HO
GENT

Departement IT en Digitale Innovatie

Container orkestratie security: indentificeren van problemen en onderzoek naar de impact van security tools

Nick Heymans

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

Promotor:
Wim De Bruyn
Co-promotor:
Steven Trescinski

Instelling: —

Academiejaar: 2020-2021

Tweede examenperiode

Departement IT en Digitale Innovatie

Container orkestratie security: indentificeren van problemen en onderzoek naar de impact van security tools

Nick Heymans

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

Promotor:
Wim De Bruyn
Co-promotor:
Steven Trescinski

Instelling: —

Academiejaar: 2020-2021

Tweede examenperiode

Woord vooraf

Samenvatting

In deze bachelorproef worden de moderne veiligheidsrisicos geanalyseerd die gepaard gaan met container virtualisatie en container orkestratie. Hierbij zal specifiek gekeken worden naar de grootste veiligheidsrisicos, welke effecten deze kunnen hebben op een productieomgeving en hoe deze vermeden kunnen worden. Tevens zal onderzocht worden welke *security tools* (zoals *Project Calico* en *Kube-hunter*) er bestaan en hoe deze ingezet kunnen worden bij het beveiligen van een container cluster. De laatste jaren is het gebruik van containers in de *Information technology* (IT) infrastructuur fors gestegen en dit zal zo blijven evolueren in de komende jaren. Aan alle technologieën zijn nu eenmaal veiligheidsrisicos verbonden, container orkestratie vormt hier geen uitzondering op de regel. In voorgaand onderzoek werd er reeds gefocust op de grootste veiligheidsrisicos desondanks is er zeer weinig ingegaan op de effecten bij het toepassen van *best practices*. Met deze paper tracht ik het gebruik en de daaraan verbonden risicos van container virtualisatie en container orkestratie te onderzoeken. Daarnaast zal er ook gekeken worden naar hoe de verschillende *security tools* kunnen helpen bij het beveiligen van containers. Ten slotte wordt er onderzocht hoe deze risicos vermeden of opgelost kunnen worden en welk effecten ze hebben op de relevante criteria. Dit laatste zal via een *proof-of-concept* opstelling gebeuren.

Inhoudsopgave

Lijst van figuren

Lijst van tabellen

1. Inleiding

Tijdens de ontwikkeling van traditionele applicaties wordt de applicatie uitgewerkt in een specifieke testomgeving. Bij het overzetten van de applicatie van de test- naar de productieomgeving (i.e., van een Linux testomgeving naar een Windows productieomgeving), komen er vaak problemen naar boven). Deze problemen kunnen vermeden worden door gebruik te maken van containers en vergemakkelijken daarbij het uitrollen en schalen.

Een container is een pakket waar één enkele applicatie in zit, samen met alle nodige afhankelijkheden (Education, 2019). Dit zorgt ervoor dat deze gemakkelijk en snel van de ene omgeving naar de andere kan overgezet worden.

Naarmate het gebruik van containers steeg, steeg ook de nood om deze vanuit één centrale locatie te beheren. Om aan deze vraag te voldoen werden container orkestratie tools, zoals Kubernetes[1], ontwikkeld. Deze tools helpen bij het opzetten, uitbreiden en verbinden van een grote hoeveelheid containers.

In deze bachelorproef worden de moderne veiligheidsrisicos geanalyseerd die gepaard gaan met container virtualisatie en container orkestratie. Hierbij zal specifiek gekeken worden naar de grootste veiligheidsrisicos, welke effecten deze kunnen hebben op een productieomgeving en hoe deze vermeden kunnen worden.

Aan alle technologieën zijn nu eenmaal veiligheidsrisicos verbonden, container virtualisatie en container orkestratie vormen hier geen uitzondering op de regel. Een groot onderdeel van container security zijn de zogenaamde *security tools*(zoals *Project Calico* en *Kube-hunter*).

[1] https://kubernetes.io/

In voorgaand onderzoek werd er reeds gefocust op de grootste veiligheidsrisicos desondanks is er zeer weinig ingegaan op de effecten bij het toepassen van *best practices* en het gebruik van *security tools*. Met deze paper tracht ik het gebruik, en de daaraan verbonden risicos, van container virtualisatie en container orkestratie te onderzoeken. Daarnaast zal er ook gekeken worden naar hoe de verschillende *security tools* kunnen helpen bij het beveiligen van containers.

Ten slotte wordt er onderzocht hoe deze risicos vermeden of opgelost kunnen worden en welk effecten ze hebben op de relevante criteria. Dit laatste zal via een [proof-of-concept] opstelling gebeuren.

1.1 Probleemstelling

De probleemstelling houdt in dat veel bedrijven gebruik maken van containers en container orkestratie zonder hierbij al te veel aandacht te besteden aan de beveiliging hiervan. Daarnaast dient er gekeken te worden naar wat voor effecten de beveiliging van een container omgeving met zich mee brengt en hoe men best te werk gaat.

1.2 Onderzoeksvraag

Wat zijn de belangrijkste security risico's? Welke *security tools* zijn er en hoe werken ze? Welke impact hebben [best practices] en *security tools* op criteria?

1.3 Onderzoeksdoelstelling

Het doel van deze bachelorproef is hoofdzakkelijk om tot een verslag te komen met daarin aanbevelingen omtrent het beveiligen van een container cluster. Deze aanbevelingen zullen gestaafd worden door enkele scenarios en hun effect op enkele criteria.

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, 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

De stand van zaken of *State of the art* geeft een algemeen beeld weer van de technologieën die worden overwogen in dit onderzoek en tevens de verschillend manieren van toepassing.

2.1 Containers

In dit hoofdstuk zal uitgelegd worden wat containers zijn, hoe ze werken en waarvoor ze worden gebruikt.

Containers bieden veel voordelen vergeleken met normale virtuele machines (VM's). Ze kunnen snel opgezet worden en zijn gemakkelijk om te configureren terwijl virtuele machines vaak groot en traag zijn. Containers zijn pakketten waarin een applicatie vervat zit, samen met zijn benodigde bibliotheken en afhankelijkheden. Hierdoor kunnen ze vlot van de ene omgeving naar de andere worden overgezet zonder dat er extra configuratie nodig is (Education, 2019). Containers maken gebruik van besturingssysteem-virtualisatie om processen te isoleren van het *host* besturingssysteem. Daarnaast controleert het tevens de CPU gebruik en hoeveelheid RAM geheugen van deze processen (Docker, 2018).

Containers hebben eigenlijk geen eigen besturingssysteem nodig, alle containers delen 1 gezamenlijke *runtime engine*. Een *runtime engine* is de laag die verantwoordelijk is voor de communicatie tussen het besturingssysteem van de host machine en de containers zelf. De meeste gebruikte runtime engine is de *Docker Engine*[1].

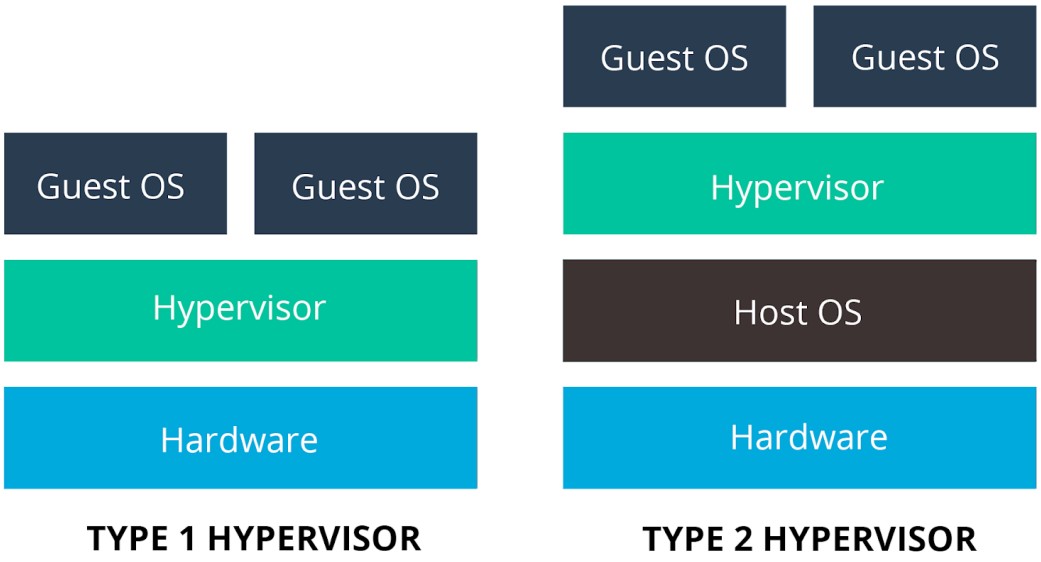
[1] docs.docker.com/engine/

2.1.1 Container vs. virtuele machine

Hypervisors

Bij traditionele virtuele machines virtualiseert een *hypervisor* de fysieke hardware. De hypervisor regelt het resource gebruik tussen de verschillende VM's en zorgt ervoor dat de hardware van de host (de fysieke hardware waar de hypervisor op geïnstalleerd is) eerlijk verdeelt wordt. Er zijn 2 types hypervisor (VMWare, 2021), namelijk:

- Type 1: Ook wel *bare metal* hypervisors genoemd. Deze werken rechtstreeks op de fysieke hardware van de host en hebben dus geen onderliggend besturingssysteem nodig. Door het rechtstreekse contact tussen de hypervisor en de hardware wordt een type 1 hypervisor beschouwd als de best presterende en meest efficiënte hypervisor. Een type 1 hypervisor wordt ook beschouwd als het veiligste van de 2, dit omdat de gebreken en kwetsbaarheden die doorgaans in besturingssystemen aanwezig zijn hier onmogelijk zijn.
- Type 2: Ook wel *hosted* hypervisors genoemd. Deze worden doorgaans geïnstalleerd op een bestaand besturingssysteem en steunt daar ook op voor het beheren van de resources. Een groot voordeel van type 2 hypervisors is dat ze een breed gamma aan hardware ondersteunen.

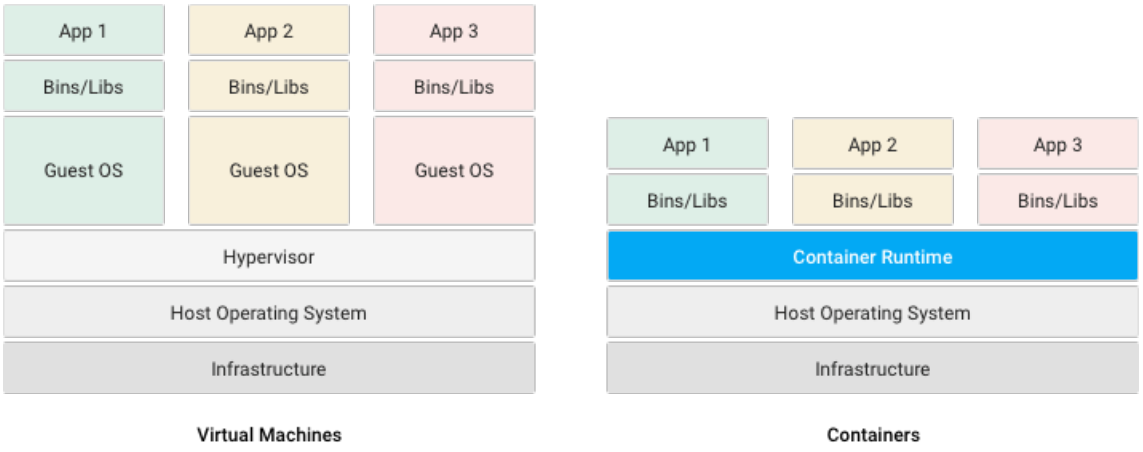


Figuur 2.1: Type 1 & Type 2 Hypervisor (VMWare, 2021)

Tegenwoordig worden type 1 hypervisors in productieomgevingen het meest gebruikt, dit vanwege hun efficiënte resource gebruik en veiligheid. Type 2 hypervisors worden meer gebruikt in testomgevingen omdat deze gemakkelijker op zetten zijn.

Elke Vm heeft zijn eigen volwaardg besturingssystemen, met gevolg dat deze veel resources gebruiken en hierbij vaak trager zijn. In plaats van de onderliggende hardware

te visualiseren gebruiken containers het besturingssysteem (meestal is dit Linux) zelf, zo bevat elke individuele container enkel de applicatie en de bijhorende bibliotheken en afhankelijkheden bevat (Education, 2020).



Figuur 2.2: Container vs. virtuele machine (Google, 2016)

2.1.2 Waarvoor worden containers gebruikt

Containers zijn zeer veelzijdig en kunnen dus in veel verschillende omstandigheden gebruikt worden. Enkele *use cases* waarvoor containers zeer geschikt zijn (Docker, 2021b):

- Microservices: Containers zijn klein en licht, waardoor ze goed passen bij microservice-architecturen waarin applicaties zijn opgebouwd uit vele, losjes gekoppelde en onafhankelijk services.
- Modernisering en migratie van applicaties: Een van de meest voorkomende benaderingen voor het moderniseren van applicaties begint met het containeriseren ervan, zodat ze naar de *cloud* kunnen worden gemigreerd.
- Nieuwe ontwikkelaars snel inwerken: Door gebruik te maken van containers verloopt het opzetten van een nieuwe lokale ontwikkelingsomgeving snel en vlot, hierdoor kunnen de ontwikkelaars direct aan de slag.

2.2 Docker

Docker[2] is een open source container platform dat sinds de release in 2013 ongelofelijk populair is geworden. In november 2019 stond de teller van aantal *pulls* op de *Docker hub* op 130 miljard, in juli 2020 stond deze al op 242 miljard. Dat is bijna dubbel op minder dan 8 maand tijd (Kreisa, 2020). Docker-containers kunnen overal draaien, in het datacenter, bij een externe serviceprovider of in de cloud. Docker containers kunnen zowel op Linux als op Windows draaien. Containers die op Windows gebaseerd zijn

[2] docker.com/

kunnen echter alleen op Windows systemen draaien, maar Linux containers kunnen op zowel Linux systemen en Windows systemen draaien (met behulp van een Linux VM). Dit komt omdat containers ontworpen zijn om het besturingssysteem van de host te gebruiken (Anil e.a., 2018).

2.2.1 Hoe werkt docker

Docker gebruikt een *Client-Server* architectuur. Deze werkt als volgt: De Docker *Client* communiceert met de Docker *Deamon* (een proces dat op de achtergrond draait en bepaalde (onderhouds-)taken uitvoert). Deze is verantwoordelijk voor het bouwen, runnen en verspreiden van containers (Docker, 2021a).

2.2.2 Docker componenten en terminologie

Docker Deamon

De Docker Daemon (`dockerd`) luistert naar Docker Application programming interface (API) verzoeken en beheert Docker objecten waaronder *images*, *containers*, netwerken, en *volumes*.

Docker Client

De Docker Client is de meest gebruikte manier voor gebruikers om te communiceren met Docker. De client stuurt alle ingevoerde commando's (zoals `docker pull` en `docker run`) door naar de Docker Daemon die deze uitvoert.

Docker registries

Een *Docker register* is een bibliotheek die *Docker images* opslaat. Het standaard register voor Docker is de *Docker Hub*[3]. Als de Docker daemon geen lokale Docker image vindt gaat deze standaard in de *Docker Hub* zoeken. Wanneer gebruik gemaakt wordt van de `docker pull` of `docker run` commando's wordt gebruikt, worden de benodigde images uit het register gehaald.

Docker objecten

Docker maakt gebruik van images, volumes en netwerken, al deze onderdelen worden objecten genoemt. Volgens Docker (2021a) zijn dit de belangrijkste objecten:

- **Images**: Docker images zijn *read-only* sjablonen met instructies om een Docker container op te zetten. Docker images kunnen uit de Docker hub gehaald worden en direct gebruikt worden zonder verdere configuratie. Verder kan je tevens bij-

[3] hub.docker.com/

komende instructies toevoegen aan de *base image* en deze opslaan als een nieuwe en aangepaste Docker image. Een Docker image is vaak gebaseerd op een andere images (i.e., Een nieuwe image kan gebaseerd zijn op een bestaande *Ubuntu* image maar installeert en configureert daarbij een *Apache* webserver). Alsook is het mogelijk om zelf een compleet nieuwe image te maken met behulp van een *dockerfile*.

- **Containers**: Een container is een uitvoerbare instantie van een image die wordt gecontroleerd via de Docker API. Een container kan verbonden worden met andere containers, aan externe opslag of gebruikt worden als basis voor nieuwe image.
- **Volumes**: De persistente gegevens die Docker containers kunnen gebruiken wordt opgeslagen in zogenaamde volumes. Deze volumes worden volledig gecontroleerd via de Docker API en bevinden zich buiten de container zelf. Hierdoor blijft het gewicht van de containers laag en kan de data blijven bestaan ook al wordt de container gestopt of verwijderd.

2.3 Container orkestratie

Container orkestratie helpt bij het opzetten, beheren, schalen en verbinden van een grote hoeveelheid containers. Container orkestratie helpt dus om complexe procedures te vergemakkelijken. Dit door veelvoorkomende processen en werkstromen te stroomlijnen en te optimaliseren. Een ander belangrijk onderdeel van orkestratie is het geautomatiseerd onderhoud van de applicaties die in de containers draaien (RedHat, 2021b).

Waarvoor word container orkestratie gebruikt?

Container orkestratie wordt vooral gebruikt voor het automatiseren en beheren van de configuratie en uitrol, het toewijzen van resources, de *Load balancing* en het monitoren van containers.

2.3.1 Container orkestratie tools

Om aan container orkestratie te gaan doen zijn er natuurlijk tools nodig die ons alle nodig functionaliteiten kunnen aanbieden. DevopsCube (2021) geeft een overzicht van de meest prominente orkestratie tools.

Kubernetes

Kubernetes(K8s)[4] is een open-source, container cluster manager en orchestratie tool. Het is gebouwd met een uitstekende resource manager voor het inzetten van containers op een efficiëntere manier. Kubernetes is voor vele organisaties de “de facto” container orkestratie tool geworden voor veel organisaties. Volgens CNCF (2021) zijn er meer dan 109 tools om containers te beheren, maar 89% is gebouwd met K8s aan de basis.

[4] kubernetes.io/

Openshift

Openshift behoort tot de 89% tools die gebouwd zijn bovenop Kubernetes. Het Openshift-project[5] word onderhouden door RedHat[6]. Het heeft zowel een open source versie (*openshift origin*[7]) als een enterprise versie (*openshift container platform*[8]).

Hasicorp Nomad

Nomad[9] is een orkestratieplatform van Hashicorp[10] dat containers op schaal kan ondersteunen. Op het vlak van applicatiemanagement is het zeer sterk vergelijkbaar met Kubernetes. Echter kan Nomad ook niet-containerapplicaties kan beheren, met gevolg dat deze zich kan distantiëren van de andere orkestratie tools. Daarnaast kan Nomad feilloos geïntegreerd worden met andere tools van Hashicorp.

2.4 Kubernetes

Hier wordt ingegaan op wat Kubernetes(K8s) en hoe het werkt. Daarnaast worden de verschillende technische termen die eigen zijn aan K8s uitgelegd.

Kubernetes is een open-source, container cluster manager en orchestratie tool. Het werd ontwikkeld door Google om hun container applicaties op grote schaal te kunnen orkestreren. Het project werd in 2014 *open-source* gemaakt, dat wil zeggen dat Google niet meer rechtstreeks de eigenaar is maar dat het project verder ontwikkeld wordt door vrijwilligers. Hierdoor konden ook andere organisaties niet alleen gebruik maken van deze krachtige tool, maar tevens meewerken aan de ontwikkeling ervan. De term *Kubernetes* is Grieks voor “stuurman van een schip” (Kubernetes, 2021). Enkele diensten die door K8s aangeboden worden:

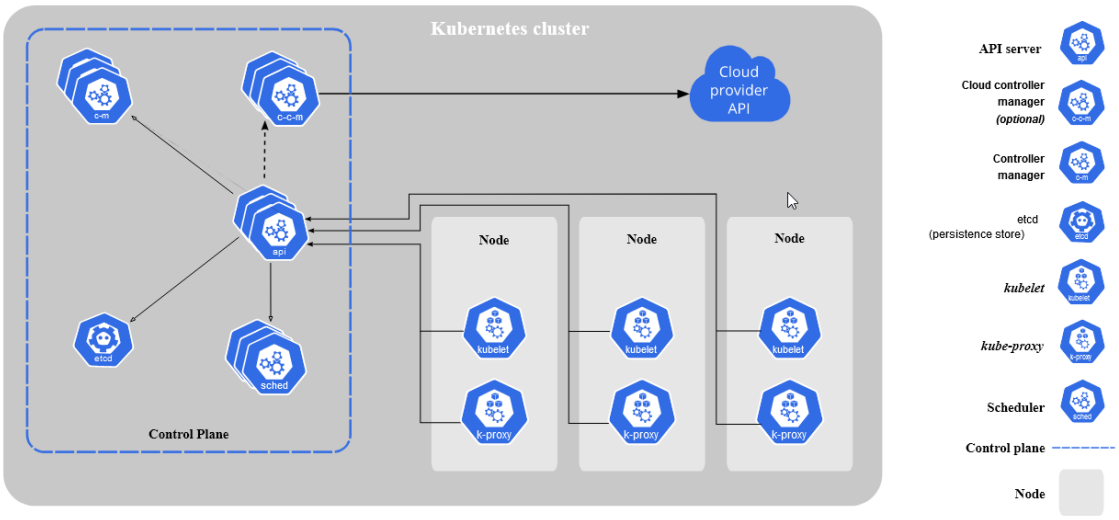
- Automatisch load-balancing op basis van de hoeveelheid verkeer.
- Opslag orkestratie: Het automatisch *mounten* van verschillende types opslag (lokale- of cloud opslag).
- Resource controle: K8s zorgt ervoor dat de beschikbare resources correct en efficiënt verdeeld worden.
- *Self-healing*: K8s kan containers heropstarten, vervangen of stopzetten als deze niet meer voldoende correct werken.

Mogelijks is volzinnen uitschrijven!!!!

[5] openshift.com/
[6] redhat.com/
[7] github.com/openshift/origin
[8] openshift.com/products/container-platform
[9] nomadproject.io/
[10] hashicorp.com/

Kubernetes componenten en terminologie

Zoals beschreven in de documentatie van Pedersen e.a. (2021) en RedHat (2021a) worden Kubernetes en de bijhorende componenten als volgt gedefinieerd: Kubernetes is een *cluster* bestaande uit 2 grote delen namelijk het *control plane* en de *nodes*. Deze nodes zijn de door K8s georkestreerde containers. Elke cluster bestaat uit uit minstens één, maar meestal meerdere, nodes. De nodes worden gebruikt om *pods* te hosten. Pods zijn de kleinst mogelijke eenheid in een K8s systeem. Een pod bestaat uit één of meerdere containers die opslag- en netwerk resources delen (RedHat, 2021a). In Figuur 2.3 worden de verschillende componenten van een K8s cluster gevisualiseerd.



Figuur 2.3: Kubernetes cluster componenten (Kubernetes, 2021)

Het hart van een K8s cluster is de ***Control plane***, hierin bevinden zich alle componenten die de cluster controleren.Tevens zitten alle gegevens omtrent de staat van de cluster en de configuratie hierin verbonden. Alle componenten die deel uitmaken van de *Control plane* kunnen op verschillende machines draaien. ierbij wordt aangeraden om wordt aangeraden om deze binnen eenzelfde machine te houden. De verschillende onderdelen en hun plaats binnen een cluster worden hieronder besproken.

De ***kube-apiserver*** wordt gezien als de *Front-end* van een Kubernetes cluster. Deze is de link tussen *nodes* en *pods* van de cluster en de *Control plane*. Het doel van deze apiserver is om de communicatie van de nodes en de Kubernetes API to bolwerken. De apiserver is ontworpen om horizontaal te schalen als deze te zwaar belast wordt. Hierbij creëert hij nieuwe instanties van zichzelf.

Alle data en informatie met betrekking tot de status van de cluser wordt opgeslagen in ***etcd***, een *key-value store database*.

De ***kube-scheduler*** is verantwoordelijk voor het toekennen van pods aan nodes en voor het verdelen van de resources tussen de verschillende Nodes. Het toekennen van een Node aan een Pod gebeurdt aan de hand van verschillende factoren. De mogelijke factoren

zijn onder andere de benodigde resources van een Pod en eventuele harware- en sofware beperkingen.

Een ***kube-controller-manager*** bestaat uit verschillende *controllers* die allemaal een verschillende taak op zich nemen. Enkele van deze [controllers] zijn:

- *Node controller*: Het hoofddoel van deze controller is om op te merken en reageren als er Pods zouden wegvallen.
- *Job controller*: Deze zoekt naar zogenaamde *job-objecten*, die eenmalige taken voorstellen, en creëert bijgevolg de nodige Pods om deze taken uit te voeren.
- *Service Account & Token controllers*: Deze creëert standaard accounts en *API access tokens* voor nieuwe Pods.

Als het *Control plane* het hart van een K8s cluster is dan kunnen we de *Nodes* het lichaam noemen. Deze doen namelijk al het zware werk en worden bestuurd door de *Control plane*.

Het eerste onderdeel van een Node is de ***kubelet***, een kleine applicatie die op elke Node aanwezig is. De *kubelet* houdt de gezondheid en status van de Pods, die binnen zijn Node draaien, in het oog. Wanneer de *Control plane* wil dat er iets gebeurd met de Node zorgt de *kubelet* ervoor dat deze acties correct uitgevoerdt worden.

De ***kube-proxy*** is een netwerk *proxy* die de verantwoordelijk is voor de K8s *networkservices*. De *kube-proxy* verzorgt netwerkcommunicatie zowel binnen- als buiten de cluster.

Het volgende component, namelijk de ***Container runtime***, werd al reeds besproken in sectie 2.1.

Ten slotte zijn er nog enkele extra *addons* die gebruikt kunnen worden om de functionaliteit van een K8s cluster uit te breiden. Een eerste voorbeeld van een *addon* is de mogelijkheid om persistent geheugen (ook wel *volumes* genoemd) toe te voegen zoals uitgelegd in sectie 2.2.2. Andere voorbeelden zijn het toevoegen van een ***cluster DNS***, ***Web UI*** of ***Dashboard*** en het opslaan van de log bestanden van de cluster met ***cluster-level logging***.

2.5 Security

Hier word het *security* aspect van containers en container orkestratie besproken. In dit hoofdstuk tracht ik antwoord te geven op de onderzoeksvragen “Wat zijn de belangrijkste security risico’s?” en “Welke *security tools* zijn er en hoe werken ze?”.

Uit een raport van (Tripwire, 2019) blijkt dat 94% van bevraagden bezorgd zijn over de veiligheid van hun containers. Uit hetzelfde rapport blijkt ook dat 47% weet dat ze mogelijks kwetsbare containers gebruiken in hun productieomgeving. Het beveiligen van een K8s cluster is dan ook een zeer grote en complexe taak. Dit vooral omdat er veel verschillende onderdelen zijn die allemaal andere veiligheidsproblemen met zich mee kunnen meebrengen. Door de hoeveelheid tijd en middelen die nodig zijn voor het implementeren van goede veiligheidsmaatregelen wordt het door veel bedrijven gezien als een *nice to have* in plaats van een *need to have*.

2.5.1 Meest voorkomende security problemen

Een van de meest voorkomende veiligheidsproblemen bij het opzetten van een K8s cluster is het fout definiëren van parameters. Dit kan er mogelijks voor zorgen dat een aanvaller kan ‘ontsnappen‘ uit een container (lees toegang krijgen tot het host systeem).

Een ander probleem zijn de onvermijdelijk *bugs* in K8s zelf. Dit is een van de gemakkelijkste veiligheidsproblemen om op te lossen, namelijk door altijd de laatste versie van Kubernetes te installeren. Als er een *bug* of kwetsbaarheid wordt ontdekt, is deze in de meeste gevallen binnen enkele dagen weggewerkt door het K8s security team.

2.5.2 Hoe een container cluster beveiligen

In dit hoofdstuk (***Alt. Hieronder***) zullen er enkele mogelijke manieren om een container cluser te beveiligen besproken worden.

Lewis (2019) en Rice (2019) geven enkele tips voor het veilig opzetten van een K8s cluster. Ten eerste wordt het gebruik van *Role Based Access Control*(RBAC) stevig aangemoedigd. Door gebruik te maken is het mogelijk om rollen toe te kennen aan bepaalde gebruikers. Deze rollen dicteren welke applicaties de gebruiker kan gebruiken wat die ermee kan doen.

Vervolgens wordt er aangeraden van enkel administrator (ook wel *root* genoemd) rechten te gebruiken als het echt niet anders kan. Dit kan ervoor zorgen dat als een aanvaller controle krijgt over de cluster, hij nog steeds gelimiteerd door de restricties die aan gewone accounts worden toegekend met RBAC. Ook de anonieme toegang tot de API kan men beter ontzeggen en vervangen door een gecontroleerde toegang met RBAC. Het gebruiken van een normaal gebruikers account kan ook afgedwongen worden door gebruik te maken van de ingebouwde parameter *RunAsUser*. ***hier komt een voorbeeld van RunAsUser***

De communicatie tussen Pods kan gecontroleerd worden aan de hand van zogenaamde ***Network policies***. Dit zijn regels die aanduiden welke pods met elkaar kunnen communiceren en welke soort data er kan uitgewisseld worden. De *Network policies* kunnen handmatig worden geconfigureerd of er kan ook gebruik gemaakt worden van een zogenaamde *Container Network Interface* (CNI). Een CNI helpt met het configureren en onderhouden van de *Network policies*. In sectie 2.6.1 wordt één van de meest prominente CNI's besproken, namelijk *Project Calico*.

hier komt een voorbeeld van een network policy

Fouten maken is menselijk, dus het is perfect mogelijk dat er tijdens het opstellen van de configuratie files enkele fouten insluipen. Hierdoor wordt het aangeraden om automatische *configutation checks* uit te voeren zodat de fouten vroegtijdig opgemerkt kunnen worden. Een tool die hiervoor vaak gebruikt wordt, namelijk ***kube-bench***, wordt besproken in sectie 2.6.2.

uitleg over Linux security features zoals SELinux/seccomp/apparmor

Het spreekwoord 'De ketting is maar zo sterk als zijn zwakste schakel' is ook hier van toepassing. De cluster kan ongelooflijk zwaar beveiligt zijn, als er gebruik wordt gemaakt van onveilige containers is al die moeite voor niets geweest.

Volgens Lewis (2019) zijn er verschillende manieren om ervoor te zorgen dat er enkel veilige containers gebruikt worden. De eerste is het gebruik maken van zogenaamde *trusted base images*. Dit zijn containers die volledig ontwikkeld zijn door het bedrijf dat ze in gebruik neemt. Het *Base* gedeelte van de naam komt van het feit dat deze containers meestal zeer simpel zijn. Zo kan elke container applicatie gebaseerd worden op dezelfde veilige container *image*. Een extra veiligheidsmaatregel die vaak aangeraden wordt is het gebruik van een *private registry*. Wat een *registry* precies is werd al reeds verduidelijkt in sectie 2.2.2. Een *private registry* is niets anders dan een *registry* die volledig door het bedrijf wordt gecontrolleerd en enkel door hen wordt gebruikt. Zij kiezen zelf welke *images* er wel of niet worden toegevoegd. Zo kan vermeden worden dat er per ongeluk een onveilige container wordt gebruikt binnen een cluster.

Spijtig genoeg is niet iedereen die met container clusters werkt even bezorgt over de veiligheid ervan. Om het gebruik van onveilige Pods te vermijden kan men een *policy agent* gebruiken. Hiermee kan er automatisch gecontroleerd worden of de Pods aan alle veiligheidsvoorschriften voldoen, alvorens ze in gebruik te nemen. Een voorbeeld van een *policy agent* is de ***Open Policy Agent***[11], onwikkeld door de *text Cloud Native Computing Foundation (CNCF)*[12].

[11] openpolicyagent.org/
[12] cncf.io/

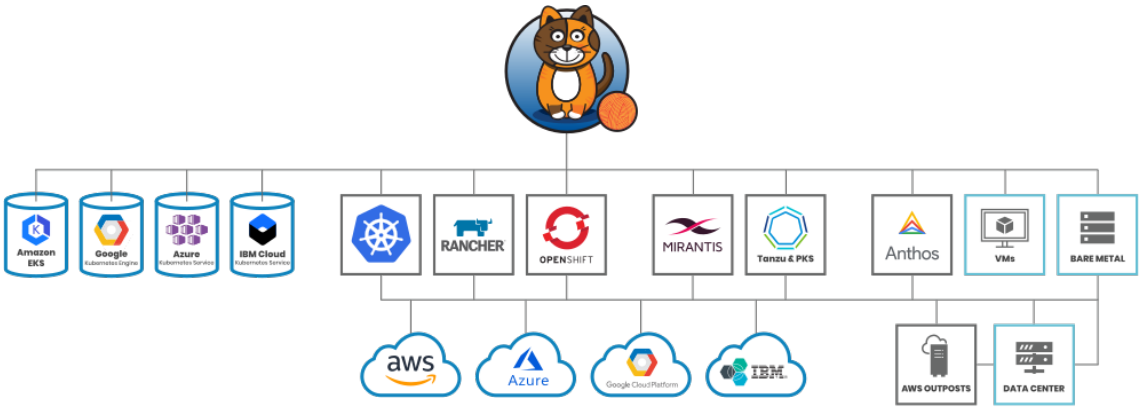
2.6 Security tools

In dit hoofdstuk zal ik trachten te antwoorden op de onderzoeksvraag “Welke security tools zijn er?”

Een andere manier om een K8s cluster verder te beveiligen is door gebruik te maken van *Security tools*. Taylor (2019) geeft een overzicht met de meest prominente tools.

2.6.1 Project Calico

Project Calico[13] is volledig *open source* en volgens Armstrong (2021) de meeste gebruikte security tool voor K8s. Calico is niet enkel een K8s security tool maar is ook volledig geïntegreerd met vele andere platformen. Figuur 2.4 geeft een overzicht van de door Calico ondersteunde platformen. Project Calico creert een soort van *microfirewall* rond elke service. De policies die ingesteld kunnen worden in Calico worden automatisch omgezet naar firewall regels. Deze worden vervolgens toegepast op elke service. Calico heeft zich onlangs aangesloten bij de Cloud Native Computing Foundation, waardoor het onder deskundig toezicht staat en nog beter geïntegreerd is in het Kubernetes ecosysteem.



Figuur 2.4: Platformen die ondersteunt zorden door Calico (Tigera, 2021)

2.6.2 Kube-Bench

Kube-Bench[14] is een *open-source* applicatie geschreven in *Go*[15] die controleert of k8s veilig is geïmplementeerd door controles uit te voeren op de cluster. De controles zijn gebaseerd op enkele guidelines die het *Center for Internet Security*[16] (CIS), een organisatie

[13] projectcalico.org/
[14] github.com/aquasecurity/kube-bench
[15] golang.org/
[16] cisecurity.org/benchmark/kubernetes/

die richtlijnen opmaakt voor het schrijven van veilige code, heeft opgesteld. De zogenaamde *CIS Kubernetes Benchmarks* worden geschreven door de Kubernetes *community* en worden door de CIS gecontrolleerd en gebundeld. *Kube-Bench* controleert niet alleen of er fouten in de beveiliging van een cluster zitten, het geeft ook mogelijke oplossingen. Enkele van de controles die gebeuren zijn het controleren van gebruikersautorisatie- en -authenticatie, de versleuteling van gegevens en kijken of de cluster het principe van *least privilege* volgt (een gebruiker krijgt enkel toegang tot gegevens die hij echt nodig heeft). De testen worden gedefinieert een in 'YAML Ain't Markup Language' (YAML) bestand zodat ze gemakkelijk aangepast en uitgebreid kunnen worden (Rice e.a., 2021). De test worden uitgevoerd op elke individuele node in de cluster waardoor het vooral geschikt is voor kleinere opstellingen.

2.6.3 Kube-hunter

Kube-hunter[17] is, net zoals *Kube-Bench*, een open-source applicatie gemaakt is door *Aqua Security*[18]. *Kube-hunter* breidt de functionaliteiten van *Kube-Bench* uit door penetratie testen uit te voeren op de cluster. Dit zorgt ervoor dat administrators problemen met een cluster gemakkelijk kunnen opsporen en verhelpen.

Er zijn drie verschillende manieren om *kube-hunter* te gebruiken. De eerste maakt gebruik van het ip-adres van de cluster om vanop afstand de testen uit te voeren. Bij de tweede manier wordt *kube-hunter* lokaal op één van de machines in de cluster geïnstalleerd om die netwerkinterfaces van die specifieke machine te testen. De derde en laatste manier om *kube-hunter* te gebruiken is om het te installeren in een Pod binnen de cluster. Deze manier kan aantonen wat voor schade er kan aangericht worden als één van de Pods gecompromitteerd zou zijn.

[17] github.com/aquasecurity/kube-hunter
[18] aquasec.com/

3. Methodologie

4. Conclusie

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 Inleiding en State-of-the-art

A.1.1 Wat zijn containers?

Het uitrollen en schalen van applicaties wordt steeds vaker gedaan met behulp van containers. Tijdens de ontwikkeling van traditionele applicaties wordt de applicatie ontwikkeld in een specifiek testomgeving. Vervolgens wordt de applicatie overgezet naar de productieomgeving wat vaak voor problemen zorgt (bijvoorbeeld van een linux testomgeving naar een Windows productieomgeving). Een container is een pakket waar één enkele applicatie in zit, samen met alle nodige afhankelijkheden(Education, 2019). Dit zorgt ervoor dat deze gemakkelijk en snel van de ene omgeving naar de andere kan overgezet worden. De containers maken gebruik van een 'runtime engine', dit is een laag die verantwoordelijk is voor de communicatie tussen het operating system van de host machine en de containers zelf. De meeste gebruikte 'runtime engine' is de 'Docker Engine'[1]. Deze is al sinds 2013 de industriestandaard als het gaat over container software(McCarty, 2018). Naarmate het gebruik van containers steeg, steeg ook de nood naar op manier om deze vanuit één centrale locatie te beheren. Om aan deze vraag te voldoen werden container orkestratie tools, zoals Kubernetes [2], ontwikkeld. Deze tools helpen bij het opzetten, uitbreiden en verbinden van een grote hoeveelheid containers.

[1] https://docs.docker.com/engine/
[2] https://kubernetes.io/

A.1.2 Waarom container applicaties?

Container applicaties hebben enkele voordelen tegenover normale applicaties, ze draaien namelijk geïsoleerd van de rest van het systeem. Ze kunnen dus perfect werken zonder afhankelijk te zijn van andere containers. Dit garandeerd dat als er één container aangetast is, de rest zonder interruptie kan verderwerken. De containers delen wel verschillende resources van het host systeem, wat de deur opent voor veiligheidsinbreuken tussen containers.

A.1.3 Context voor dit onderzoek

Gartner (Pettey, 2019) voorspeld dat tegen 2022 maar liefst 75% van alle internationale organisaties gecontaineriseerde applicaties zullen gebruiken in hun productieomgeving. Dit zowel in lokale datacenters alsook in online cloud omgevingen. Uit een raport van Tripwire (2019) blijkt dat 94% van bevraagden bezorgd zijn over de veiligheid van hun containers. Uit hetzelfde raport blijkt ook dat 47% weet dat ze kwetsbare containers gebruiken in hun productieomgeving. Spijtig genoeg werd bij voorgaande onderzoeken, zoals StackRox (2020), het effect van 'security best practices en tools' op opzetsnelheid, benodigde resources en stabiliteit steeds onderbelicht.

A.1.4 Verloop van het onderzoek

In deze paper zal ik onderzoeken wat de belangrijkste bronnen van veiligheidsinbreuken zijn en hoe deze vermeden kunnen worden. Tegelijkertijd krijg ik via dit onderzoek de opportuniteit om er achter te komen of er vooral technische problemen of menselijke fouten aan de basis liggen van de veiligheidsrisico's. In de volgende paragraaf staat er beschreven hoe ik te werk zal gaan.

A.2 Methodologie

Voor dit onderzoek zullen er drie scenario's opgezet worden. Elk scenario zal verschillende keren worden uitgevoerd en voor elk criteria zal het genomen worden (eventueel reking houdende met uitschieters). Bij elke scenario zullen er verschillende 'security best practices en tools' gebruikt worden. Deze zullen getest worden op basis van de volgende criteria:

- Deployment snelheid
- Benodigde resources
- Stabiliteit

Voorbeelden van scenario's:

- S(0): Er wordt een container applicatie opgezet in een Kubernetes cluster zonder extra security configuratie.
- S(1): Er wordt een container applicatie opgezet in een Kubernetes cluster en enkele 'best practices' worden toegepast.
- S(2): Er wordt een container applicatie opgezet in een Kubernetes cluster waar er gebruik word gemaakt van enkele 'security tools' zoals 'Project Calico' en 'Kube-hunter'.

Door gebruik te maken van deze scenario's en criteria hopen we vast te stellen dat het toepassen van 'security best practices en tools' een positieve invloed heeft op het gebruik van containers en orkestratie tools.

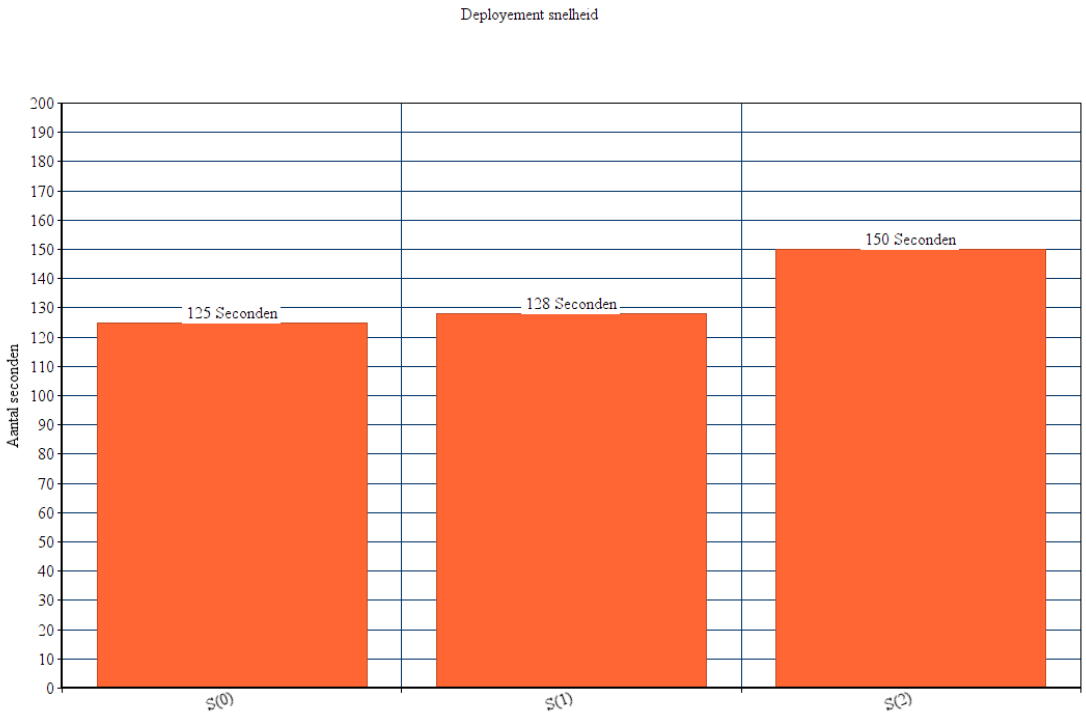
A.3 Verwachte resultaten

Op basis van de criteria wordt er verwacht dat Scenario 0 en 1 even snel op opgezet kunnen worden en evenveel resources gebruiken. Ze zullen beide kwetsbaarder zijn aangezien de 'best practices' vooral bestaan uit het correct gebruik van wachtwoorden en gebruiker privileges. Scenario 2 daarintegen zal iets meer tijd nodig hebben om opgezet te worden(zie Figuur1) en zal daarbij meer resources gebruiken(zie Figuur2). Dit zou te wijten zijn aan de gebruikte 'security tools' die extra tijd en resource nodig hebben.

A.4 Verwachte conclusies

Uit dit onderzoek willen we concluderen dat het toepassen van 'best practices' en het correct gebruik van security tools een positief effect teweeg brengt bij het gebruik van container orkestratie tools. We trachten daarnaast ook aan te duiden dat het omzeilen van security risico's een belangrijk aspect is bij het ontwikkelen van container applicaties. Tot slot kan er geconcludeerd worden dat het beveiligen van container clusters steeds belangrijker wordt. Daarnaast is het tevens van belang dat de persoon die een cluster opzet daarbij de onderliggende werkwijze goed kent en zich bewust is van de mogelijke valkuilen.

A.5 Bijlagen



Figuur A.1: Verwachte opstart tijd

Benodigde resources	CPU %	Memory %
S(0)	45%	30%
S(1)	46%	32%
S(2)	55%	45%

Figuur A.2: Verwacht resource gebruik

Bibliografie

Anil, N., Coulter, D., Victor, Y., Parente, J., Warren, G. & Wenzel, M. (2018, augustus 31). *What is Docker?* Microsoft. Verkregen 9 maart 2021, van https://docs.microsoft.com/en-us/dotnet/architecture/microservices/container-docker-introduction/docker-defined

Armstrong, J. (2021, januari 6). *A 2020 Review of the Worlds Most Popular Kubernetes CNI*. https://www.projectcalico.org/a-2020-review-of-the-worlds-most-popular-kubernetes-cni/

CNCF. (2021, maart 16). *CNCF Cloud Native Interactive Landscape*. https://landscape.cncf.io/

DevopsCube. (2021, maart 1). *List of Best Docker Container Orchestration Tools/Services*. https://devopscube.com/docker-container-clustering-tools/

Docker. (2021a, maart 5). *Docker overview*. Docker. Verkregen 9 maart 2021, van https://docs.docker.com/get-started/overview/

Docker. (2021b, maart 3). *How Docker HelpsDevelopment Teams*. Docker. https://www.docker.com/use-cases

Education, I. C. (2019, mei 25). *Containerization*. IBM. https://www.ibm.com/cloud/learn/containerization#toc-what-is-co-r25Smlqq

Education, I. C. (2020, september 2). *Containers vs. VMs: Whats the Difference?* IBM Cloud Education. Verkregen 8 maart 2021, van https://www.ibm.com/cloud/blog/containers-vs-vms

Google. (2016). *Containers at Google: A better way to develop and deploy applications*. Google. https://cloud.google.com/containers

Kreisa, J. (2020, juli 30). *Docker Index: Dramatic Growth in Docker Usage Affirms the Continued Rising Power of Developers*. Docker. Verkregen 9 maart 2021, van https://www.docker.com/blog/docker-index-dramatic-growth-in-docker-usage-affirms-the-continued-rising-power-of-developers/

Kubernetes. (2021, februari 1). *What is Kubernetes?* Kubernetes. https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

Lewis, I. Kubernetes Security Best Practices. Engles. In: KubeCon + CloudNativeCon (Kubernetes Forum Seoul, 10 december 2019). Seoul: Cloud Native Computing Foundation, 2019, december 10. Verkregen 29 maart 2021, van https://www.youtube.com/watch?v=wqsUfvRyYpw

McCarty, S. (2018, februari 22). *A Practical Introduction to Container Terminology*. https://developers.redhat.com/blog/2018/02/22/container-terminology-practical-introduction/#h.6yt1ex5wfo3l

Pedersen, B. E., Tokuda, T., Nakamura, H., Yi, J. & Wang, S. (2021, januari 3). *Kubernetes Components*. https://kubernetes.io/docs/concepts/overview/components/

Pettey, C. (2019, april 23). *6 Best Practices for Creating a Container Platform Strategy*. https://www.gartner.com/smarterwithgartner/6-best-practices-for-creating-a-container-platform-strategy/

RedHat. (2021a, maart 24). *Introduction to Kubernetes architecture*. RedHat. Verkregen 24 maart 2021, van https://www.redhat.com/en/topics/containers/kubernetes-architecture

RedHat. (2021b, maart 16). *What is container orchestration?* https://www.redhat.com/en/topics/containers/what-is-container-orchestration

Rice, L. The State of Kubernetes Security. Engels. In: Paris Container Day. Paris Container Day. Parijs, 2019, juni 12. Verkregen 29 maart 2021, van https://www.youtube.com/watch?v=_l56oUxHSio

Rice, L., Rojas, R., Huang, H. & Rotem, Y. (2021, februari 23). *kube-bench*. Aqua Security. https://github.com/aquasecurity/kube-bench

StackRox. (2020, februari 19). *State of Container and Kubernetes Security* (onderzoeksrap.). StackRox. https://www.stackrox.com/kubernetes-adoption-security-and-market-share-for-containers/

Taylor, T. (2019, maart 25). *Top 7 Kubernetes security tools to harden your container stack*. Verkregen 29 maart 2021, van https://techgenix.com/kubernetes-security-tools/

Tigera. (2021, maart 28). What is Calico? https://www.projectcalico.org/

Tripwire. (2019, januari 1). *Tripwire State of Container Security Report* (onderzoeksrap.). Tripwire. Verkregen 30 november 2020, van https://3b6xlt3iddqmuq5vy2w0s5d3-wpengine.netdna-ssl.com/state-of-security/wp-content/uploads/sites/3/Tripwire-Dimensional-Research-State-of-Container-Security-Report.pdf

VMWare. (2021, februari 8). *Hypervisor*. https://www.vmware.com/topics/glossary/content/hypervisor