment variabele
$ pod=$(kubectl get pods -n demoapp1 -l app=apache \
-o jsonpath='{.items[0].metadata.name}')

# Open een shell in de container binnenin de Apache POD
# en controleer als nginx pods bereikt kunnen worden
$ kubectl exec $pod -n demoapp1 -it -- /bin/sh \
-c "curl nginx"

# Bewaar eerste Nginx pod in environment variabele
$ pod2=$(kubectl get pods -n demoapp1 -l app=nginx \
-o jsonpath='{.items[0].metadata.name}')

# Open een shell in de container binnenin de Nginx POD
# en controleer als Apache pods bereikt kunnen worden
$ kubectl exec $pod2 -n demoapp1 -it -- /bin/sh \
-c "curl apache"
```

Beide controles tonen de output van de Apache of Nginx website waardoor bevestigd is dat de communicatie tussen deze pods/services correct verloopt.

Het toepassen van dit experiment verloopt opnieuw door de YAML-definitie van de officiële Chaos Mesh website te raadplegen en dit op het systeem over te brengen in een nieuw bestand genaamd 'network-partition.yaml'.

Men kan alreeds een voorgedefinieerde YAML-definitie van dit NetworkChaos experiment via volgende link terugvinden: [network-partition.yaml](#)

Voer het experiment vervolgens uit via **kubectl apply -f network-partition.yaml**.

Door nu opnieuw de commando's toe te passen waarmee men de bereikbaarheid van de Nginx en Apache pods kon testen zal men nu gedurende de duurtijd van het experiment geen output meer te zien krijgen. Dit experiment toont het belang aan van Services die verantwoordelijk zijn voor de communicatie tussen pods op verschillende nodes.

Experiment 3: Stresschaos experiment

In dit experiment zal over een periode van twee minuten een belasting van ongeveer 200 MB geheugenverbruik gegenereerd worden op de nginx pods in de demo-applicatie in namespace demoapp1. Vervolgens zal het nut aangetoond worden van het Kubernetes object **HorizontalPodAutoscaler (HPA)**, die extra pods zal creëren wanneer een gecon-

figureerde CPU- of geheugenlimiet overschreden wordt.

Bij de demo-applicatie in `demoapp1` is een kanttekening te maken aangezien deze oorspronkelijk opgezet is zonder een limiet op de resources die het mag gebruiken. Hierdoor zou een applicatie alle resources van een node kunnen aanspreken, maar dit is geen ideaal scenario. Zo is een pod ingesteld volgens 3 QoS klassen. Pods zonder geconfigureerde resource management vallen onder de klasse 'best effort' en komen hierdoor als eerste in aanmerking om vernietigd te worden wanneer de node resources beperkt worden. (S, 2020)

Om de pods in de `demoapp1` namespace te configureren zodanig deze maximum 200 mCPU en 256MB gebruiken voert men volgende twee commando's uit (Kubernetes, 2022d):

```
$ kubectl set resources deployment nginx -n demoapp1 \
--limits=cpu=200m,memory=256Mi
```

```
$ kubectl set resources deployment apache -n demoapp1 \
--limits=cpu=200m,memory=256Mi
```

De officiële documentatie voor een StressChaos experiment op te zetten kan men hier terugvinden: <https://chaos-mesh.org/docs/simulate-heavy-stress-on-kubernetes/>

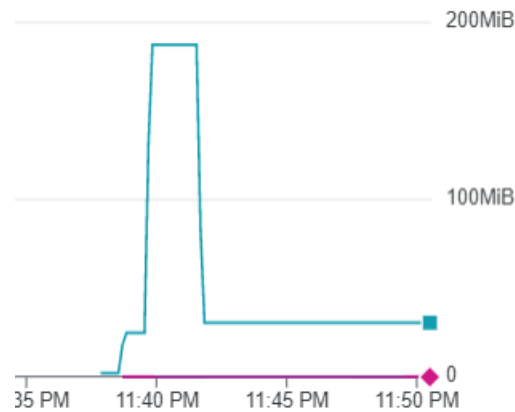
Men kan ook gebruik maken van de alreeds geconfigureerde YAML-definitie voor dit specifieke experiment via volgende link: Experiment 3: `memory-stress.yaml`

Maak een nieuw bestand aan genaamd 'memory-stress.yaml' in de directory 'chaosmesh-experiments' en plaats de inhoud uit bovenstaande link hierin. Start vervolgens het experiment op via commando **`kubectl apply -f memory-stress.yaml`**

Tijdens de uitvoer van het experiment kan men via het Google Cloud Platform de geheugenbelasting opvolgen via het **Workloads** menu. Kies vervolgens de Nginx Deployment in de `demoapp1` namespace om het overzicht van deze specifieke Deployment te zien. In dit overzicht ziet men drie metrics die opgevolgd worden nl. CPU, geheugen en diskverbruik. Tijdens de uitvoer van het experiment zal te zien zijn dat een piek veroorzaakt wordt in het geheugenverbruik.

Een oplossing voorzien om deze belasting te helpen opvangen bestaat in de vorm van het Kubernetes object **HorizontalPodAutoscaler**, hieronder in Hoofdstuk 6.3.7 beschreven. Nadat men de stappen heeft uitgevoerd die hierin beschreven staan kan men vervolgens het experiment opnieuw uitvoeren.

Let op: Men kan niet zomaar het experiment herhalen maar zal eerst het bestaande experiment moeten verwijderen. Dit doet men via commando **`kubectl delete stresschaos memory-stress-example -n demoapp1`**. Nu kan men opnieuw de YAML-definitie van het experiment oproepen via commando **`kubectl apply -f memory-stress.yaml`**. Via de k9s tool kan men vervolgens zien hoe drie extra Nginx pods aangemaakt worden om de belasting te helpen opvangen.



Figuur 6.8: GKE workloads: belasten van geheugen Nginx pods

Kubernetes object: HorizontalPodAutoscaler

Wanneer pods zwaar belast worden qua geheugen (of CPU) dan zal dit een negatieve impact hebben op de goede werking van een applicatie. Een te zware belasting waarbij een pod meer resources verbruikt dan toegelaten zal zelf resulteren in het beëindigen van de pod door de kernel Out-Of-Memory killer (OOMkill). Dit proces zal Kubernetes helpen om het geheugen te beheren wanneer pods aan nodes toegewezen worden en zal eveneens beslissingen nemen welke pods te vernietigen wanneer resources op de node in gevaar komen. (Alletto, 2021)

Een antwoord bieden om dit risico te helpen vermijden is het gebruiken van het Kubernetes object **HorizontalPodAutoscaler**. Dit object zal ervoor zorgen dat extra pods gegenereerd worden wanneer een geconfigureerde resources threshold overschreden wordt.

Een voorgeconfigureerde YAML-definitie voor het creëren van een HPA, gericht op de Nginx pods in demoapp1, en met een threshold ingesteld op 100 MB kan men via volgende link terugvinden: [hpa.yaml](#)

Creëer een nieuwe YAML-definitie op het systeem genaamd 'hpa.yaml' en voer dit vervolgens uit via commando **kubectl apply -f hpa.yaml**.

6.3.8 Conclusie Chaos Mesh

De installatie van Chaos Mesh verloopt snel en eenvoudig, zowel met het onliner script in een lokale omgeving als via de package manager Helm in de cloud omgeving. Het opzetten van het Chaos Dashboard daarentegen ging enkele keren mis waardoor dit onbruikbaar was. Deze tool vereist weinig resources in een cluster.

Het Chaos Dashboard is gebruiksvriendelijk opgesteld en vereist weinig kennis om direct aan de slag te kunnen. Het aanbod van experimenten is ruim en het configureren van deze via de GUI is eenvoudig, maar eveneens ontbreekt hier nog wat gewenste functionaliteit.

Zo is het bv. niet mogelijk om een experiment op te zetten waarbij doorlopende controle van de bereikbaarheid van een applicatie getest wordt via HTTP-requests, kan men geen logs raadplegen van uitgevoerde experimenten ...

Ook zou het praktisch zijn om in de configuratie van een experiment aan te geven dat dit iteratief uitgevoerd dient te worden bv. elke twintig seconden herhalen gedurende een periode van vijf minuten. Dit kan men momenteel enkel bekomen door een workflow te configureren waarbij enkelvoudige experimenten in serie herhaald worden, maar dit is vrij omslachtig aangezien telkens dezelfde configuratie gewoonweg herhaald wordt. Een extra nadeel is dat een experiment niet kan herhaald worden zonder dat eerst het origineel experiment gewist wordt.

Experimenten opzetten via de terminal is mogelijk, maar biedt geen meerwaarde t.o.v. het uitvoeren van experimenten via het Chaos Dashboard. Meeste experimenten in Chaos Mesh werden wel oorspronkelijk toegepast in de terminal doordat het Chaos Dashboard bij eerdere installaties nog onbruikbaar was. De documentatie voor de experimenten op te zetten is overzichtelijk en richt zich zowel op uitvoer via de terminal als via de GUI.

6.4 Litmus

In de zoektocht naar een chaos engineering tool die net zoals voorgaande tool Chaos Mesh zowel experimenten kon uitvoeren vanuit de terminal als via de browser, werd de keuze gemaakt om Litmus te onderzoeken.

Het Litmus project startte in 2017 met als doel om simpele chaos experimenten op te zetten in een Kubernetes cluster. Het werd een Cloud Native Computing Foundation (CNCF) sandbox project in 2020, en wordt vandaag onderhouden door vijf verschillende organisaties. Sinds begin 2022 is het project geëvolueerd naar een CNCF incubating project. (CNCF, 2022c)

Vereisten

Om Litmus te kunnen installeren moeten drie zaken aanwezig zijn (Litmus-docs, 2022):

- Kubernetes versie 1.17 of recenter
- Een Persistent Volume van 1GB waar Litmus de chaos configuratie en chaos-metrics zal opslaan. Standaard zal Litmus gebruik maken van de default storage class om deze Persistent Volume toe te wijzen.
- Helm versie 3 of kubectl

De installatie van Litmus is enkel toegepast in GKE waar een default storage class aanwezig is. Dit is echter niet het geval bij de lokale clusters eerder opgezet via Kubeadm en Kubespray. Verder onderzoek als deze tool kan geïnstalleerd worden in een lokale omgeving is hierdoor nog vereist.

6.4.1 Installatie

De installatie van Litmus verloopt in volgend beschreven stappenplan via Helm. Alternatief kan men ook de installatieprocedure via de kubectl commandline tool uitvoeren. Raadpleeg hiervoor de bron in 6.4.

De installatie kan men opsplitsen in twee delen. Eerst zullen de benodigde pods geïnstalleerd worden om Litmus op het systeem te krijgen. Nadien zal via de browser de toegang geconfigureerd worden tot het Litmus ChaosCenter, vanwaar men later eveneens experimenten zal kunnen uitvoeren.

Voer volgende stappen uit om Litmus op het systeem te installeren:

```
# Voeg de Litmus helm repository toe
$ helm repo add litmuschaos \
  https://litmuschaos.github.io/litmus-helm/

# Maak een namespace aan waaronder Litmus toegevoegd wordt
$ kubectl create ns litmus

# Installeer Litmus in de namespace litmus
$ helm install chaos litmuschaos/litmus --namespace=litmus

# Verifieer werking pods frontend, database en server
$ kubectl get pods -n litmus
```

Firewall regels configureren

Vooraleer men via de browser connectie kan maken met het Litmus ChaosCenter zullen eerst de nodige firewall regels moeten geconfigureerd worden. Om te weten te komen welke poorten open gezet moeten worden voert men commando **kubectl get svc -n litmus** uit.

Dit toont alle actieve services in de litmus namespace. Daar ziet men o.a. dat voor de eerder vernoemde frontend- en server pod een Nodeport service is geconfigureerd. Deze laten toe om de pod van buitenaf te betreden.

Voer o.b.v. info uit de output van vorig commando volgende stappen uit om de firewall te configureren. Hierbij is vooral het poortnummer in kolom Ports van belang. Deze poorten variëren echter bij elke installatie. Volgende commando's configureren firewall regels specifiek voor een GKE cluster en kunnen dus niet gebruikt worden in een lokale omgeving. Verder onderzoek hoe deze poorten te openen in een lokale omgeving is hierdoor nog nodig.

```
# Firewall regel die verkeer op poort frontend service toelaat.
# Gebruik frontend Service poortnummer (na dubbelpunt)
$ gcloud compute firewall-rules create frontend-service-rule \
  --allow tcp:[port]
```



```
# Firewall regel die verkeer op poort server service toelaat.  
# Gebruik één v.d. server Service poortnummers (na dubbelpunt)  
$ gcloud compute firewall-rules create server-service-rule \  
--allow tcp:[port]
```

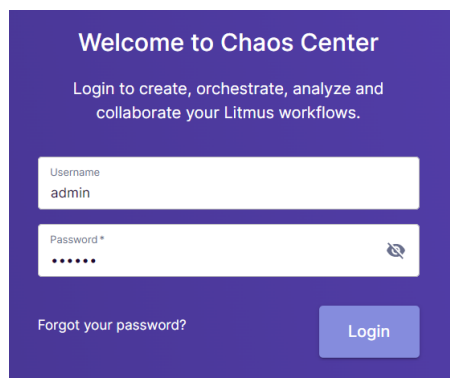
Bij elk van bovenstaande commando's zal men bij een succesvolle uitvoer de regels 'Creating firewall...working..Created' en 'Creating firewall...done' te zien krijgen.

Toegang configureren tot Litmus ChaosCenter

Om via de browser naar het Litmus ChaosCenter te gaan kan men gebruik maken van één van de externe IP-adressen van de nodes. Deze kan men bekomen door in de terminal het commando **kubectl get nodes -o wide** uit te voeren. Ook zal men het poortnummer nodig hebben van de frontend service, waar eerder een firewall regel voor geconfigureerd is. Gebruik **http://[node IP-adres]:[frontend service poort]** in de browser om toegang te krijgen tot Litmus ChaosCenter.

Wanneer men voor het eerst connecteert met het Litmus ChaosCenter kan men gebruik maken van de hieronder vermelde default credentials. Vervolgens zal men direct een nieuw wachtwoord moeten configureren alvorens de toegang te verkrijgen tot het Litmus ChaosCenter.

- user = admin
- wachtwoord = litmus



Figuur 6.9: login Litmus ChaosCenter

Later in dit onderzoek, vanaf Hoofdstuk 6.4.7, zal omschreven worden hoe een experiment op te zetten via deze GUI.

Nu de accountactivatie in Litmus ChaosCenter afgerond is zal men heel wat extra pods terugvinden in de namespace litmus. Dit kan men net zoals voordien controleren door in de terminal het commando **kubectl get pods -n litmus** uit te voeren. Controleer als de aanwezige pods allemaal operationeel zijn.

Een eerder gecreëerde cluster waarbij de resources per node te beperkt waren en géén

autoscaler actief was resulteerde in een falende pod die de status Pending bleef behouden. Door de Events van deze pod te bekijken kwam aan het licht dat onvoldoende CPU de oorzaak was. Vandaar bij voorgaand hoofdstuk in sectie 5.1.1 het nodige belang gehecht werd aan voldoende resources toe te wijzen aan de nodes in de cluster.

6.4.2 Litmus experimenten uitvoeren via de terminal

Vooraleer men experimenten kan uitvoeren moet men eerst volgende Litmus concepten begrijpen:

- **Chaos Experiment:** De generieke low-level code van een experiment waar men niks hoeft in te wijzigen.
- **Chaos Engine:** De parameters waarmee men een experiment specifiek gaat richten naar een bepaalde applicatie/pod/... m.a.w. de connectie tussen applicatie en ChaosExperiment die men via een YAML-definitie configureert.
- **Chaos Result:** Het resultaat van de uitvoer van een experiment.

In Bijlage B.1 kan men eerst een theoretische benadering vinden hoe men een experiment dient op te zetten om uitgevoerd te worden via de terminal. Nadien wordt beschreven hoe het eerste experiment 'pod-delete' opgezet kan worden. Vervolgens kan men nog een aantal andere uitgevoerde experimenten via de terminal terugvinden, alvorens over te gaan tot het uitvoeren van experimenten via de browser m.b.v. Litmus ChaosCenter.

6.4.3 Experiment 1: Nginx pod delete

De theoretische benadering hoe men m.b.v. Litmus een experiment kan opzetten via de terminal zal in dit hoofdstuk omgezet worden naar de praktijk. Het eerste experiment die aan bod komt is het vernietigen van een Nginx pod in de namespace demoapp1. In deze namespace is ook een Deployment met Apache pods actief, die gevrijwaard zal blijven van de impact van dit experiment door dit zorgvuldig te configureren in de ChaosEngine definitie.

Nota: Experiment 2 en 3 zijn uitbreidingen op dit experiment en zullen dus gebruik maken van de alreeds geconfigureerde permissies onder directory litmus-experiments/serviceaccounts/. Dit kan men eveneens zien in de ChaosEngine definitie, waar de naam onder parameter 'experiments' steeds pod-delete zal zijn.

De bedoeling van dit experiment is aantonen dat een pod vernietigen in Kubernetes opgevangen wordt door de ReplicaSet die bij een Deployment hoort. Deze zal er steeds voor zorgen dat het gewenste aantal pods van een Deployment verzekerd wordt. Bij creatie van de Nginx Deployment in demoapp1 werd in het commando aangegeven dat drie replicas moesten bestaan. Dit experiment zal slechts 1 willekeurige Nginx pod vernietigen.

De benodigde ChaosExperiment-objecten (zie Bijlage B.1) zijn alreeds aanwezig zowel in de namespace demoapp1 als demoapp2.

Via volgend stappenplan stelt men de permissies van het eerste pod-delete experiment en creëert men een ChaosEngine waarmee het experiment kan uitgevoerd worden:

1. Open volgende link via de browser: <https://litmuschaos.github.io/litmus/experiments/categories/contents/>
2. Ga in het linkermenu via de dropdownlist Kubernetes - Generic - Pod Chaos naar het experiment **Pod Delete**. Daar vindt men de beschrijving en configuratie van dit experiment.
3. Kopieer de YAML-definitie in sectie **Minimal RBAC configuration**.
4. Ga via de Cloud Shell terminal naar de directory `/litmus-experiments/servicaccounts/`
5. Maak een nieuwe file 'pod-delete-sa.yaml' aan en kleef de inhoud uit voorgaande stap hierin.
6. Verander bij elke parameter 'namespace:' de waarde default naar `demoapp1`, en sla vervolgens op.
7. Voer het bestand uit via commando **kubectl apply -f pod-delete-sa.yaml**. Dit zal volgende output genereren:

```
serviceaccount/pod-delete-sa created
role.rbac.authorization.k8s.io/pod-delete-sa created
rolebinding.rbac.authorization.k8s.io/pod-delete-sa created
```

8. Herhaal bovenstaande stappen 1 en 2. Ga naar sectie **Experiment Examples** en kopieer de inhoud van dit bestand.
9. Maak een YAML-bestand aan in de directory `litmus-experiments` en noem dit bv. `nginx-pod-kill.yaml`
10. Kleef de inhoud uit sectie **Experiment Examples** in dit YAML-bestand en wijzig de waarde van parameters `namespace` naar 'demoapp1', en `applabel` naar 'app=nginx', zodat het experiment specifiek gericht wordt op de Nginx pods van de demo-applicatie in namespace `demoapp1`.
11. Sla het bestand op en voer het experiment uit via **kubectl apply -f nginx-pod-kill.yaml**
12. Volg het experiment op via k9s of via commando's uit subsectie B.1: Een experiment uitvoeren.

Resultaat: Dit experiment toonde de werking van de ReplicaSet die binnen enkele seconden na het vernietigen van een willekeurige Nginx pod alreeds een nieuwe pod heeft gecreëerd. Om een voorgeconfigureerde YAML-definitie van dit experiment te raadplegen kan men volgende link gebruiken: Experiment 1: `nginx-pod-kill.yaml`

6.4.4 Experiment 2: Nginx pod delete met iteratie

In dit experiment wordt een parameter toegevoegd aan de bestaande ChaosEngine uit voorgaand experiment die ervoor zorgt dat de uitvoer elke tien seconden herhaald zal worden. Dezelfde YAML-definitie in de link op het eind van vorig experiment kan hierbij geraadpleegd worden, waarbij onderaan de nodige parameters nog uit commentaar gehaald moeten worden.

Open het YAML-bestand 'nginx-pod-kill.yaml', ga naar sectie 'env' en verleng de tijd van het experiment naar zestig seconden. Voeg eveneens een extra parameter 'CHAOS_INTERVAL' toe om de iteratie in te stellen. Zie onderstaand vb.:

- name: TOTAL_CHAOS_DURATION
value: '60'
- name: CHAOS_INTERVAL
value: '10'

Sla vervolgens op en start het experiment op dezelfde manier als voordien via **kubectl apply -f nginx-pod-kill.yaml**.

Men kan bij de uitvoer van het experiment zien via k9s dat elke tien seconden één willekeurige pod vernietigd wordt gedurende één minuut waardoor de ReplicaSet meermaals getriggerd zal worden om een nieuwe pod te creëren.

Om te zien als dit een impact heeft op de bereikbaarheid van de Nginx pods zal in volgend experiment een **HTTP-probe** toegevoegd worden aan de ChaosEngine definitie.

6.4.5 Experiment 3: Nginx pod delete met iteratie en probe

In dit experiment controleert men via een **HTTP-probe** als de applicatie bereikbaar is gedurende de uitvoer van het experiment. Via deze probe wordt een GET-request gestuurd naar een opgegeven IP-adres, in dit geval het extern IP-adres van de LoadBalancer (of NodePort) service van de Nginx pods. Door eveneens een HTTP-response code op te geven die men verwacht terug te krijgen zal kunnen gecontroleerd worden als de applicatie tijdig bereikbaar is.

De ChaosEngine-definitie van dit experiment kan men via volgende link raadplegen: Experiment 3: nginx-pod-kill-probed.yaml

Er zijn vier verschillende probes beschikbaar in Litmus, waarvan men een voorgeconfigureerde YAML-definitie kan terugvinden en toepassen in een ChaosEngine. Deze vindt men hier terug: <https://docs.litmuschaos.io/docs/concepts/probes/>

Men kan een HTTP-probe configureren via verschillende parameters die o.a. bepalen:

- hoe snel men de response verwacht
- hoeveel keer deze probe moet uitgevoerd worden
- hoeveel keer opnieuw mag geprobeerd worden wanneer de probe faalt
- ...

Kopieer de inhoud van dit bestand en sla dit lokaal op via de Cloud Shell terminal in een nieuw YAML-bestand genaamd 'nginx-pod-kill-probed.yaml'. Voer het experiment vervolgens uit via commando **kubectl apply -f nginx-pod-kill-probed.yaml** en volg opnieuw op via k9s.

Men kan de status van de HTTP-probe nadien controleren door het resultaat van het experiment op te vragen die bewaard wordt in een ChaosResult object. Om de exacte naam van dit object te weten te komen gebruikt men commando **kubectl get chaosresults -n appdemo1**.

Eens men de naam kent kan via commando **kubectl describe chaosresult [chaosresult-naam] -n appdemo1** het resultaat opgevraagd worden.

Een deel van de output, waarin aangetoond wordt dat de probe geslaagd is en de Nginx website nog bereikbaar is wanneer Nginx pods vernietigd worden, kan men hieronder zien:

```
--vorige output weggelaten--
Probe Status:
  Name:    check-frontend-access-url
  Status:
    Continuous:    Passed
  Type:           httpProbe
```

Dit experiment is handig om de responsetijd van een website te controleren wanneer deze te kampen krijgt met pods die plots vernietigd worden.

(Optioneel) Door onderaan in de ChaosEngine-definitie bij sectie 'env' de parameter 'PODS_AFFECTED_PERC' toe te voegen kan men een hoger percentage instellen van pods die getroffen worden door het experiment.

6.4.6 Experiment 4: Geheugen belasten van nginx pods

In dit experiment zal het geheugen belast worden van de nginx pods in de demo-applicatie in namespace demoapp1. Applicaties in een productieomgeving kunnen te maken krijgen met pieken in resourcegebruik, dit door verwachte maar eveneens onverwachte redenen. Vandaar het belangrijk is te testen hoe een applicatie reageert in deze omstandigheden, en welke manieren er in Kubernetes bestaan om hulp te bieden in zulke situaties.

Bij de demo-applicatie in demoapp1 is een kanttekening te maken aangezien deze oorspronkelijk opgezet is zonder een limiet op de resources die het mag gebruiken. Hierdoor zou een applicatie alle resources van een node kunnen aanspreken, maar dit is geen ideaal scenario. Zo is een pod ingesteld volgens 3 QoS klassen. Pods zonder geconfigureerde resource management vallen onder de klasse 'best effort' en komen hierdoor als eerste in aanmerking om vernietigd te worden wanneer de node resources beperkt worden. (S, 2020)

Om de pods in de demoapp1 namespace te configureren zodanig deze maximum 200 mCPU en 256MB gebruiken voert men volgende twee commando's uit (Kubernetes, 2022d):

```
$ kubectl set resources deployment nginx -n demoapp1 \
--limits=cpu=200m,memory=256Mi
```

```
$ kubectl set resources deployment apache -n demoapp1 \
--limits=cpu=200m,memory=256Mi
```

De ChaosEngine-definitie kan men raadplegen via volgende link: Experiment 4: nginx-pod-memory-hog.yaml

Men kan de inhoud uit bovenstaande link kopiëren en plaatsen in een nieuwe ChaosEngine in directory litmus-experiments met als naam 'nginx-pod-memory-hog.yaml'.

Dit experiment zal een nieuwe RBAC-configuratie vereisen, aangezien een ander ChaosExperiment nl. 'pod-memory-hog' wordt aangeroepen in de ChaosEngine. Alle benodigde configuratie kan men raadplegen via volgende link: <https://litmuschaos.github.io/litmus/experiments/categories/pods/pod-memory-hog/>

Men kan op een identieke manier zoals bij experiment 1 te werk gaan door een bestand 'pod-memory-hog-sa.yaml' in de subdirectory /litmus-experiments/serviceaccounts/ te creëren, de inhoud uit bovenstaande link in dit bestand te plaatsen en overal de namespace parameter aan te passen. Vervolgens voert men het bestand uit om de serviceaccounts en RBAC-configuratie te activeren.

Let op: In deze ChaosEngine moet men parameters configureren om de Container Runtime en het socket path te definiëren. Hier dient men op te geven de Container Runtime 'containerd' te gebruiken met verwijzing naar het socket path. Indien men dit niet zou doen dan faalt het experiment doordat één van de helper pods niet kan opstarten.

Voer het experiment uit via commando **kubectl apply -f nginx-pod-memory-hog.yaml**. Men kan dit experiment opvolgen via de Metrics Explorer in Google Cloud, waar men als metric kiest voor 'Memory usage' en groepeert op pods in de namespace demoapp1. Zo zal men de pieken zien verschijnen van zodra het experiment start.

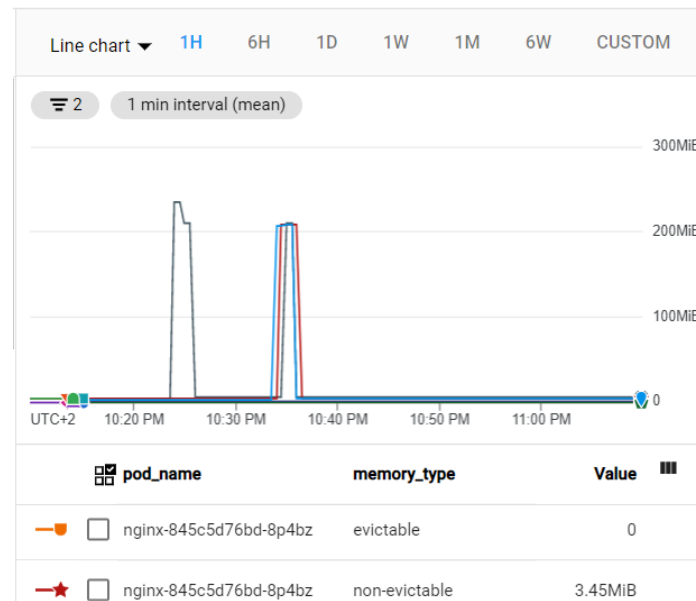
In vorig besproken chaos engineering tool Chaos Mesh kwam het Kubernetes object HorizontalPodAutoscaler alreeds aan bod in een soortgelijk experiment. Uitleg over wat een HPA doet kan men in Hoofdstuk 6.3.7 terugvinden.

Een vooraf gedefinieerde YAML-definitie voor deze HPA kan men raadplegen via volgende link: Experiment 4 oplossing: hpa.yaml

Gebruik volgend stappenplan om de HorizontalPodAutoscaler toe te passen:

1. Plaats de inhoud van de YAML-definitie 'hpa.yaml' uit bovenstaande link in een nieuw bestand genaamd 'hpa.yaml' in de directory litmus-experiments. In dit bestand kan men zien dat de threshold ingesteld is op 100 MB en dat er mag geschaald worden tot zes pods indien nodig. Zodra de threshold overschreden wordt zullen nieuwe pods gecreëerd worden. Zie Figuur 6.11 ter verduidelijking.
2. Activeer de HorizontalPodAutoscaler m.b.v. commando **kubectl apply -f hpa.yaml**.
3. Herhaal experiment 4 via **kubectl apply -f nginx-pod-memory-hog.yaml**.

Via k9s ziet men hoe drie nieuwe pods gecreëerd worden om de belasting mee te helpen



Figuur 6.10: Geheugen belasting nginx pods tot 200 MB

opvangen. Deze nieuwe Nginx pods zullen gevrijwaard blijven van de geheugenbelasting die het lopende experiment veroorzaakt en dus niet belast worden zoals de vooraf bestaande nginx pods.

NAME	PF	READY	RESTARTS	STATUS
apache-5bcb8b58cc-5wjxq	●	1/1	0	Running
apache-5bcb8b58cc-q5lxx	●	1/1	0	Running
apache-5bcb8b58cc-wbqgh	●	1/1	0	Running
nginx-845c5d76bd-6c455	●	1/1	0	Running
nginx-845c5d76bd-8p4bz	●	1/1	0	Running
nginx-845c5d76bd-99857	●	0/0	0	Pending
nginx-845c5d76bd-hhqhh	●	0/1	0	ContainerCreating
nginx-845c5d76bd-rhnwf	●	1/1	0	Running
nginx-845c5d76bd-v8sjh	●	0/1	0	ContainerCreating
nginx-pod-memory-hog-runner	●	1/1	0	Running
pod-memory-hog-39n048-ffxks	●	1/1	0	Running
pod-memory-hog-helper-cozpoq	●	1/1	0	Running
pod-memory-hog-helper-getnua	●	1/1	0	Running
pod-memory-hog-helper-zzfnmp	●	1/1	0	Running

Figuur 6.11: HorizontalPodAutoscaler creëert drie nieuwe nginx pods

6.4.7 Experimenten uitvoeren via Litmus ChaosCenter

Bovenstaande experimenten zijn tot nu toe allemaal uitgevoerd via de terminal. Hierdoor moet men echter verschillende manuele stappen doorlopen alvorens te kunnen overgaan tot de uitvoer van een experiment. Via de browser kan men deze experimenten ook uitvoeren in de GUI genaamd Litmus ChaosCenter, die in een voorgaand hoofdstuk 6.4.1 alreeds werd opgezet.

Ga via de browser naar Litmus ChaosCenter en vul de credentials in om toegang te krijgen. Zie het stappenplan in Bijlage B.2 om experiment 3 (= herhalende pod vernietiging met probe) te herhalen via de GUI.

Eens het stappenplan doorlopen is zal de workflow vervolgens opstarten waarin het experiment slechts één specifiek onderdeel is. In de workflow zal men eerst het experiment (m.a.w. het object ChaosExperiment uit de ChaosHub) installeren alvorens de ChaosEngine (= de configuratie van het experiment) uit te voeren. Na de uitvoer van het experiment start het laatste onderdeel van de workflow nl. de toegebrachte chaos ongedaan maken.

Tijdens de uitvoer van het experiment kan men via k9s zien dat er regelmatig pods van de Podtato-Head applicatie in de namespace 'litmusexperiments' verwijderd worden. Doordat deze pods via een Deployment opgezet zijn zal de ReplicaSet er voor zorgen dat het gewenste aantal pods in de configuratie steeds gerespecteerd blijft. De HTTP-probe zal controleren als de applicatie bereikbaar blijft gedurende het experiment.

Workflow herhalen/aanpassen

Wanneer men de configuratie van een uitgevoerde workflow wil aanpassen, of deze opnieuw wil uitvoeren, dan gaat men in het linkermenu bij Litmus Workflows naar het tabblad Scheduled.

Door uiterst rechts naast de workflow het menu te openen ziet men de optie **Rerun Schedule** om de workflow opnieuw uit te voeren.

Via de optie 'Save Template' kan men een wijziging aanbrengen aan de configuratie van de workflow. Hierbij wordt gevraagd een nieuwe naam aan de workflow te geven. Men kan een aanpassing namelijk NIET rechtstreeks uitvoeren in de bestaande workflow! De nieuwe aangepaste workflow zal men vervolgens aanspreken via optie 'Schedule a workflow'. In het tabblad **Choose a workflow** kan men vervolgens kiezen voor 'Create a new workflow by cloning an existing workflow'. Daar zal de aangepaste versie van de bestaande workflow terug te vinden zijn. Aangezien alle configuraties alreeds opgenomen zijn in deze kan men direct doorgaan tot het uitvoeren van de aangepaste workflow.

Extra functionaliteit in Litmus ChaosCenter

Andere Litmus experimenten die alreeds uitgevoerd werden in de terminal kunnen op soortgelijke manier via een workflow opgezet worden. Men kan zelfs verschillende experimenten gelijktijdig uitvoeren in een workflow via de knop **Edit Sequence** in tabblad **Tune Workflow** (zie stap 5 in bovenstaand stappenplan).

In het linkermenu kan men bij **Observability** o.a. statistieken raadplegen van alreeds uitgevoerde experimenten, een monitoring dashboard toevoegen in de GUI ...

Deze extra opties zijn niet verder onderzocht doordat meerdere experimenten gelijktijdig uitvoeren geen meerwaarde biedt in deze context, en een monitoring dashboard alreeds beschikbaar is via het Google Cloud Platform.

6.4.8 Conclusie Litmus

Het installeren van Litmus verliep vlot via de package manager Helm. Ook het opzetten van de GUI ChaosCenter lukte zonder problemen. De documentatie mocht wel duidelijker zijn omtrent de minimum vereisten qua CPU en geheugen per node. Zo werd na eerste installatiepogingen duidelijk dat de Litmus pods vrij veel resources nodig hadden om operationeel te kunnen zijn.

De verschillende objecten die Litmus gebruikt om een experiment op te zetten nl. ChaosExperiment, ChaosEngine, ServiceAccount, RBAC-regels ... maken deze chaos engineering tool vrij ingewikkeld om aan te leren. Ook is de documentatie die hulp kan bieden bij de experimenten soms moeilijk te vinden.

Litmus heeft net zoals Chaos Mesh het voordeel van veel experimenten in het aanbod te hebben. Experimenten opzetten via de terminal is vrij omslachtig. Men moet manueel de nodige experimenten vooraf installeren, de nodige permissies configureren, de ChaosEngine manueel opzetten ... Ook wanneer een experiment niet naar wens verloopt en men dit vervolgens wil troubleshooten, moet men verschillende objecten analyseren vb. helper pods onderzoeken, uitvoer van chaosengine object bestuderen ... Hierdoor is het duidelijk dat Litmus niet ontworpen is om via de terminal experimenten uit te voeren.

Bovenstaand proces wordt makkelijker wanneer experimenten opgezet worden via het Litmus ChaosCenter. Experimenten worden bij de start van een workflow steeds geïnstalleerd, men hoeft geen YAML-definitie op te stellen maar gewoon parameters te configureren, men kan indien nodig logs raadplegen na afloop van een experiment om troubleshooting eenvoudiger te maken ...

Litmus is een waardige tool om chaos experimenten uit te voeren op applicaties in een Kubernetes cluster, maar heeft een steile leercurve vooraleer men er daadwerkelijk mee aan de slag kan. De documentatie omtrent de installatie en het opzetten van experimenten is er wel, maar vereist wat zoekwerk. Ook het hoge aantal resources die de Litmus pods nodig hebben in vergelijking met voorgaand onderzochte tool Chaos Mesh kan als nadelig aanschouwd worden.

7. Conclusie

Gedurende dit onderzoek werd een antwoord gezocht op de vragen uit Hoofdstuk 1.2. Volgende antwoorden kan men aanschouwen als de conclusie van dit onderzoek:

1. Biedt het een meerwaarde om een lokale Kubernetes cluster op te zetten m.b.v. automatisatietools ten opzichte van een Kubernetes cluster via Google Cloud op te zetten?

Verschillende lokale Kubernetes clusters werden opgezet tijdens dit onderzoek. Een single-node Minikube cluster kan snel en eenvoudig opgezet worden maar biedt geen meerwaarde om het inzicht in de werking van Kubernetes te verruimen. Eveneens kan men geen experimenten toepassen die gericht zijn op een node, aangezien dit de enige node in de omgeving zou treffen. Een multi-node cluster opzetten via Kubeadm is een tijdrovende manuele taak, maar geeft wel het nodige inzicht welke stappen nodig zijn om een cluster operationeel te krijgen. Deze manuele taken konden grotendeels opgelost worden via de geautomatiseerde setup m.b.v. de Ansible playbook in Kubespray, maar deze had bijna een half uur nodig om de cluster op te zetten wat opnieuw als tijdrovend kan aanschouwd worden. Een monitoring platform opzetten in een lokale omgeving is om deze reden niet verder onderzocht.

De snelheid en het gemak waarmee een cluster opgezet wordt in Google Cloud is ten opzichte van een lokale setup een enorm verschil. Een cluster opzetten in Google Cloud duurt slechts enkele minuten. Men kan extra functionaliteit toevoegen tijdens de setup zoals autoscaling, waardoor nodes automatisch kunnen schalen. Eveneens is er de aanwezigheid van een monitoring dashboard in Google Cloud, waardoor sommige experimenten zoals het belasten van het geheugen beter opgevolgd kunnen worden. Google Cloud biedt 300 dollar aan gratis krediet aan gedurende negentig dagen. Hierdoor is het mogelijk deze setup te gebruiken in een curriculum systeem-en netwerkbeheer.

2. Welke chaos engineering tool die in dit onderzoek aan bod komt is het meest geschikt om experimenten uit te voeren op een applicatie in een Kubernetes cluster?

Het antwoord op deze vraag is minder eenvoudig want elke onderzochte tool heeft namelijk zijn voor- en nadelen. Chaos Toolkit en Litmus opzetten ging vrij vlot, maar de GUI van Chaos Mesh bracht tijdens de setup de nodige problemen met zich mee. Chaos Toolkit kan enkel gebruikt worden via de terminal en heeft een beperkt aanbod qua experimenten ten opzichte van de andere onderzochte tools. De tool moet uitgebreid worden via verschillende extensies/plugins om meer functionaliteit te creëren. Het voordeel bij Chaos Toolkit is dat de structuur, de uitvoer en het resultaat van de experimenten in de terminal vrij duidelijk zijn. Deze tool is eveneens goed gedocumenteerd. Puur bekeken op experimenten uitvoeren via de terminal zou de voorkeur naar deze tool gaan.

Chaos Mesh en Litmus hebben een breed gamma aan experimenten die zowel uitgevoerd kunnen worden via de terminal als via de GUI. Ondanks experimenten uitvoeren in de terminal weinig meerwaarde biedt ligt de moeilijkheidsgraad bij Litmus om dit te realiseren veel hoger dan bij Chaos Mesh. De experimenten bij Chaos Mesh zijn beter gedocumenteerd dan bij Litmus en de leercurve bij Chaos Mesh is ook minder steil. Litmus heeft bovendien meer resources nodig om operationeel te kunnen zijn tegenover Chaos Mesh. Op vlak van functionaliteit bij de uitvoer van experimenten is Litmus wel beter voorzien dan Chaos Mesh en Chaos Toolkit. Als documentatie en leercurve de doorslag kunnen geven in de vergelijking geniet Chaos Mesh de voorkeur. Het combineren van deze tools is echter perfect mogelijk. In dat opzicht kan men eventueel het beste van beiden combineren en zowel Chaos Toolkit als Chaos Mesh gebruiken om experimenten op te zetten.

3. Welke chaos engineering experimenten zijn relevant om een beter inzicht te creëren in de werking van Kubernetes?

Dit onderzoek is vertrokken vanuit het standpunt om een betere basiskennis in Kubernetes te creëren via deze experimenten. Het had dan ook weinig zin om node-loos complexe applicaties op te zetten die een gevorderde kennis van Kubernetes zouden vereisen. Door de applicatie(s) simpel te houden konden alreeds enkele relevante zaken omtrent de werking van Kubernetes aangetoond worden zoals o.a.:

- het vernietigen van pods in een Deployment triggert de ReplicaSet om nieuwe pods te creëren.
- het vernietigen van pods in de Podtato-Head applicatie bracht aan het licht dat het voordeliger is om meerdere pods via een Deployment te creëren, zodat een applicatie bereikbaar blijft wanneer een pod getroffen wordt.
- het simuleren van netwerkproblemen toonde het belang aan van de communicatie tussen Services die verantwoordelijk zijn voor de bereikbaarheid van pods verspreid over verschillende nodes.
- het geheugen belasten van pods kan opgevangen worden via een HorizontalPodAutoscaler die extra pods creëert om deze belasting op te vangen.
- het simuleren van een nodefaling toonde aan dat Kubernetes alle pods op de getroffen node kon verhuizen.

4. Heeft dit onderzoek geleid tot nieuwe vragen die uitnodigen tot verder onderzoek?

Door de tijdsdruk gedurende dit onderzoek zijn mogelijk interessantere lokale setups van een Kubernetes cluster die minder tijdrovend zijn niet aan bod gekomen. Het opzetten van een monitoring platform in een lokale omgeving is evenmin aan bod gekomen en vereist verder onderzoek. Eveneens zijn er nog interessante open-source chaos engineering tools niet onderzocht, die mogelijk beter geschikt zijn voor educatieve doeleinden.

Zowel de zoektocht naar een vlottere manier om een lokale omgeving op te zetten, alsook een geschikte chaos engineering tool te vinden om te gebruiken in deze omgeving, nodigt zich uit tot verder onderzoek.

A. Onderzoeksvoorstel

Het onderwerp van deze bachelorproef is gebaseerd op een onderzoeksvoorstel dat vooraf werd beoordeeld door de promotor. Dat voorstel is opgenomen in deze bijlage.

A.1 Introductie

Netflix is de pionier wat betreft chaos engineering. Een drie dagen durende outage door een grootschalige database corruptie in 2008 zorgde ervoor dat zij hun diensten, toen nog gehuisvest in verticale server racks in hun datacenter, migreerden naar de AWS cloud. Tijdens deze zeven jaar durende migratie naar de cloud herbouwden zij stap voor stap de architectuur van hun monolitische applicatie tot honderden microservices.

Om de weerbaarheid van de cloud infrastructuur te testen ontwikkelden ze tijdens de migratie ook verschillende tools, nu beter bekend als chaos engineering tools. Op deze manier konden zij de zwaktes in het systeem vroegtijdig opsporen en verbeteren, om de uptime zo hoog mogelijk te houden. Hun bevindingen en tools werden kort nadien gepubliceerd en lagen aan de basis van een nieuwe mindset in disaster recovery en business continuity. (Yury Izrailevsky, 2016)

De groeiende complexiteit van systemen en de kwalijke gevolgen die een falen kunnen veroorzaken leiden tot dit onderzoek waarin gezocht wordt om open source chaos engineering tools en experimenten op te zetten om een Kubernetes cluster te testen, met de bedoeling deze in het toekomstig curriculum systeem- en netwerkbeheer te verwerken.

A.2 State-of-the-art

Er zijn tegenwoordig meerdere open source chaos engineering platformen beschikbaar. Gremlin, ChaosMesh, Litmus ... zijn enkele van de bekendste die tools aanbieden om chaos experimenten uit te voeren. Met behulp van deze tools kan men proactief zwaktes in een systeem opsporen en verbeteren, om de weerbaarheid ervan in productie te verhogen.

Chaos engineering wint aan populariteit en er is een groeiende diversiteit in de teams die dit toepassen. Wat begon als een engineering oefening werd al snel opgepikt door SRE teams. Vandaag zijn er vele platform-, infrastructuur-, operations- en applicatie-ontwikkelingsteams die deze tools toepassen om de betrouwbaarheid van hun systeem en applicaties te verhogen. De positieve effecten van chaos engineering toe te passen zijn o.a. een toegenomen uptime, minder tijd nodig om problemen te detecteren en te herstellen, minder bugs die in productie terecht komen ... (Kolton Andrus, 2021)

A.3 Methodologie

De experimenten zullen opgezet en uitgevoerd worden in een lokale virtuele omgeving, waarin verschillende virtuele machines in hetzelfde netwerk via Vagrant en Ansible opgezet worden. Deze virtuele machines gaan verschillende microservices bevatten en samen ondergebracht worden als nodes in een Kubernetes cluster.

Experimenten opzetten gebeurt steeds in 4 stappen (Pawlikowski, 2020):

1. Men neemt een (set van) variabele(n) en een betrouwbare manier om deze te meten.
2. Men definieert een normaal gedragspatroon voor deze variabelen (= steady state).
3. Men stelt een hypothese op omtrent het gedragspatroon van de variabele(n) bij een bepaalde gebeurtenis.
4. Men voert het experiment uit en analyseert het resultaat.

Gebruik makende van verschillende open source chaos engineering tools, waaronder o.a. ChaosToolkit, Powerful Seal ... zullen meerdere experimenten toegepast worden op een Kubernetes cluster en via een open source monitoring tool opgevolgd worden. Deze experimenten veroorzaken falingen zoals het uitvallen van een server, onbeschikbaarheid van DNS, latency injection, resource exhaustion, enz. Het doel van deze experimenten is om inzicht te verwerven in hoe een systeem/applicatie zich gedraagt wanneer falingen gebeuren en eveneens oplossingen te voorzien om deze in de toekomst te kunnen vermijden.

A.4 Verwachte resultaten

Het resultaat van dit onderzoek zal een aanbeveling zijn naar een geschikte open source tool en enkele chaos engineering experimenten die kunnen opgezet worden die in lijn liggen met het curriculum systeem- en netwerkbeheer.

Deze aanbeveling zal gemaakt worden op basis van criteria die belangrijk zijn voor het onderwijs:

1. Is er voldoende documentatie beschikbaar om de tool op te zetten?
2. Wat is de kost van de tool?
3. Welke voorkennis is nodig om de tool te kunnen gebruiken?

De manier waarop deze tool en monitoring opgezet wordt kan afwijken van de oorspronkelijke methodologie. Op basis van het resultaat van de verschillende experimenten zullen ook aanbevelingen gegeven worden hoe men het systeem kan verbeteren om een zo hoog mogelijke uptime te garanderen.

A.5 Verwachte conclusies

Het opzetten van verschillende types servers en microservices in Docker containers komt alreeds aan bod in het huidige curriculum. Monitoring toepassen op een Kubernetes cluster en vervolgens Chaos Engineering experimenten erop uitvoeren kan een waardige bijdrage leveren. Waar momenteel de focus ligt in het opzetten en troubleshooten, zal door de aanbevolen chaos engineering experimenten uit te voeren een beter inzicht verworven worden in de werking van een systeem/applicatie en hoe men proactief problemen kan opsporen en aanpakken.

Lijst van figuren

2.1	hypervisor types (Vembu, 2019)	18
2.2	evoloutie in softwarearchitectuur (Sanjaya, 2020)	20
2.3	k8s clusterarchitectuur (Collabnix, 2019)	23
6.1	Podtato-Head applicatie	54
6.2	k9s uitvoer van Chaos Toolkit experiment 1	58
6.3	Probe faalt bij uitvoer Chaos Toolkit experiment 2	60
6.4	http probe faalt bij uitvoer Chaos Toolkit experiment 4	61
6.5	Chaos Mesh Experimenten per groep	69
6.6	Configureren van experiment in Chaos Dashboard	70
6.7	opvragen van Events van getroffen pod via k9s	73
6.8	GKE workloads: belasten van geheugen Nginx pods	76
6.9	login Litmus ChaosCenter	79
6.10	Geheugen belasting nginx pods tot 200 MB	85
6.11	HorizontalPodAutoscaler creëert drie nieuwe nginx pods	85

B. Bijlagen

B.1 Litmus experimenten opzetten via de terminal

Experimenten installeren op het systeem

De generieke code van een experiment vindt men terug in een Chaos Experiment object en worden bewaard in de Litmus Chaos Hub. Alle mogelijke Litmus experimenten kan men hier terugvinden, geklasseerd in verschillende categorieën. (ChaosHub, 2022)

Men dient eerst de nodige categorie van experimenten te installeren in dezelfde namespace(s) van de demo-applicatie(s). De experimenten die via de terminal uitgevoerd worden vallen allen onder categorie **generic/all-experiments**.

Om deze experimenten te installeren voert men volgend commando uit voor zowel de namespaces demoapp1 en demoapp2:

```
$ kubectl apply -f \
https://hub.litmuschaos.io/api/chaos/2.7.0?file=charts/generic\
/experiments.yaml -n demoapp[1 en 2]
```

Permissies van een experiment instellen

Het uitvoeren van deze experimenten moet eerst voorafgegaan worden door de nodige permissies te configureren via **Role Based Access Control (RBAC)**. Hierdoor verkrijgt het later beschreven experiment in de ChaosEngine de toestemming om uitgevoerd te worden binnen een bepaalde namespace. Permissies configureert men bij elk experiment wanneer men dit via de terminal uitvoert.

Een vooraf gedefinieerde RBAC-configuratie per experiment kan men terugvinden via volgende link: <https://litmuschaos.github.io/litmus/experiments/categories/contents/>

De YAML-definitie in sectie **Minimal RBAC configuration** slaat men op in een bestand onder een nieuw gecreëerde subdirectory (bv. `serviceaccounts`) in de alreeds bestaande directory `litmus-experiments`. Men hoeft enkel nog de waarde van elke parameter 'namespace' in dit bestand aanpassen naar de naam van de namespace waarin het experiment uitgevoerd wordt m.a.w. de namespace `demoapp1` of `demoapp2`.

Via commando **kubectl apply -f [bestand].yaml** maakt men vervolgens de nodige bestanden aan om de permissies te activeren.

Een ChaosEngine definiëren

Nu de nodige permissies ingesteld zijn kan men een ChaosEngine definiëren waarin de link wordt gelegd naar het ChaosExperiment. Ook deze definitie kan men terugvinden in eerder vermelde link, in de sectie **Experiment Examples**. Kopieer ook hier de inhoud en sla deze op in een nieuw bestand in de directory `litmus-experiments`. (Experiments, 2022).

Men zal zien dat in de ChaosEngine definitie de ServiceAccount aangesproken wordt. De parameter `jobCleanUpPolicy` zal er voor zorgen dat de 'helper pods' die Litmus lanceert tijdens het experiment terug verwijderd zullen worden na afloop.

Alle mogelijke parameters in aanvulling van een experiment kan men terugvinden in de sectie **Experiment tunables**. Zo kan men o.a. beslissen via parameter `CHAOS_INTERVAL` een experiment in iteraties uit te voeren, via parameter `PODS_AFFECTED_PERC` het percentage getroffen pods in te stellen ...

Een experiment uitvoeren

Vervolgens is men klaar om een experiment uit te voeren. Dit kan men doen via commando **kubectl apply -f [experiment-naam].yaml**. Dit commando zal géén output genereren.

Tijdens de uitvoer kan men via de UI monitoring tool `k9s` het experiment live opvolgen. Hierin zal men eveneens zien dat in de `demoapp[1/2]` namespace tijdelijke helper pods gelanceerd worden om de uitvoer van het experiment te faciliteren.

Enkele handige commando's om de uitvoer tijdens-, of het resultaat na een experiment te controleren zijn:

```
# ChaosEngine object(en) ophalen
$ kubectl get chaosengine -n demoapp[1|2]

# Het verloop van experiment tonen door het
# ChaosEngine object te beschrijven
$ kubectl describe chaosengine [naam] -n demoapp[1|2]
```

```
# ChaosResult object(en) ophijsten
$ kubectl get chaosresult -n demoapp[1/2]

# Het resultaat van experiment tonen door het
# ChaosResult object te beschrijven
$ kubectl describe chaosresult [naam] -n demoapp[1/2]
```

B.2 Stappenplan experiment 3 opzetten in ChaosCenter

1. Ga in het linkermenu naar **Litmus Workflows** en klik vervolgens op 'Schedule a workflow'
2. In tabblad **Choose Agent**: selecteer de (enige) Self-Agent die hier aanwezig is en klik op Next. Een self-agent is een benaming voor de cluster waarop we het experiment willen uitvoeren.
3. In tabblad **Choose a workflow**: kies voor 'Create a new workflow using the experiments from ChaosHubs' en klik op Next.
4. In tabblad **Workflow Settings**: Geef de workflow een naam naar wens (vb. iterated-podtato-head-pod-kill) en klik op Next.
5. In tabblad **Tune workflow**: kies rechtsboven voor 'Add a new experiment'. Kies in de lijst die vervolgens getoond wordt voor 'generic/pod-delete' en bevestig via Done.
6. Men ziet de naam van het experiment in de workflow verschijnen. Rechts naast de naam klikt men het potlood aan om de nodige aanpassingen in te stellen.
7. In sectie **General**: verander niks en klik op Next.
8. In sectie **Target Application**:
 - (a) open de dropdownlist 'appns' (app namespace) en selecteer de namespace 'demoapp2', waar de podtato-head applicatie actief is.
 - (b) open de dropdownlist 'applabel' en kies voor 'app.kubernetes.io/name=podtato-head'.
 - (c) Ga verder door op Next te klikken.
9. In sectie **Define the steady state**:
 - (a) kies voor 'Add a new probe'.
 - (b) geef de probe een naam naar wens vb. application-access
 - (c) kies als probe type 'http' m.a.w. bereikbaarheid van URL controleren
 - (d) kies als probe mode 'Continuous' m.a.w. gedurende heel het experiment
 - (e) stel de **Probe Properties** in m.a.w. hoe vaak gecontroleerd moet worden als de opgegeven URL bereikbaar is a.d.h.v. verschillende parameters.
 De Podtato-Head applicatie bevat meerdere pods waarvan één pod nl. 'podtato-head-entry' de toegang tot de applicatie voorziet via de Service LoadBalancer (of NodePort). Wanneer het experiment deze pod zou vernietigen kan de http-probe hierdoor falen aangezien de applicatie tijdelijk onbereikbaar wordt. Vandaar moet men rekening houden dat voldoende tijd ingesteld wordt bij 'interval' om de kans op slagen tijdens het opnieuw proberen van de probe te

verhogen.

- (f) stel de **Probe Details** in m.a.w. de URL van de PodTato-Head applicatie, hoe deze getest wordt nl. via een GET-request, hoelang het mag duren om de response te krijgen op deze request en welke HTTP status code terug verwacht wordt, in dit geval code 200 (OK).

Een metric om de laadsnelheid van een pagina te volgen is de **TTFB of time to first byte**. Men kan dit vooraf controleren door naar de applicatie te surfen, de Developer Tools (F12) te openen en naar tabblad Network te gaan. Daar voert men een reload uit via CTRL + F5 en kan men zien hoelang het duurt van request tot eerste byte van de response. (Mensink, 2022)

- (g) Bevestig alle configuraties onderaan via Done.
10. In sectie **Tune Experiment**:
 - (a) stel 'Total chaos duration' in op 120 seconden m.a.w. experiment duurt 2 minuten
 - (b) stel 'Chaos interval' in op 10 m.a.w. elke 10 seconden wordt een pod verwijderd
 - (c) stel 'Force' in op false m.a.w. graceful termination (Dinesh, 2018)
 - (d) Bevestig configuratie door op 'Finish' te klikken.
 11. In sectie **Advanced options to tune workflow**: activeer 'Cleanup Chaos Workflow Pods' zodat na het uitvoeren van het experiment, de nodige helper pods ook terug verwijderd worden. Bevestig vervolgens via 'Save Changes'.
 12. Klik rechtsonderaan op Next om alle configuraties te bevestigen en naar het volgende tabblad te gaan.
 13. In tabblad **Reliability Score**: staat standaard op 10, is niet van belang voor de uitvoer van experimenten. Klik op Next om door te gaan.
 14. In tabblad **Choose a chaos schedule**: kies voor 'Schedule now' en klik op Next. Men kan hier ook opteren een 'Recurring Schedule' te creëren die het experiment op vaste tijdstippen vb. elk uur, elke dag, elke maand ... zal uitvoeren.
 15. In tabblad **Summary**: Bevestig de configuratie rechtsonderaan via Finish. Bevestig nogmaals via 'Go to workflow'.

Bibliografie

- Alletto, J. (2021, oktober 23). *OOMKilled: Troubleshooting Kubernetes Memory Requests and Limits*. <https://www.containiq.com/post/oomkilled-troubleshooting-kubernetes-memory-requests-and-limits>
- Bailey, R. (2016, juni 17). *Are there issues with running user pods on a Kubernetes master node?* <https://stackoverflow.com/questions/37873164/are-there-issues-with-running-user-pods-on-a-kubernetes-master-node>
- Basiri, A., Behnam, N., de Rooij, R., Hochstein, L., Kosewski, L., Reynolds, J. & Rosenthal, C. (2016). *Chaos Engineering*. <https://netflixtechblog.com/5-lessons-weve-learned-using-aws-1f2a28588e4c>
- Biradar, S. (2019, juli 31). *What is kubectrl?* <https://github.com/collabnix/dockerlabs/blob/master/kubernetes/beginners/what-is-kubectrl.md>
- ChaosHub. (2022, april 1). *One Stop for your Chaos Engineering Experiments*. <https://hub.litmuschaos.io/>
- ChaosMesh. (2021, juli 9). *Chaos Mesh QA*. <https://chaos-mesh.org/blog/chaos-mesh-q&a/>
- ChaosMesh. (2022a, maart 28). *Manage User Permissions*. <https://chaos-mesh.org/docs/manage-user-permissions/>
- ChaosMesh. (2022b, mei 9). *Send HTTP Requests on Workflow*. <https://chaos-mesh.org/docs/send-http-request-on-workflow/>
- ChaosMesh-documentation. (2022a, april 1). *Environment preparation*. <https://chaos-mesh.org/docs/quick-start/>
- ChaosMesh-documentation. (2022b, april 8). *Install Chaos Mesh using Helm*. <https://chaos-mesh.org/docs/production-installation-using-helm/#install-chaos-mesh-using-helm>
- ChaosToolkit. (2022a, mei 19). *Chaos Engineering Concepts in the Chaos Toolkit*. <https://chaostoolkit.org/reference/concepts/>

- ChaosToolkit. (2022b, mei 17). *chaostoolkit-kubernetes plugin*. <https://github.com/chaostoolkit/chaostoolkit-kubernetes>
- ChaosToolkit. (2022c, mei 22). *Drain nodes*. https://chaostoolkit.org/drivers/kubernetes/#drain_nodes
- ChaosToolkit. (2022d, mei 18). *How to Install the Chaos Toolkit*. <https://chaostoolkit.org/reference/usage/install/>
- ChaosToolkit. (2022e, mei 22). *Http provider*. <https://chaostoolkit.org/reference/api/experiment/#http-provider>
- ChaosToolkit. (2022f, mei 22). *Pods in phase*. https://chaostoolkit.org/drivers/kubernetes/#pods_in_phase
- ChaosToolkit. (2022, mei 22). *Uncordon node*. https://chaostoolkit.org/drivers/kubernetes/#uncordon_node
- CloudNativeLandscape. (2022, mei 1). *CNCF Cloud Native Interactive Landscape*. <https://landscape.cncf.io/card-mode?category=chaos-engineering&grouping=category>
- CNCF. (2022a, februari 16). *Chaos Mesh moves to the CNCF Incubator*. <https://www.cncf.io/blog/2022/02/16/chaos-mesh-moves-to-the-cncf-incubator/>
- CNCF. (2022b, mei 1). *Graduated and incubating projects*. <https://www.cncf.io/projects/>
- CNCF. (2022c, januari 11). *LitmusChaos becomes a CNCF incubating project*. <https://www.cncf.io/blog/2022/01/11/litmuschaos-becomes-a-cncf-incubating-project/>
- Collabnix. (2019, juli 31). *What is K8s made up of?* <https://github.com/collabnix/dockerlabs/blob/master/kubernetes/beginners/what-is-kubernetes/README.md#what-is-k8s-made-up-of>
- DeFabia, J. (2022, mei 22). *kubectl cordon*. https://jamesdefabia.github.io/docs/user-guide/kubectl/kubectl_cordon/
- Dinesh, S. (2018, mei 18). *Kubernetes best practices: terminating with grace*. <https://cloud.google.com/blog/products/containers-kubernetes/kubernetes-best-practices-terminating-with-grace>
- Docker. (2021). *Install Docker Engine on Ubuntu*. <https://docs.docker.com/engine/install/ubuntu/#install-using-the-convenience-script>
- Eliot, S. (2019, maart 1). *PRINCIPLES OF CHAOS ENGINEERING*. https://github.com/chaos-eng/chaos-eng.github.io/blob/master/content/_index.en.md
- Experiments, L. (2022, april 1). *Pod Delete*. <https://litmuschaos.github.io/litmus/experiments/categories/pods/pod-delete/#experiment-examples>
- Farcic, V. & Pope, D. (2020, juni 19). *Kubernetes Chaos Engineering With Chaos Toolkit And Istio*. <https://www.devopstoolkitseries.com/posts/chaos/>
- Gavant, J. (2022, april 1). *Deliver podtato-head using Helm*. <https://github.com/podtato-head/podtato-head/tree/main/delivery/chart>
- GKE. (2022, mei 1). *Cluster Autoscaler*. <https://cloud.google.com/kubernetes-engine/docs/concepts/cluster-autoscaler>
- Google-Cloud. (2022, april 29). *Google Cloud Free Program*. <https://cloud.google.com/free/docs/gcp-free-tier>
- Holt, A. & Huang, C.-Y. (2018). *Embedded Operating Systems: A Practical Approach*. <https://doi.org/https://doi.org/10.1007/978-3-319-72977-0>
- K9s. (2022, april 1). *K9s Installation*. <https://k9scli.io/topics/install/>
- Keao, Y. (2021, augustus 5). *Admission webhook "vauth.kb.io" denied the request*. <https://github.com/chaos-mesh/chaos-mesh/issues/2187>

- Kolton Andrus, M. F. (2021). *State of Chaos Engineering report* (onderzoeksrap.). Gremlin. <https://www.gremlin.com/state-of-chaos-engineering/2021/?ref=blog>
- Kubeadm. (2021, augustus 17). *kubeadm/README.md*. <https://github.com/kubernetes/kubeadm/blob/main/README.md>
- Kubernetes. (2022a). *Creating a cluster with kubeadm*. <https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/create-cluster-kubeadm/#initializing-your-control-plane-node>
- Kubernetes. (2022b, april 19). *Installing kubeadm*. <https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/#before-you-begin>
- Kubernetes. (2022c, januari 10). *Pods*. <https://kubernetes.io/docs/concepts/workloads/pods/>
- Kubernetes. (2022d, april 18). *Resource Management for Pods and Containers*. <https://kubernetes.io/docs/concepts/configuration/manage-resources-containers/>
- Kubespray. (2022a, april 14). *Deploy a Production Ready Kubernetes Cluster*. <https://github.com/kubernetes-sigs/kubespray/blob/master/README.md>
- Kubespray. (2022b, maart 28). *Kubespray requirements*. <https://github.com/kubernetes-sigs/kubespray#requirements>
- Litmus-docs. (2022, april 27). *Getting started - Installation*. <https://docs.litmuschaos.io/docs/getting-started/installation>
- Mannambeth, M. (2021, oktober 20). *Kubernetes for the absolute beginners*. <https://www.udemy.com/course/learn-kubernetes/>
- Matykevich, S. (2018, juni 28). *A Multitude of Kubernetes Deployment Tools: Kubespray, kops, and kubeadm* (S. Turol, C. Gutierrez & S. Matykevich, Red.). <https://www.altoros.com/blog/a-multitude-of-kubernetes-deployment-tools-kubespray-kops-and-kubeadm/>
- MDN. (2022, april 28). *HTTP response status codes*. <https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>
- Mensink, M. (2022, januari 18). *Wat is Time To First Byte (TTFB) en hoe verbeter je deze?* <https://www.hypernode.nl/blog/time-to-first-byte-ttfb/>
- Minikube. (2022, februari 24). *Minikube documentation*. <https://minikube.sigs.k8s.io/docs/>
- Mukul, M. (2020, augustus 24). *Kubernetes Create a new token and join command to rejoin/add worker node*. <https://monowar-mukul.medium.com/kubernetes-create-a-new-token-and-join-command-to-rejoin-add-worker-node-74bbe8774808>
- Osnat, R. (2020, januari 10). *A Brief History of Containers: From the 1970s Till Now*. <https://blog.aquasec.com/a-brief-history-of-containers-from-1970s-chroot-to-docker-2016>
- Pawlikowski, M. Why SREs can't afford to NOT do Chaos Engineering. In: 2020, december 8. <https://www.usenix.org/conference/srecon20americas/presentation/pawlikowski>
- Power, O. (2019). *An introduction to CNI*. <https://kubernetes.io/docs/setup/production-environment/tools/cni-plugins/#an-introduction-to-cni>
- Ratis, P. (2022, maart 1). *Awesome Chaos Engineering: Notable Tools*. <https://github.com/dastergon/awesome-chaos-engineering#notable-tools>
- RedHat-Ansible. (2022, mei 23). *What is an Ansible playbook?* <https://www.redhat.com/en/topics/automation/what-is-an-ansible-playbook>

- Richardson, C. (2015, mei 19). *Introduction to Microservices*. <https://www.nginx.com/blog/introduction-to-microservices/>
- S, T. (2020, januari 27). *KUBERNETES: WHY IS IT SO IMPORTANT TO SET UP SYSTEM RESOURCE MANAGEMENT?* <https://systemadminspro.com/kubernetes-why-is-it-so-important-to-set-up-system-resource-management/>
- Sanjaya, H. (2020, maart 11). *Monolith vs Microservices*. <https://medium.com/hengky-sanjaya-blog/monolith-vs-microservices-b3953650dfd>
- Simic, S. (2020, april 30). *How to Install Minikube on Ubuntu 18.04 / 20.04*. <https://phoenixnap.com/kb/install-minikube-on-ubuntu>
- Singh, B. (2020, december 26). *Kubernetes : The Rise of Containers and Microservices*. <https://medium.com/nerd-for-tech/kubernetes-the-rise-of-containers-and-microservices-50abff7c2cee>
- UniversityOfMinnesota. (2022, mei 17). *Python Virtual Environments in Linux*. <https://latisresearch.umn.edu/python-virtual-environments-in-linux>
- Vagrant. (2022a, mei 23). *Providers: VirtualBox*. <https://www.vagrantup.com/docs/providers/virtualbox>
- Vagrant. (2022b, mei 23). *Vagrant: Development Environments Made Easy*. <https://www.vagrantup.com/>
- Varga, G. (2022). *gusztavvargadr/ubuntu-desktop* (H. V. Cloud, Red.). <https://app.vagrantup.com/gusztavvargadr/boxes/ubuntu-desktop>
- Vembu. (2019, december 12). *Type-1 and Type-2 Hypervisors explained*. <https://www.vembu.com/blog/type-1-and-type-2-hypervisor/>
- Weaveworks. (2022). *Integrating Kubernetes via the Addon*. <https://www.weave.works/docs/net/latest/kubernetes/kube-addon/#-installation>
- Yfantis, V. (2020, april 30). *What Is a Hypervisor and What Are Its Benefits?* <https://www.parallels.com/blogs/ras/hypervisor/>
- Yury Izrailevsky, R. M., Stevan Vlaovic. (2016, februari 11). *Completing the Netflix Cloud Migration*. <https://about.netflix.com/en/news/completing-the-netflix-cloud-migration>