HO GENT

Faculteit Bedrijf en Organisatie

Ansible role testing: analyseren en verbeteren van de workflow

Robin Ophalvens

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

Promotor:
Irina Malfait
Co-promotor:
Bert Van Vreckem

Instelling: HoGent

Academiejaar: 2019-2020

Tweede examenperiode

Faculteit Bedrijf en Organisatie

Ansible role testing: analyseren en verbeteren van de workflow

Robin Ophalvens

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

Promotor:
Irina Malfait
Co-promotor:
Bert Van Vreckem

Instelling: HoGent

Academiejaar: 2019-2020

Tweede examenperiode

Woord vooraf

Met behulp van deze bachelorproef probeerde ik aan te tonen dat een bepaalde werkwijze met behulp van een software een volwaardig alternatief kan zijn voor een reeks van testing best practices die vandaag de dag als standaard worden beschouwd binnen de DevOps wereld. De grootste voordelen daarbij zijn dat het technisch veel minder complex is waardoor men zich meer kan focussen op de ontwikkeling van de software zelf, in plaats van de testing fase.

Daarnaast is deze bachelorproef een laatste stap naar het bekomen van de bachelor in de Toegepaste Informatica met Systeem- en Netwerkbeheer als specialisatie. Tijdens mijn opleiding ben ik aanraking gekomen met het GNU/Linux besturingssysteem alsook de principes van de DevOps wereld, wat mijn interesse naar deze onderwerpen heeft aangewakkerd en waaruit een passie gegroeid is.

Tijdens dit onderzoek ben ik door enkele individuen geholpen die deze grote onderneming samen met mij tot een goed einde hebben gebracht. Zonder hun zou deze paper niet bestaan dus om die reden zet ik deze persoon graag even in de spotlight.

Als eerste wens ik mijn co-promotor, Bert Van Vreckem te bedanken voor zijn technische input, zijn suggesties voor deze bachelorproef en vooral het idee dat naar dit onderzoek heeft geleid. Ik hoop oprecht dat deze bachelorproef u kan bijstaan bij het testen van uw Ansible rollen.

Ook wens ik mijn promotor Irina Malfait te bedanken voor de grote onderneming die zij nam om deze uitgebreide scriptie door te nemen en feedback te geven naar inhoud en taal toe. Ten alle tijde kon ik bij haar terecht bij vragen over zowel de bachelorproef als de stage.

En tenslotte wens ik ook een dankbaarheid te betuigen aan mijn medestudent en beste vriendin Jolien Vervinckt. Ook zij heeft de immense uitdaging op zich genomen om deze bachelorproef door te nemen, fouten eruit halen, en aanpassingen te suggereren waar nodig en dankzij haar hebben wij samen de finishlijn van deze opleiding bereikt.

Ik wens u veel leerplezier toe en ik hoop dat ook u gebruik kan maken van deze bachelorproef naar de toekomst toe!

Samenvatting

In dit onderzoek werd er gezocht hoe men de workflow tussen software containers, configuratie management tools en testing tools kan verbeteren. Om specifieker te zijn werd er bekeken of er een vlottere workflow bestaat om Ansible rollen op een snelle en consistente manier te testen waarbij de ontwikkelaar minimale inspanning aan voorbereiding dient te doen. Het doel was om een methodologie te vinden die men zowel op een lokale machine kan uitvoeren alsook op *Continous Integration/ Development* platformen, zij het met minimale aanpassingen. Ansible rollen worden namelijk tegenwoordig hoofdzakelijk via Virtuele Machines (afgekort VM's), Docker omgevingen, cloud omgevingen of op*bare-metal* machines getest, elk met hun voor- en nadelen. VM's zijn makelijk op te stellen maar vragen tijd en computer resources. Anderzijds zijn er de welbekende Docker containers die veel lichter zijn dan VM's maar wel complexer om in te stellen.
Bij dit onderzoek werd er bekeken of Molecule, een module onder Ansible, een volwaardige oplossing is voor deze probleemstelling en werd alsook getest of men Podman zou kunnen gebruiken in plaats van Docker om containeropgevingen op te zetten op lokale machines. Tenslotte worden 'approaches' besproken hoe men Molecule zowel lokaal als op CI omgevingen kan gebruiken.

Inhoudsopgave

Lijst van figuren

Lijst van tabellen

1. Inleiding

Ansible[1] is een open-source *software provisioning*, *configuratie management* en *application-deployment* tool die in 2012 werd gelanceerd door Michael DeHaan. Dankzij dit programma is het voor een systeembeheerder zeer eenvoudig om zowat elke taak te automatiseren die kan variëren van het configureren van systeem specifieke instellingen, gebruikers en groepen aanmaken tot software installeren en deze configureren.
Via Ansible Galaxy, een community platform, kan men deze geautomatiseerde taken, ook wel gekend als 'roles', met elkaar delen zodat een systeembeheerder snel een bepaalde taak kan uitvoeren zonder dat hij of zij de code daarvoor van de grond af hoeft te schrijven.

Een zogenaamde role programmeren is een zeer uitgebreid proces dat afhankelijk is van de complexiteit van de taak die dient uitgevoerd te worden. Een ontwikkelaar wilt dan uiteraard zijn role kunnen testen om eventuele anomalieën te ontdekken en te elimineren, of eenvoudigweg om enkele optimalisaties te testen.

Er bestaan tegenwoordig verschillende manieren om deze rollen te testen waarbij de meest toegepaste methode het gebruik van een Virtuele Machine is. De ontwikkelaar start een VM op waarna Ansible wordt opgeroepen om de gevraagde taken uit te voeren. Een andere methode bestaat eruit om een container op te zetten via bijvoorbeeld Docker of in sommige gevallen voert men de code rechtstreeks uit op een oude machine die niet meer in productie wordt gebruikt.

Deze testmethodes hebben elk hun voor- en nadelen, men zal deze moeten in acht nemen wanneer een keuze gemaakt wordt. Een Virtuele Machine is namelijk zeer gemakkelijk om op te stellen en zal meestal voor minder compatibiliteitsproblemen zorgen dan een container. Het nadeel is echter dat het in vergelijking met een container redelijk hardware

[1] https://www.ansible.com/

intensief is. Een container is dan weer het omgekeerde. Daar heeft men wat meer kennis nodig om iets dergelijks op te zetten met als groot voordeel dat het zeer licht is. Er is dus een zeker dilemma tussen gebruiksvriendelijkheid en *resource* intensiviteit. Daarom zou het handig zijn mocht er een soort van gulden middenweg bestaan.

Maak kennis met Molecule. Wat begon als een éénmansproject door retr0h[2] is ondertussen mee opgenomen in het Ansible project dat wordt onderhouden door Red Hat, een multinational Amerikaans softwarebedrijf. Molecule heeft als doel om ontwikkelaars te assisteren bij het ontwikkelen en testen van Ansible roles door op een snelle en gebruiksvriendelijke manier containers op te zetten en te beheren. Hierop is het mogelijk om Ansible code op uit te voeren. Bij dit onderzoek wordt er kritisch nagegaan of dit effectief het geval is en wordt als bijdrage naar de DevOps wereld toe gekeken of Podman kan gebruikt worden om containers op te zetten. Daarnaast zal er onderzocht worden wat de impact van deze manier van werken kan zijn op *Continious Development/Integration* platformen zoals Travis CI, Jenkins en zo voort.

1.1 Probleemstelling

Het concrete probleem bestaat er dus uit dat de meeste testmethodes veel voorbereiding eisen en dat dit over het algemeen een tijdrovend proces is. Anderzijds dient er ook een manier gezocht te worden waarbij testing zowel op lokale machines als op cloud omgevingen en CI platformen kan uitgevoerd worden, bij voorkeur met zo weinig mogelijk extra configuratie. Het zou ook een meerwaarde bieden om te achterhalen waarom men bij Molecule de voorkeur geeft aan Docker en niet aan Podman.

1.2 Onderzoeksvraag

Is Molecule een volwaardige testing oplossing voor Ansible rollen? Is Molecule geschikt zowel lokaal als op CI platformen? Welke *approaches* kunnen we gebruiken voor CI/CD platformen? Is podman een effectieve vervanger voor Docker?

1.3 Onderzoeksdoelstelling

Het doel van deze bachelorproef is om hoofdzakelijk een proof-of-concept op te zetten voor de workflow die zal voorgesteld worden. Idealiter is er ook een consistente en gebruiksvriendelijke aanpak rond hoe men een gelijkaardig resultaat tussen lokale testen en CI platform testen kan verzekeren. Dit zou dan eerder om een aanbeveling gaan. Tenslotte wordt er een vergelijkende studie gemaakt tussen het gebruik van Podman en Docker bij Molecule en welke driver men zou aanraden.

[2] https://github.com/retr0h

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. Deze methodieken worden in een 'handleiding' manier besproken om te bewijzen dat deze herproduceerbaar zijn.

Een concrete toepassing wordt alsook besproken in Hoofdstuk 3.6 samen met een tweede, iets beperktere toepassing in Hoofdstuk 3.7

In Hoofdstuk 4 worden de resultaten van de besproken methodologieën verder toegelicht en wordt er geprobeerd om te achterhalen wat de effectieve voor- en nadelen zijn van de aanpakken. Ook wordt er een advies gegeven naar de toekomst toe.

In Hoofdstuk 5, tenslotte, wordt de conclusie gegeven en een antwoord geformuleerd op de onderzoeksvragen. Daarbij wordt ook een aanzet gegeven naar toekomstig onderzoek binnen dit domein.

2. Stand van zaken

De stand van zaken of *State of the art* geeft een beeld van de technologieën die worden overwogen voor dit onderzoek en op welke manieren ze kunnen toegepast worden om een antwoord te vinden op de onderzoeksvragen.

2.0.1 Ansible

Sinds 2012 is Ansible uitgegroeid tot een stabiele en veelgebruikte (Holst, 2020) *Configuration Management Tool* dat alsook een goede optie is binnen een productieomgeving (Gardner, 2019). Dit is vooral op te merken door verscheidene handleidingen, continue updates vanuit Red Hat en een actieve community die op verschillende platformen te vinden is. Als men echter informatie wil opzoeken rond deze onderwerpen zal men snel ondervinden dat het beschikbare materiaal vooral bestaat uit blogposts, online artikels, videos, ebooks en boeken. Een snelle zoekopdracht met Ansible als parameter via een tool zoals Google Scholar bewijst dit.

Werking van Ansible

De kracht van een *Configuration Mangement Tool* zoals Ansible is dat het zowel lokaal als op *remote* machines kan worden uitgevoerd. Ansible verbindt zich tijdelijk met andere computers via SSH (Secure Shell), of via Windows Remote Management voor Microsoft Windows computers, en voert daarna de gevraagde commando's uit.

Hoe vertelt men wat men Ansible moet doen? Er bestaan verschillende werkwijzes die men kan toepassen om Ansible code op te stellen en te laten uitvoeren.

Als eerste is het perfect mogelijk om via een terminal te werken. Hoewel deze manier van werken *straight-to-the-point* is kan het snel zeer onoverzichtelijk worden als men veel code moet behandelen. De meest gekende manier is om te werken met zogenaamde playbooks. Een playbook is een YAML bestand waar men alle gewenste configuratie in kan schrijven en daarna via de terminal kan oproepen. Het is al een stuk overzichtelijker dan exclusief met de terminal werken. Onderstaande code geeft een voorbeeld van een playbook:

```
- hosts: webservers
  remote_user: root

  tasks:
    - name: ensure apache is at the latest version
      yum:
        name: httpd
        state: latest
    - name: write the apache config file
      template:
        src: /srv/httpd.j2
        dest: /etc/httpd.conf
```

Dit is een simpele playbook die voor alle webserver computers er voor zal zorgen dat Apache, een software om een webserver op te zetten, op het systeem staat. Hierbovenop zal het een configuratiebestand plaatsen in de juiste map. Op dit moment is deze playbook nog leesbaar en is het eenvoudig om te begrijpen wat er gaande is. Echter, wanneer men een grootschalig automatisatieproject moet uitvoeren zal ook dit bestand zeer groot worden wat opnieuw een negatieve impact heeft op overzichtelijkheid en leesbaarheid.

Omwille van deze reden raadt de Ansible documentatie aan om met een zogenaamde 'role' te werken. Wat er dan in feite gebeurt is dat men in plaats van met één YAML file te werken eerder de code gaat verspreiden over meerdere bestanden, afhankelijk van hun functie. Zo kan er een 'role' aangemaakt worden die zich exclusief bezig zal houden met de Apache software, een andere role kan zich dan ontfermen over een andere software.

De structuur van zo een role met zijn onderdelen ziet er als volgt uit:

- defaults
- files
- handlers
- meta
- tasks
- templates
- vars

De functie van elk onderdeel wordt hier even kort aangehaald.

Tasks

Ongetwijfeld het belangrijkste deel van de playbook. Een task, zoals zijn naam al doet vermoeden, beschrijft wat Ansible exact moet uitvoeren op een computer. Een voorbeeld van zo een task voor een Centos 7 Linux systeem is het volgende:

```
- name: "Ensure required packages are present"
  yum:
  name: "httpd"
  state: present
```

Voor systeembeheerders, of iedereen in feite, die niet vertrouwd zijn met Ansible kan dit formaat nogal verwarrend lijken. Deze task heeft namelijk als doel om te controleren of HTTPD, ook wel bekend als de Apache Webserver, geïnstalleerd staat op het systeem en alsook om het te downloaden wanneer dit niet het geval is. Waarom is dit niet gewoon een standaard commando om een package te installeren zoals ‘yum install httpd‘ ?

Mocht men dit zo implementeren dan zou Ansible het Centos7 systeem forceren om altijd de HTTPD package te downloaden en te installeren, zelfs al is het reeds aanwezig op het systeem. Voor één package en één systeem zal dit uiteraard niet zo veel uitmaken, maar als de scope wordt vergroot naar bijvoorbeeld 10 packages die moeten worden geïnstalleerd op 10 systemen, dan zal het een hele tijd duren om deze taak uit te voeren.

Met dit formaat, de Ansible code dus, wordt het probleem opgelost. Ansible zal eerst kijken of de gevraagde software geïnstalleerd is op het systeem. Indien dit het geval is, dan gaat Ansible verder naar de volgende task.

In een scenario waar er echter meerdere packages geïnstalleerd moeten worden zou het logisch lijken om meerdere ‘nodes‘ te maken onder ‘yum‘, maar dit is zeer inefficiënt als men met veel packages moet werken. In plaats daarvan kan er gekozen worden om gebruik te maken van variabelen. De conventie van Ansible zegt dat variabelen best worden ingesteld onder de vars map.

Vars

Via variabelen kan men repetitieve code vermijden, maar het biedt ook de kans om specifieke instellingen te configureren afhankelijk van het type systeem.

Als eerste wordt er gekeken om variabelen te gebruiken die het gemakkelijker moeten maken om Tasks op te stellen.

Onder de vars folder zal Ansible zoeken naar bestanden waar variabelen ingesteld zijn en zal deze proberen toe te passen op de Tasks. Een variabele maken is zeer eenvoudig zoals onderstaande code illustreert:

```
package_list:
  - httpd
  - firewalld
service_list:
  - httpd
  - firewalld
```

Bij deze code wordt er een package lijst en een service lijst aangemaakt. In de package_list lijsten we de softwarepakketten op die Ansible dient te installeren en in de service_list worden de programma's ingevuld die het systeem moet activeren. Variabelen worden op volgende manier aangeroepen bij een Task:

```
- name: "Ensure required packages are present"
  yum:
    name: "{{ package_list }}"
    state: present

- name: "Ensure httpd service is started and enabled"
  service:
    name: "{{ item }}"
    state: started
    enabled: true
  with_items: "{{ service_list }}"
```

Hier worden twee manieren getoond hoe variabelen kunnen worden aangeroepen. In tegenstelling tot andere programmeertalen is een 'import' van het vars bestand niet nodig. Ansible zal zelf naar deze zoeken onder de Vars folder. In beide gevallen roept men de variabel zelf aan door dubbele accolades te gebruiken met twee spaties voor en na de naam van de variabele die opgeroepen moet worden. Voor de eerste task valt op dat onder yum de variabele rechtstreeks als een lijst kan meegegeven worden terwijl het bij de tweede task eerder lijkt op een soort van For-loop. Om te weten welke van de twee van toepassing is voor een bepaalde task gaat men dit best even opzoeken via de Ansible documentatie. Zelf experimenteren en zien wat het resultaat is blijft uiteraard ook een oplossing.

Defaults

Onder Vars stelt met gewoonlijk variabelen in die op elk systeem uitgevoerd moeten worden, maar wat als het nodig is om per computer een bepaalde variabele te kunnen instellen, zoals een naam bijvoorbeeld? Gebruikelijk doet men dit door onder een map genaamd host_vars per computer de gewenste variabelen in te stellen. Met Defaults kan men, zoals de naam al doet voorstellen, standaardwaarden instellen voor deze specifieke variabelen.

Files

Files is een folder waar men bestanden kan plaatsen die moeten worden gekopieërd naar een bepaalde bestemming. Hoe dit exact moet gebeuren wordt gespecifieerd in een task.

Handlers

Handlers zijn een soort van tasks die worden geactiveerd op basis van een bepaalde event. Een voorbeeldsituatie is de volgende: Een task installeert een Apache webserver voor een Linux systeem dat wordt gevolgd door een andere task die een configuratiebestand van Apache aanpast. Om deze aanpassing echter toe te passen op het systeem moet de webserver opnieuw worden gestart. In dit geval is een handler een zeer interessante keuze. Men kan namelijk een handler koppelen aan de task die het configuratiebestand aanpast en zo instellen dat Ansible de webserver zal herstarten wanneer een bestand wordt gewijzigd.

```
handlers:
  - name: restart apache
    service:
      name: apache
      state: restarted
```

Templates

Een Template in Ansible is een bestand dat typisch functioneert als een configuratiebestand voor een bepaalde software op een specifiek systeem. De kracht van een template bestaat eruit dat het mogelijk is om in de template variabelen in te stellen. Per computer die dit configuratiebestand moet krijgen, kunnen de juiste variabelen hiervoor ingevuld worden. Deze variabelen kan men opnieuw meegeven bijvoorbeeld via host_vars.

Onderstaande playbook geeft een voorbeeld waar een webpagina wordt geplaatst in de map /var/www/html/ met als bestandsnaam index.html.

```
- hosts: all
  become: yes
  tasks:
    - name: Install index.html
      template:
        src: index.html.j2
        dest: /var/www/html/index.html
```

Onder 'src' of voluit 'source' is te zien dat index.html eindigt met een j2 extentie. j2 of Jinja2[1]is programma dat specifiek gemaakt is om met templates te werken. Het laat toe om onder andere gebruik te maken van variabelen zoals er hier eerder naar verwezen werd.

Een voorbeeld van hoe een Jinja2 template er uit kan zien voor een index.html pagina:

```
<html>
<center>
<h1> The hostname of this webserver is {{ ansible_hostname }}</h1>
<h3> It is running on {{ ansible_os_family}}system </h3>
</center>
</html>
```

Dit is een simpele HTML pagina waar we gebruik maken van twee variabelen: ansible_hostname en ansible_os_family. Hier hoeft men deze variabelen niet manueel in te stellen, aangezien Ansible dit op zich neemt.

ansible_hostname vraagt de hostname van het systeem op en ansible_os_family zoekt uit welk type besturingssysteem actief is. Dit kan RedHat, Debian, Ubuntu, Windows, Arch_linux, enzovoort zijn.

[1]https://github.com/pallets/jinja

Het resultaat zou er als volgt kunnen uitzien:

```
<html>
<center>
<h1> The hostname of this webserver is Webserver01</h1>
<h3> It is running on Debian system </h3>
</center>
</html>
```

Meta

Metadata geef extra informatie over de role en is belangrijk als men een role wenst te plaatsen op Ansible Galaxy, een online platform waar men rollen met elkaar kan delen.

Naast vanzelfsprekende informatie zoals de naam van de role, de auteur, een omschrijving, eventuele tags is het ook mogelijk om 'dependencies' mee te geven. Het is namelijk mogelijk dat een role vertrouwt op een andere role om te kunnen functioneren. Wanneer men via Ansible Galaxy een role download wordt er in de metadata gekeken of alle benodigde rollen aanwezig zijn en indien dit niet het geval mocht zijn, worden deze geïnstalleerd.

2.0.2 Op welke manieren test men Ansible rollen?

Vooraleer een role wordt toegepast binnen een productieomgeving zou het uiteraard interessant zijn mocht deze eerst getest kunnen worden. In de praktijk bestaan hier vele oplossingen voor, elk met hun voor- en nadelen.

Virtuele machine

Een virtuele machine of afgekort 'VM' is een volledig besturingssysteem dat bovenop een huidig besturingssysteem wordt gevirtualiseerd. Dankzij een VM kan men een Ansible role testen met de volledige zekerheid dat er geen compatibiliteitsproblemen zullen zijn, wat soms het geval kan zijn bij containers. Een nadeel aan VM's is echter dat de workflow redelijk inefficiënt kan zijn als men met meerdere VM's gaat werken. In de meeste gevallen worden Ansible roles gemaakt met als doel dat ze op zo veel mogelijk Linux distributies kunnen worden uitgevoerd. Vagrant, een VM-orchestration programma, probeert dit probleem op te lossen door de VM's zelf te beheren en daarnaast is er de mogelijkheid dat de Ansible code automatisch wordt losgelaten op deze VM's. Een pijnpunt blijft echter wel dat VM's redelijk veel 'resources' vragen van de hardware, dus probeert men dit tegen te gaan door met containers te werken, zoals die van Docker.

Software containers

Docker

Docker [5] is een *Platform as a Service* type software dat wordt gebruikt om softwareapplicaties te verpakken in zogenaamde containers. Deze containers draaien bovenop de kernel, een deel van het besturingssysteem, en hebben als voordeel dat ze geïsoleerd zijn van elkaar. Elke container is gevuld met de vereiste softwarebenodigdheden en configuratiebestanden zodat ze volledig zelfstandig kunnen draaien zonder dat het besturingssysteem zelf hoeft in te grijpen.

In de IT wereld, en om specifieker te zijn de branche die zich bezighoudt met opzetten van softwareapplicaties op servers, doet het probleem zich vaak voor dat een bepaalde applicatie op één systeem wel werkt, en op andere dan weer niet. Ook al volgt men dezelfde stappen om een specifieke applicatie aan de praat te krijgen. Docker speelt met behulp van containers hier op in. Doordat de container reeds gevuld is met alle benodigde pakketten en configuratiebestanden zou men in theorie op elke machine die Docker draait hetzelfde resultaat moeten krijgen. Via een online platform genaamd de Docker Hub [6] kan men de blauwdrukken van een container met elkaar delen. Op deze manier kan gegarandeerd worden dat een applicatie kan functioneren zonder compabiliteitsproblemen.

Het concept achter de container is niet nieuw. De eerste prototypes dateren terug tot 1979 maar het is onweerlegbaar om te zeggen dat de introductie van Docker en de populariteitexplosie van containers hand in hand gaan. (Datadog, 2018)]

Podman

Podman [7], ontwikkeld door Redhat, is een container engine tool die als doel heeft om een volwaardig alternatief te zijn voor Docker. De syntax om de tool te bedienen is bijna identiek aan die van Docker, wat ervoor moet zorgen dat Docker-gebruikers hier snel mee aan de slag zouden kunnen gaan.

Podman biedt enkele interessante functies aan, maar de belangrijkste voordelen voor de meeste gebruikers zijn de volgende twee eigenschappen:

- Daemonless. Of anders gezegd, Podman hoeft geen proces in de achtergrond te draaien om te kunnen functioneren
- Rootless or root. Podman kan zowel op admin accounts als op normale accounts draaien

Docker werkt met een Daemon of anders gezegd een client/server model. De gebruiker die een container wenst op te zetten geeft een commando in dat via een API wordt afgehandeld door de Docker daemon. Podman werkt op een andere manier dat bekend staat als de ‘Fork and Exec‘ model, een methode waar bij elk nieuw proces dat uitgevoerd wordt een nieuwe programma wordt aangemaakt. („Fork and Exec”, g.d.)

[5] https://www.docker.com/
[6] https://hub.docker.com/
[7] https://podman.io/

Podman is ontwikkeld op basis van de ideeën van de Open Container Initiative, een project dat open standaarden opstelt voor virtualisatie op het niveau van besturingssystemen, inclusief Linux containers. Dankzij OCI is er de mogelijkheid om *unprivileged* containers te creeëren, wat een stuk veiliger is dan *privileged* containers. (Graber, 2014). OCI doet dit door gebruik te maken van zogenaamde namespaces en cgroups.

Om hier even kort op in te gaan, namespaces is een deel van de Linux kernel dat kernel middelen kan verdelen in zogenaamde partities zodat een bepaald proces toegang heeft tot de middelen van een bepaalde partitie terwijl een ander proces dan toegang heeft tot andere middelen. Het is een fundamenteel aspect als een machine met containers wilt werken.

Cgroups, ofwel Control groups, is een andere feature van de Linux kernel dat toelaat om groupen van processen te beheren, te beperken en auditeren. Ook dit is belangrijk om containers te kunnen opstellen.

Namespaces heeft namelijk ook een grote impact bij het gebruik van SELinux. Security-Enhanced Linux is een onderdeel van Linux dat een reeks van security policies aanbiedt, ondermeer het gebruik van de zogenaamde *Mandatory Access Control*. Het is een module die bovenop de kernel van een Linux systeem draait en een extra laag security biedt bovenop de *permissions feature* die de Linux besturingssystemen gebruiken. In essentie laat het toe om via labels de toegang van processen, gebruikers, enz. te beperken tot de labels die zij toegewezen kregen. („Arch Wiki - SELinux”, 2020)

SELinux is in tegenstelling tot Docker niet namespaced wat betekent dat het onmogelijk is om SELinux policies toe te passen in de container zelf, enkel daarrond. Zo moeten gebruikers die zowel SELinux als Docker op hun systeem draaien aan SELinux meegeven dat Docker de toestemming krijgt om containers op te zetten en beroep kan doen op systeemmiddelen. Verder is het ook belangrijk naar de ontwikkeling van Ansible rollen toe, aangezien rollen waarbij SELinux gebruikt wordt, niet kunnen getest worden via Docker containers.

Black-box testing

Black-box testing is een software testing techniek die de functionaliteiten van een applicatie overloopt zonder dat het ’onder de motorkap’ van de applicatie gaat kijken, bij wijze van spreken. De tester weet wat het programma doet, welk resultaat hij verwacht, maar hoe het programma dit doet is voor hem minder van belang. (Patton, 2006)

Bij systeem- en netwerkbeheer zou black-box testen bijvoorbeeld kunnen nagaan of een server zijn taak wel degelijk uitvoert. Een voorbeeld hiervan is het tonen van een webpagina of de toegang geven tot een fileserver applicatie. Veronderstel dat men een Ansible role ontwikkelt die als taak heeft om de Apache webserver te installeren, dan zou na de afloop van de implementatie van de role een black-box test kunnen uitgevoerd worden om te zien of de webserver effectief wel pagina’s toont, of bijvoorbeeld alsook een beveiligde verbinding via HTTPS ondersteunt. Hoe de applicatie of server dit doet is niet van belang, enkel de resultaten zijn hier belangrijk.

2.0.3 Continuous Integration/ Development

Travis CI

Continuous Integration is een software ontwikkeling aanpak waarbij ontwikkelaars meerdere malen per dag een kopie van hun code synchroniseren naar een centrale code opslag. Op deze opslag kan men dan bijvoorbeeld het volledige product laten testen om de stand van zaken van het project te bepalen. Deze aanpak wordt vooral gebruikt bij Extreme Programming. (Fowler, 2016)

Hoe vertaalt dit zich naar het testen van een Ansible role? De meeste role ontwikkelaars publiceren namelijk hun role op Ansible Galaxy en alsook een kopie van de source code op Github, Gitlabs etc. Andere ontwikkelaars in de open-source wereld kunnen deze code bekijken en een zogenaamde fork maken (een openbare kopie van deze code met de volledige referentie naar het originele project). Met zo een fork kan men een verbetering maken en dan kan men aan de ontwikkelaar van het originele project vragen of de verbetering mag opgenomen worden bij zijn project. Deze vraag staat gekend als een *Pull request*. („About pull requests", 2020)

Via een Pull request kunnen de twee ontwikkelaars discussiëren over de aanpassingen hoe deze geïmplementeerd zouden kunnen worden. Voor de originele ontwikkelaar zou het wel interessant zijn mocht hij bij elke Pull request op een gemakkelijke manier een bewijs kunnen zien dat de nieuwe code effectief werkt.

Dit is waar CI omgevingen zoals Travis CI hun entree doen. Men heeft de mogelijkheid om Travis CI te koppelen aan een bepaalde Github repository. Als men dan een commit of een pull request aanbrengt kan men, door vooraf een configuratiebestand op te stellen weliswaar, de nieuwe code testen zonder dat hier een machine voor dient gebruikt te worden. Travis CI werkt namelijk volledig in de cloud.

Molecule

Zoals beschreven bij de inleiding wordt er onderzocht of Molecule een praktische oplossing kan zijn voor Ansible ontwikkelaars. Binnen de community begint Molecule in een langzaam tempo populariteit te winnen. Zo is het gebruik van deze software te vinden in code repositories van toch wel grote projecten zoals Nginx (een webserver applicatie) [1] of worden gepromoot door bekende figuren binen de community zoals Jeff Geerling [2], die eveneens een boek heeft geschreven over hoe men Ansible kan gebruiken in de DevOps wereld of Kubernetes. (Geerling, 2015) en (Geerling, 2020). Op het internet zijn er vele artikels en videos te vinden die Molecule demonstreren en het nut ervan uitleggen. Ook de documentatie van Molecule zelf[3] is zeker niet iets wat men mag onderschatten, ondanks de samenvattende vorm waar deze in geschreven is. De beste manier om up-to-date te blijven met deze software is door de Github pagina [4] in het oog te houden samen met de issues

[1] https://github.com/nginxinc/ansible-role-nginx
[2] https://www.jeffgeerling.com/
[3] https://molecule.readthedocs.io/en/latest/
[4] https://github.com/ansible-community/molecule

pagina.

Linux init systemen en SystemD

Wanneer een systeem met een UNIX-besturingssysteem opstart wordt bij de initialisatie van de kernel een bepaalde applicatie opgeroepen die als PID (Process Identification Number) 1 toegewezen krijgt. Deze applicatie neemt dan het stuur over en start alle andere applicaties op die belangrijk zijn voor het besturingssysteem om te kunnen functioneren. Bij zowat alle moderne grote Linux distributies is deze applicatie SystemD. Er bestaan nog andere zogenaamde init systemen naast SystemD, maar voor dit onderzoek zijn deze niet relevant.

Waarom is dit belangrijk? Bij Docker containers is het namelijk de bedoeling dat de applicatie die men wenst te draaien binnen de container een PID van 1 krijgt. Docker is namelijk gemaakt om processen te isoleren, niet delen van een besturingsysteem. Om die reden zijn SystemD of andere init systemen niet aanwezig op images en containers aangezien dit een extra vorm van overhead en complexiteit zou veroorzaken.

3. Methodologie

Om het nut van Molecule bij het testen van Ansible rollen te bewijzen werd er gekozen om deze software vanuit verschillende invalshoeken te bekijken, met elk als doel om op één of meerdere onderzoeksvragen een antwoord te kunnen bieden. Als eerste wordt er een scenario rechtstreeks op het systeem uitgevoerd, dit om de functionaliteiten van Molecule te demonstreren en hoe dit zich kan vertalen in het verloop van het onderzoek. Ten tweede wordt er gebruik gemaakt van Github en Travis CI om aan te tonen hoe nuttig Molecule kan zijn in een omgeving met meerdere ontwikkelaars gevolgd door een scenario waarbij er gebruik wordt gemaakt van een Ubuntu Virtuele Machine. Dit is dezelfde VM die draait op Travis CI zodat er op die manier kan worden bekeken of een verschil in Linux distributies een invloed heeft op het resultaat van een Molecule test. Tenslotte wordt er ook getest of Podman een oplossing kan bieden voor de beschreven scenario's in plaats van Docker.

Gedurende dit onderzoek werd er gebruik gemaakt van de volgende hardware en software:

- Besturingssysteem: Pop!_OS 20.04 LTS x86_64 (Ubuntu variant)
- Host: XMG FUSION 15 (XFU15L19) Late 2019
- Kernel: 5.4.0-7626-generic (Linux)
- CPU: Intel i7-9750H (12) @ 4.500GHz
- Max Memory: 15856MiB

3.1 Methode 1: Lokaal testen met Molecule en Docker

3.1.1 Requirements:

- Linux of MacOS besturingssysteem.
- Ansible geïnstalleerd op het systeem
- Docker geïnstalleerd op het systeem
- Een virtuele Python omgeving

Ansible kan niet werken van uit een Windows systeem [1] dus het is aangeraden om met UNIX-besturingssystemen zoals Linux of MacOS te werken. Het is echter in theorie mogelijk om Ansible te draaien op Windows via de *Windows Subsystem for Linux* maar dit wordt officieel niet door Ansible ondersteunt en wordt ten sterkste afgeraden in productie-omgevingen. Er is geen specifieke Linux distributie vereist, maar de beste ondersteuning gaat men vinden bij populaire distributies zoals Debian, Ubuntu, Fedora, Arch Linux of hun varianten.

Alsook dient Docker gebruiksklaar te zijn op het systeem. Installatieinstructies voor Docker zijn via deze link te vinden: https://docs.docker.com/get-started/

Het is aangeraden om gebruik te maken van een Python virtual environment. Via PIP, een Python package manager, kan men dan snel de benodigde software bekomen:

```
$ python3 -m pip install molecule docker
```

Ansible wordt automatisch geïnstalleerd mocht deze nog niet aanwezig zijn op het systeem aangezien het een *dependency* (een vereiste) is van Molecule. De Docker package die hier wordt meegegeven is een driver zodat de Molecule python code kan communiceren met Docker en zijn containers.

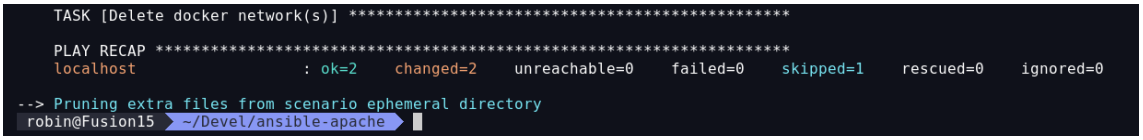
3.1.2 Inleiding

Voor deze methode worden de aanbevelingen en 'defaults' gevolgd die beschreven staan in de documentatie van Molecule. Dit houdt concreet in dat er eerst op een standaard desktop systeem met Docker en Ansible getest zal worden. Molecule zal in staat zijn om via Docker containers op te zetten en daarna te laten provisioneren via Ansible.

3.1.3 Molecule toevoegen aan een role

Als eerste moet Molecule worden toegevoegd aan een Ansible role. Molecule voorziet een commando om zelf een nieuwe Ansible role aan te maken, of voegt de benodigde configuratiebestanden toe aan een bestaande role. Om molecule in zijn volledigheid te

[1] https://docs.ansible.com/ansible/latest/user_guide/windows_faq.html



Figuur 3.1: Verwachte eindresultaat na een eerste test

kunnen demonstreren wordt er een nieuwe role aangemaakt waar enkele tasks aan zullen toevoegd worden.

```
$ molecule init role ansible-role-test -d docker
```

De '-d' optie geeft de mogelijkheid om te specifiëren welke driver er dient gebruikt te worden om de containers aan te maken (Docker of Podman). De rolnaam is naar keuze in te stellen. Meer opties zijn alsook beschikbaar door 'molecule init role –help' uit te voeren in een terminal.

Er is nu een nieuwe folder aangemaakt op de huidige locatie. Via 'molecule test' kan er een eerste controle uitgevoerd worden of alles naar behoren werkt.

Belangrijk!: Indien een foutmelding als resultaat wordt gegeven omtrent 'molecule[docker]', deactiveer tijdelijk de virtual python environment mocht deze actief zijn en installeer manueel de Docker driver voor de huidige gebruiker:

```
$ python3 -m pip install -user docker
```

Figuur 3.1 toont welk eindresultaat er verwacht wordt na een geslaagde molecule test.

3.1.4 Werking van Molecule

Molecule werkt met een zogenaamde 'Test Matrix' die beschrijft welke taken er worden uitgevoerd naargelang het meegeven commando. Een 'molecule test' zal de volledige Test matrix overlopen en uitvoeren. De relevante onderdelen van zo een test matrix worden beknopt even beschreven:

- destroy: deactiveert en verwijdert test containers mochten deze nog draaien na een vorige sessie
- syntax: zoals de naam beschrijft zal het eerst de configuratiefiles controleren of er geen syntaxfout is gemaakt
- create: creeërt de containers
- converge: voert de ansible rollen uit op de containers
- idempotence: Bij het uitvoeren van een Ansible role betekent idempotence dat het systeem in één keer naar de gewenste eindtoestand wordt gebracht. Als dezelfde

role een tweede keer wordt uitgevoerd op het systeem zou bij de PLAY RECAP als resultaat "Changed=0"moeten verschijnen.
- verify: voert een Ansible of TestInfra code uit om extra testen uit te voeren. Bijvoorbeeld of een bepaald programma actief is op het systeem.

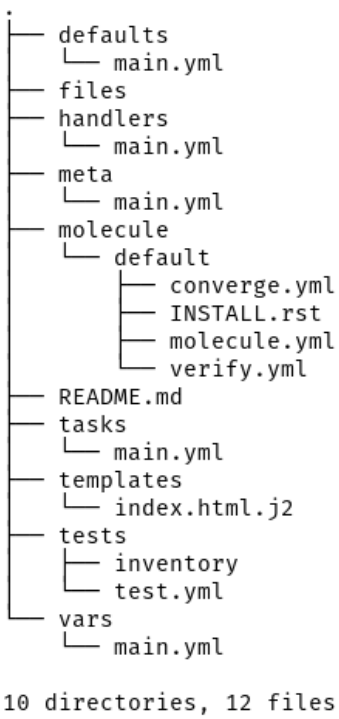
Het ‘test‘ commando zal de volledige test matrix uitvoeren. Dit commando is handig om met minimale gebruikersinteractie de molecule test uit te voeren.

‘Molecule converge‘ zal vooral tijdens de ontwikkeling van de role gebeuren. Het is echter niet handig om tijdens de ontwikkelings fase een ‘molecule test‘ te gebruiken omdat anders de containers worden aangemaakt, vervolgens worden geconfigureerd met Ansible en tenslotte dan weer worden verwijderd. Wanneer een role faalt, wil een ontwikkelaar de mogelijkheid hebben om de container te kunnen inspecteren en het probleem te proberen vaststellen. *Converge* laat toe om Molecule de containers te laten aanmaken en te configureren met Ansible. Ze worden aan het einde van dit commando niet vernietigd. Op die manier kan met via ‘molecule login‘ bijvoorbeeld inloggen op de container en troubleshooting uitvoeren.

3.1.5 Bespreking van Molecule structuur

Beschouw figuur 3.2. Men kan al snel opmerken dat het Molecule init commando een Ansible role heeft aangemaakt volgens de huidige best practices zoals aparte folders voor tasks, templates, vars, etc. Molecule heeft alsook een eigen folder voorzien waar enkele zaken naar wens kunnen worden afgesteld. Als eerste wordt de belangrijkste configuratie file geïnspecteerd: ‘molcule.yml‘ onder ‘molecule/default’.

Op dit moment is het echter nog maar een klein bestand. Via molecule.yml wordt er geconfigureerd welke dependencies moeten worden gebruikt, met welke driver Molecule (Docker of Podman) moet gewerkt worden en welke provisioner er gebruikt dient te worden. Het belangrijkste deel is echter het gedeelte "platforms". Onder deze rubriek specifieert men welke containers Molecule moet aanmaken en testen. Op dit moment, en by-default, wordt Centos 7 als image gebruikt. Elke Linux Image kan men hier gebruiken zolang de rol die men zal schrijven rekening houdt met distro specifieke zaken. (Package manager, filesysteem structuren, etc).



Figuur 3.2: Structuur van de Ansible role met Molecule

```
---
dependency:
  name: galaxy
driver:
  name: docker
platforms:
  - name: centos7
    image: docker.io/pycontribs/centos:7
    privileged: true
    command: "/usr/sbin/init"
    tty: True
provisioner:
  name: ansible
verifier:
  name: ansible
```

Figuur 3.3: Een voorlopig kleine molecule configuratie file

3.1.6 Een task schrijven en toevoegen aan de test

Observeer ‘converge.yml‘ onder ‘molecule/defaults/‘. Dit is niets anders dan een Ansible playbook van Molecule dat op zijn beurt de role in ontwikkeling zal aanroepen en daarna zal uitvoeren op de verschillende containers.

```
---
- name: Converge
  hosts: all
  tasks:
    - name: "Include ansible-apache"
      include_role:
        name: "ansible-apache"
```

Figuur 3.4: converge.yml

Op dit moment zal de role echter nog niets uitvoeren op de container aangezien er nog geen taken zijn geschreven die hij kan uitvoeren. Om dit te remediëren wordt een simpele task geschreven die de Apache Webserver zal downloaden, een index.html (een eenvoudige webpagina) op de juiste plek zal zetten en alsook zal testen of de service actief is. Tenslotte wordt het HTTP protocol ook toegevoegd aan de firewall mocht deze actief zijn.

In ‘main.yml‘ onder de Tasks folder zullen dus de volgende tasks worden toegevoegd:

```
# tasks file for ansible-apache
- name: "Ensure required packages are present"
  yum:
    name: "{{ package_list }}"
    state: present

- name: "Ensure latest index.html is present"
  template:
    src: index.html.j2
    dest: /var/www/html/index.html

- name: "Ensure httpd service is started and enabled"
  service:
    name: "{{ item }}"
    state: started
    enabled: true
  with_items: "{{ service_list }}"

- name: "Whitelist http in firewalld"
  firewalld:
    service: http
    state: enabled
    permanent: true
    immediate: true
```

Figuur 3.5: Enkele tasks dat de httpd webserver zal downloaden en configureren

Bij de tasks worden er alsook vars en templates gebruikt:

```
#vars file for ansible-apache
package_list:
  - httpd
  - firewalld
service_list:
  - httpd
  - firewalld
```

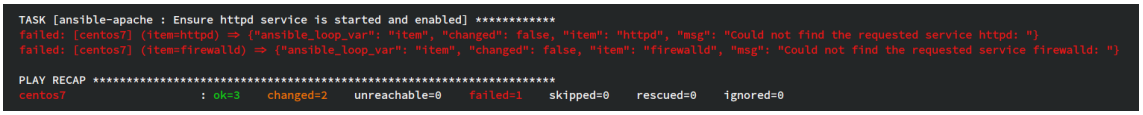
Figuur 3.6: vars/main.yml

In de map templates wordt er alsook een Jinja2 index.hmtl pagina aangemaakt.

```
<div style="text-align: center">
<h2>Managed by Ansible</h2>
</div>
```

Figuur 3.6 toont de vars file met de packages en services die Ansible dient te behandelen.

Wanneer men nu een ‘molecule converge‘ uitvoert zal men echter opmerken dat er een foutmelding verschijnt. Deze ziet er ongeveer zo uit:



Figuur 3.7: Foutcode als resultaat. Ansible geef mee dat de services niet te vinden zijn.

De error geeft aan dat Ansible de httpd en firewalld services niet kon vinden. Dit is niet omdat ze niet geïnstalleerd zijn, maar omdat Ansible normaal communiceert met Systemd om deze informatie op te vragen en de processen te starten. Systemd is namelijk een programma dat op de meeste Linux besturingssystemen draait om processen te beheren. Het probleem is echter dat Systemd niet aanwezig is bij deze container images. Via de terminal kan men ‘molecule login‘ uitvoeren en op die manier controleren of SystemD effectief niet aanwezig is op de container, bijvoorbeeld door een systemctl commando uit te voeren.

```
$ molecule login
[root@centos7 /]# systemctl start httpd
Failed to get D-Bus connection: Operation not permitted
```

Het is duidelijk dat de shell het commando niet kan doorgeven naar "systemd"simpelweg omdat het gewoon niet aanwezig is op de image. Dit zorgt echter wel voor problemen.

Developers van Ansible rollen vertrouwen nu eenmaal op systemd om procssesen te beheren. Hoe lost men dit dan op?

Gelukkig bestaan er Docker images waarbij systemd ingebouwd is en klaar voor gebruik. Bij vele developers kan deze manier van werken wenkbrauwen doen fronsen aangezien het normaal *not-done* is om besturingssysteem specifieke programma's mee te nemen in containers. (BMitch, 2016) In dit geval is het echter wel noodzakelijk en dit is namelijk slechts een testomgeving, niet een productieomgeving.

3.1.7 Een nieuwe Docker image gebruiken

Wanneer men nog eens de configuratiefile van molecule opent (./molecule/default/molecule.yml dus) dan is er te zien dat er een standaard Centos 7 image wordt gebruikt met zo goed als geen andere opties. Om systemd aan de praat te krijgen zullen er enkele aanpassingen moeten aangebracht worden. Eerst en vooral zal een image gebruikt worden waar systemd aanroepbaar is. Geerlingguy op Docker Hub voorziet zulke images [1]. Voor dit onderzoek zal men zijn Centos 7 image gebruiken. Alsook worden enkele opties toevoegd om compatibiliteit en stabiliteit problemen te voorkomen. Figuur 3.8 toont deze aanpassingen.

```
--
dependency:
  name: galaxy
driver:
  name: docker
platforms:
  - name: centos7
    image: "geerlingguy/docker-centos7-ansible:latest"
    command: "/usr/sbin/init"
    volumes:
      - /sys/fs/cgroup:/sys/fs/cgroup:ro
    privileged: true
    pre_build_image: true
    tty: True
provisioner:
  name: ansible
verifier:
  name: ansible
```

Figuur 3.8: Aangebrachte aanpassingen in Molecule configuratie file

'molecule destroy' zal de huidige actieve containers deactiveren en verwijderen. Voert men nadien opnieuw een converge uit, dan zouden er deze keer geen errors meer mogen zijn. SystemD is dus actief op de container.

[1] https://hub.docker.com/u/geerlingguy/

3.1.8 Molecule Verify code aanbrengen

Molecule voorziet een aparte fase waar men 'verificatie' code kan schrijven voor de Ansible rollen die werden uitgevoerd. Voor een HTTPD server kan dit een systemd check zijn of de service actief is of men kan via wget de index.html pagina ophalen en testen of het overeenkomt met het verwachte resultaat. Er is keuze uit Ansible testcode of anderzijds Python's TestInfra library.

Op zich kan de code die services test in de Tasks worden overgenomen in de Verify sectie.

3.1.9 Molecule testen op meerdere containers

Er is ondertussen al een grote stap voorwaarts gezet waarbij men op een snelle manier Ansible code kan testen met behulp van een Centos Docker container. In de realiteit gaat men echter voor meerdere Linux distributies code proberen schrijven, meestal voor Red Hat en Debian gebaseerde distro's. Omwille van deze reden zal het dan ook interessant zijn om de Ansible role op een Debian gebaseerde distro te testen, bijvoorbeeld op Debian zelf of een Ubuntu image.

Voor Molecule is deze aanpassing zeer eenvoudig. Men hoeft enkel in de molecule configuratiefile onder 'platforms' een entry te maken voor deze image. In feite kan de code voor de Centos7 image volledig overgenomen worden buiten de effectieve image die dient gebruikt te worden uiteraard. Het wordt dus snel duidelijk dat er op een zeer schaalbare manier kan worden getest. Dit langs de Molecule kant uiteraard. In de Ansible code, vooral onder de 'tasks', moeten er nog enkele onderdelen worden aangepast. In Debian gebaseerde Linux distributies staat de Apache webserver niet bekend als 'httpd' in de package manager maar eerder als 'apache2'. Alsook hebben Debian gebaseerde distributies een eigen firewall software die ufw heet.

Om dit probleem op te lossen, dient men in de Ansible code met volgende zaken rekening houden:

- Code schrijven waardoor Ansible een distributie kan herkennen
- vars aanpassen naargelang het type distributie
- specifieke taken uitvoeren per distributie type

Een mogelijke oplossing is te vinden in figuur 3.9 en 3.10. Via een Ansible specifieke variabele kan men het type distributie opvragen en er op deze manier mee verder werken. Alsook zijn er onder de vars map twee bestanden gemaakt, eentje genaamd Redhat.yml en de andere Debian.yml. Zoals de namen al doet vermoeden zullen er in deze bestanden de specieke variabelen ingevuld worden per distrubutie familie. Om het voorbeeld van de Apache webserver te hernemen; onder RedHat staat de software bekend als 'httpd' en bij Debian als 'apache2'.

Met deze variabelen kan er een onderscheid gemaakt worden tussen de verschillende distributiefamilies. Nu rest er enkel deze var bestanden effectief toe te passen bij de Tasks.

```
#vars file for
#RedHat Linux distributions
package_list:
  - httpd
  - firewalld
service_list:
  - httpd
  - firewalld
```

Figuur 3.9: RedHat vars file

```
#vars file for Debian/
#Ubuntu Linux distributions
package_list:
  - python3-apt
  - apache2
  - ufw
service_list:
  - apache2
  - ufw
```

Figuur 3.10: Debian vars file.

Ansible voorziet een variabele waardoor men op een snelle manier kan bepalen met welk type distributie de machine werkt. Deze aanpassingen aan de task file zijn te zien op figuur 3.11.

Wanneer men nu een molecule test uitvoert waarbij men een Centos 7 en Debian 10 image gebruikt dan kan men opmerken dat Molecule zich hier goed kan aanpassen.

```
---
#Determine OS type and family
- name: Source specific variables
  include_vars: "{{ item }}"
  with_first_found:
    - "{{ ansible_distribution }}.yml"
    - "{{ ansible_os_family }}.yml"
  tags: httpd,pretask

# tasks file for ansible-apache
- name: "Ensure required packages are present"
  yum:
    name: "{{ package_list }}"
    state: present

- name: "Ensure latest index.html is present"
  template:
    src: index.html.j2
    dest: /var/www/html/index.html

- name: "Ensure httpd service is started and enabled"
  service:
    name: "{{ item }}"
    state: started
    enabled: true
  with_items: "{{ service_list }}"

- name: "Whitelist http in firewalld"
  when: ansible_facts['os_family'] == "RedHat"
  firewalld:
    service: http
    state: enabled
    permanent: true
    immediate: true

- name: "Whitelist http in ufw"
  when: ansible_facts['os_family'] == "Debian"
  ufw:
    rule: allow
    name: WWW
```

Figuur 3.11: Aangepaste tasks configuratiefile rekening houdend met verschillende Linux distributies

3.2 Methode 2: Molecule testen in een CI/CD omgeving

De tweede methode die wordt bekeken is bijna identiek met de vorige met als verschil dat de code nu zal worden opgeladen naar Github en een service zoals Travis CI. Met behulp van Travis CI kan men snel code testen die op Github werd geplaatst.

3.2.1 Requirements:

- Ansible geïnstalleerd op het systeem
- Molecule geïnstalleerd op het systeem
- Docker software en python driver geïnstalleerd op het systeem
- Een Github repository om de code te uploaden
- Een Travis CI account met een project dat gelinkt is met de Github repo

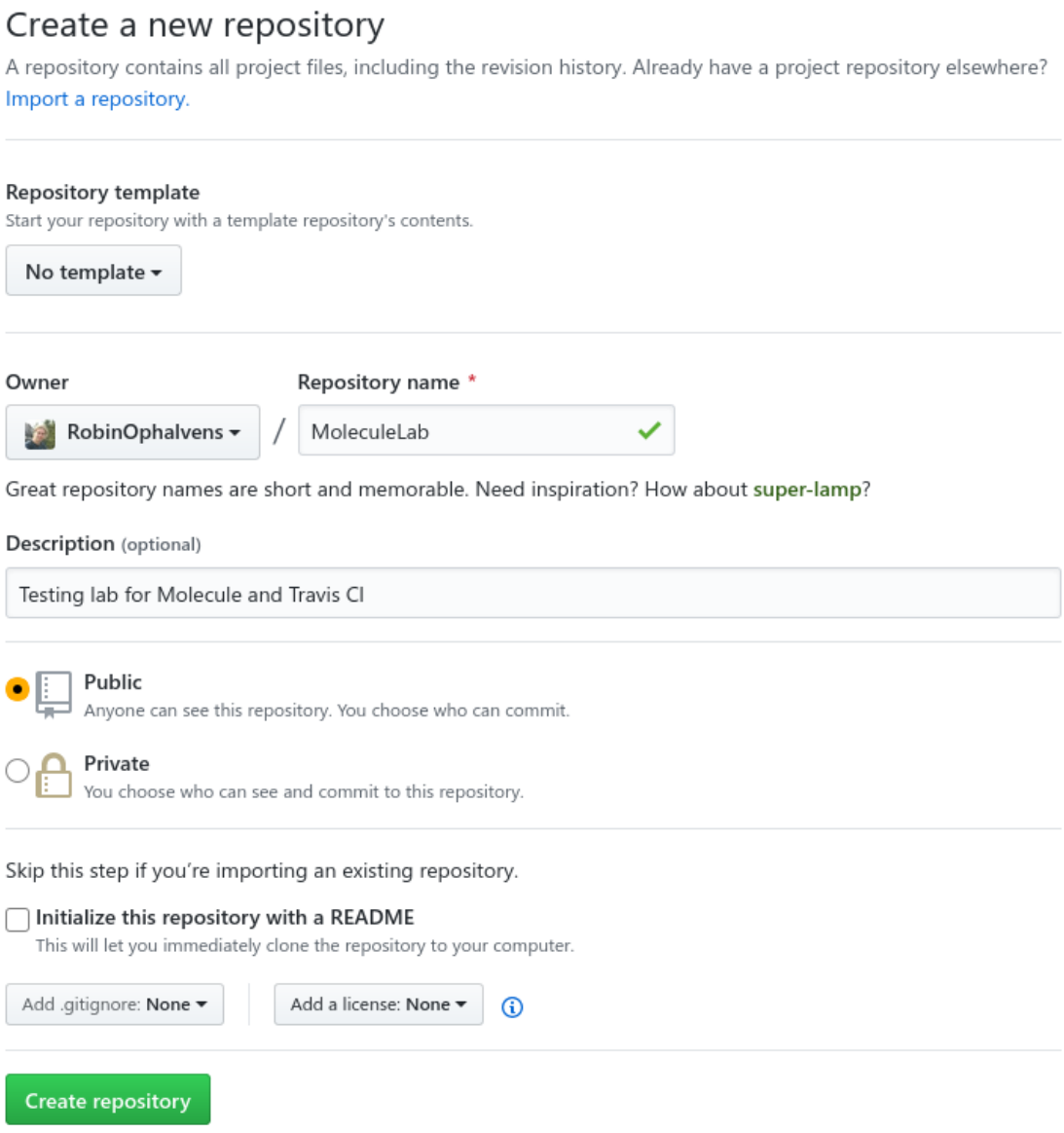
De requirements van de eerste methode zijn hier hetzelfde. Refereer naar deze, op pagina 34, mochten bovenstaande requirements (met uitzondering de laatste twee) nog niet vervuld zijn.

3.2.2 Github repository aanmaken en travis.yml configureren

De eerste stap bestaat eruit om een Github repository te maken waar Travis CI later aan gekoppeld kan worden. De repository zelf hoeft niet speciaal te zijn, de enige requirement is dat het publiek beschikbaar moet zijn als men met de gratis versie Travis CI wenst te werken. Om problemen later te voorkomen krijgt deze repository best dezelfde naam als de role waar Molecule mee werkt. Figuur 3.12 toont een mogelijk voorbeeld:

Na de repository te *clonen* (of met andere woorden downloaden) is het tijd om onze code hier aan toe te voegen. Om ervoor te zorgen dat Travis CI een test kan uitvoeren dient er verteld te worden wat het platform moet doen met de meegegeven code. Om dit te doen creeërt men een bestand in het project genaamd .travis.yml. Lijn 1 vertelt met welke programmeertaal Travis moet werken en er wordt verder alsook aangegeven dat Docker nodig is voor de test. Daarop volgen de softwarepakketten die dienen geïnstalleerd te worden zodat Molecule kan functioneren.

Voor *debugging* wordt alsook kort op de de console weergegeven op welke versie Ansible en Molecule draaien en tenslotte wordt Molecule uitgevoerd op dezelfde manier zoals dit in Methode 1 werd gedaan.



Figuur 3.12: Creatie van Github repository.Vergeet niet om de repository public te maken!

```
---
language: python

# Use the new container infrastructure
sudo: required

#Enable docker support
services:
- docker

install:
# Install dependencies for Molecule test
- python3 -m pip install ansible
- python3 -m pip install pytest
- python3 -m pip install testinfra
- python3 -m pip install molecule
- python3 -m pip install docker

# Check ansible and molecule version
- ansible -version
- molecule -version

# Create ansible.cfg with correct roles_path
- printf '[defaults]\nroles_path=../' >ansible.cfg

script:
- molecule test
```

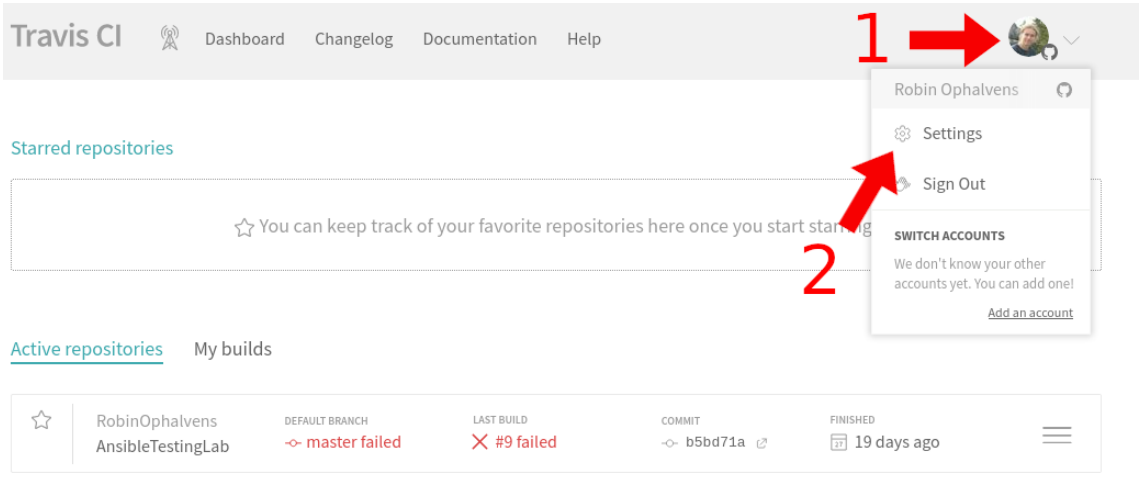
Figuur 3.13: .travis.yml bestand dat nodig is
om via Travis een Molecule test uit te voeren

3.2.3 Travis CI configureren en koppelen aan Github

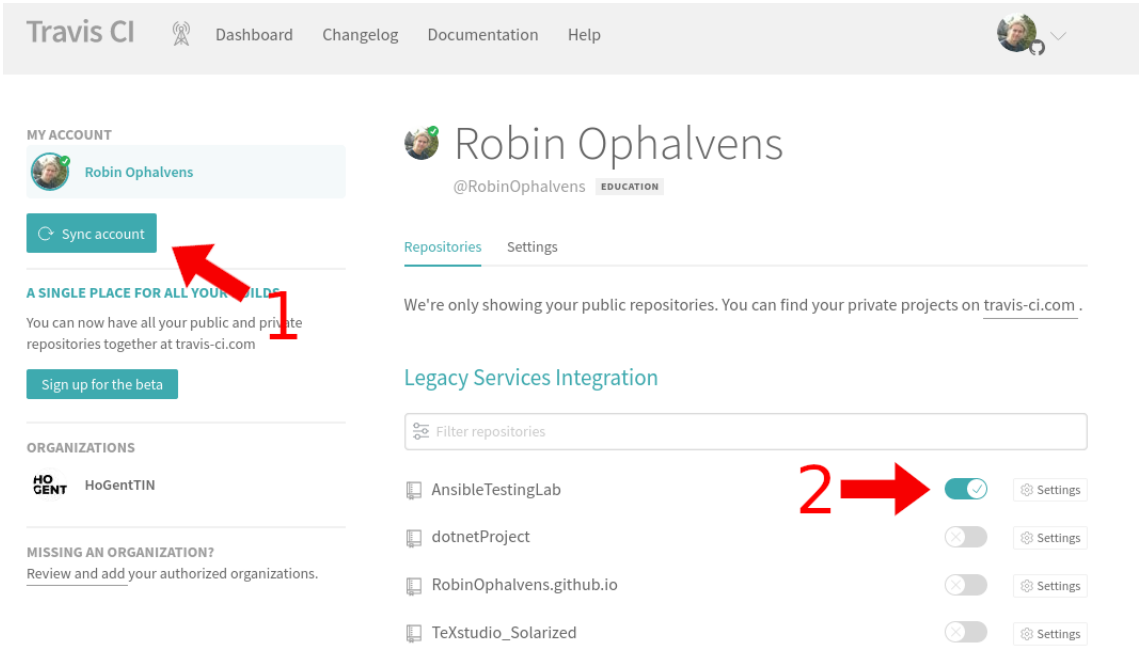
Nu de code klaar is om te testen is de volgende stap om Travis CI te koppelen aan de Github repository. Een korte demonstratie hoe dit wordt gedaan, zal met enkele figuren worden geïllustreerd.

Stappenplan

1. Maak een account aan op https://travis-ci.org mocht dit nog niet gedaan zijn
2. Ga naar opties zoals wordt getoond op figuur 3.14
3. Synchroniseer Travis CI met Github
4. Kies een Github repository waar Travis CI mee moet werken zoals wordt gedemonstreerd op figuur 3.15



Figuur 3.14: Startpagina van Travis CI



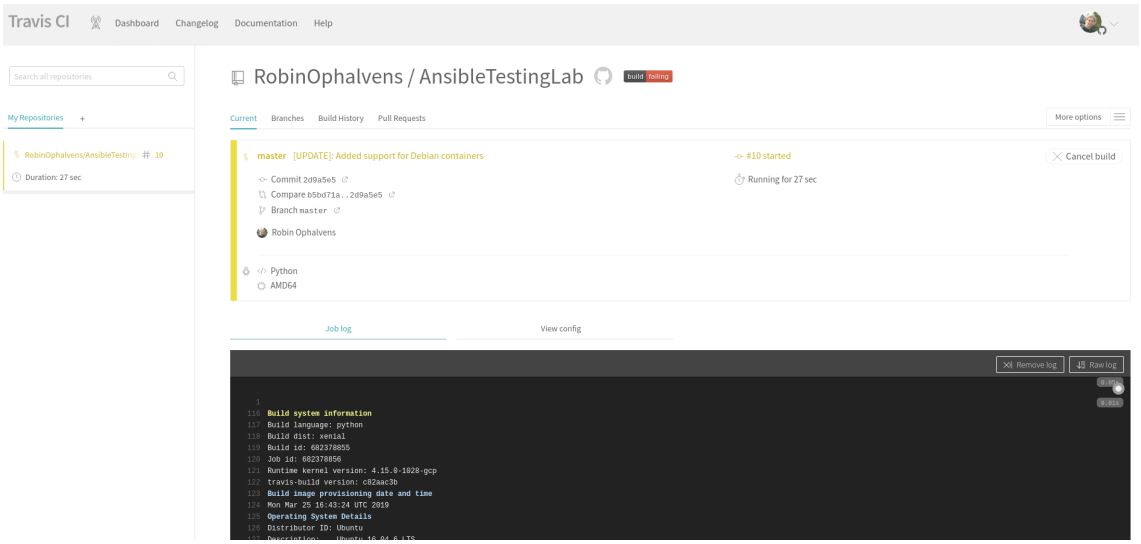
Figuur 3.15: Travis CI synchroniseren en koppelen aan Github

3.2.4 Een code wijziging testen met Travis CI

Nu alles klaar staat is het tijd om een commit te maken en deze te *pushen* naar de aangemaakte Github repository.

```
#Add the role to a commit
$ git add ./
$ git commit -m "[NEW]: First test with Travis"
$ git push
```

Na de push naar Github zal Travis CI spoedig beginnen. Op de Dashboard startpagina zal te zien zijn dat Travis is begonnen met de role te testen. Door op een knop rechts te drukken aan het begin van de zwarte box kan men gemakkelijk de voortgang opvolgen.



Figuur 3.16: De dashboard in Travis toont nu dat er een role wordt getest

Als alles goed is gegaan zou er aan het einde van de *Play Recap* niets in het rood mogen staan en zou de laatste lijn moeten bestaan uit *Done. Your build exited with 0.* Ook als men terug naar boven scrollt op dashboard zal het gele thema vervangen zijn door een groene. Redelijk vanzelfsprekend dus.

```
568     PLAY RECAP *********************************************************************
569     localhost                  : ok=2    changed=2    unreachable=0    failed=0    skipped=1    rescued=0    ignored=0
570
571 --> Pruning extra files from scenario ephemeral directory
572 The command "molecule test" exited with 0.
573
574
575 Done. Your build exited with 0.
```

Figuur 3.17: De Play recap. Als alles goed is gegaan is de exit code altijd 0

3.3 Methode 3: Molecule testen met een Ubuntu VM en Travis CI

In sommige gevallen kan het gebeuren dat we lokaal een ander resultaat krijgen in vergelijking met die Travis CI, desondanks dat het de doel van Docker is om juist dit probleem te elimineren. Om de verschillen tussen allerlei besturingssystemen te minimaliseren, kan er geopteerd worden om via een VM te werken die op hetzelfde besturingssysteem draait als die waarop de code wordt uitgevoerd in Travis CI, namelijk Ubuntu 16.04 LTS (Xenial).

Bij deze methode wordt er uitgelegd hoe een VM kan worden opgezet en alsook hoe men hier mee kan testen via Travis CI.

3.3.1 Requirements

- Ansible geïnstalleerd op het systeem
- Een software dat VM's kan virtualiseren (bij voorkeur Virtualbox of libvirt)
- Vagrant geïnstalleerd op het systeem
- Een Github repository om de code te uploaden
- Een Travis CI account met een project dat gelinkt is met de Github repo.

3.3.2 Voorbereiding: de virtuele machine opzetten

Om de workflow zo gebruiksvriendelijk mogelijk te houden zal er gekozen worden om met Vagrant te werken. Vagrant is een zogenaamd *Orchestration* programma dat Virtuele Machine's van platformen zoals VirtualBox of libvirt kan opstarten en beheren.

Vagrant kan men via de meeste package managers installeren of gewoonweg rechtstreeks van hun website: https://www.vagrantup.com/

Voor de meeste gebruikers zal virtualbox de beste keuze zijn aangezien deze gemakkelijk en snel te installeren is en vereist minimale configuratie. Opnieuw is virtualbox beschikbaar bij de meeste package managers, maar het is mogelijk om de software rechtstreeks van de ontwikkelaar te halen via deze link: https://www.virtualbox.org/

Maak ergens een map naar keuze en voer het commando **vagrant init** uit. Deze zal vagrant initialiseren in deze map en zal een zogenaamde *Vagrantfile* maken.

Een Vagrantfile is een configuratiebestand dat beschrijft op welke manier Vagrant te werk moet gaan. Zo kan er ingesteld worden welke software (of provider) er gebruikt moet worden om de Virtuele Machines op te zetten. Alsook kan er gekozen worden hoeveel resources elke VM krijgt, wat interessant voor deze methode is dat er ook gewerkt kan worden met shared folders.

Shared folders laten gemakkelijk toe om bestanden op het systeem te delen met de virtuele machine. Op deze manier kan de role die bij voorgaande methodes is opgesteld snel worden overgezet naar de VM.

Met tekstbewerker naar keuze wordt de inhoud van de *Vagrantfile* aangepast naar volgende structuur:

```
1  $script = <<-SCRIPT
2  echo 'Installing Python3, PIP and Ansible'
3  sudo apt-get -qq install python3 python3-pip
4  ssh-keygen -b 2048 -t rsa -f /home/$USER/.ssh/id\_rsa -q -N ""
5  SCRIPT
6
7  Vagrant.configure("2") do |config|
8          ##### DEFINE VM #####
9          config.vm.define "ubnt-01" do |config|
10                 #Naam instellen van de VM
11                 config.vm.hostname = "ubnt-01"
12                 #Welk besturingssysteem wordt er gebruikt
13                 config.vm.box = "generic/ubuntu1604"
14                 #Update check uitzetten
15                 config.vm.box_check_update = false
16                 #De folder 'data' wordt naar de VM
17                 #overgezet onder locatie /vagrant
18                 config.vm.synced_folder './data', '/vagrant', type: 'rsync'
19                 #Het ip adres wordt ingesteld
20                 config.vm.network "private_network", ip: "192.168.18.9"
21                 #Max aantal Ram geheugen dat de VM mag gebruiken
22                 config.vm.memory: 2048
23                 #Script dat wordt uitgevoerd
24                 config.vm.provision "shell", inline: $script
25                 config.vm.provision "ansible_local" do |ansible|
26                         ansible.playbook = "/vagrant/ansible/site.yml"
27                 end
28         end
29 end
```

Figuur 3.18: Inhoud van de Vagrantfile

Het bovenste script gedeelte voert enkele simpele shell commando's uit die Python en Ansible op de Virtuele Machine zal installeren. Lijn 9 tot en met 25 houden zich bezig met enkele configuratieopties in te stellen die nodig zijn voor de virtuele machine. De belangrijkste instellingen bevinden zich op lijn 17 en 23 tot en met 25. Daarbij wordt een data folder samen met zijn inhoud overgezet naar de VM. Dit voorkomt dat er veel tijd dient gespendeerd te worden om de juiste bestanden in de VM te krijgen.

Alsook worden de shell commado's die bovenaan werden gedefinieerd uitgevoerd en tenslotte zal een Ansible playbook ervoor zorgen dat Docker klaar staat voor gebruik op de machine. Aan het einde van de playbook wordt een lange sleutel op de console getoond, ofwel de public SSH-key van het systeem. Dit kan gebruikt worden om vanuit de VM code te uploaden naar github zonder dat de gebruiker zich hiervoor dient te authenticeren.

3.3.3 Voorbereiding: Ansible rollen

In de vorige paragraaf werd er vermeld dat er een Ansible playbook wordt uitgevoerd dat Docker op het systeem zal installeren. Enkele stappen moeten gevolgd worden zodat de virtuele machine aan de slag kan met de Ansible code.

1. In de map waar de vagrantfile staat, maak een nieuwe map genaamd 'data' aan
2. In de map 'data' moet er nog een map aanwezig zijn genaamd 'ansible'.
3. Onder 'ansible' wordt er nog een 'roles' map gemaakt.
4. In de map 'roles' moeten volgende twee Ansible rollen worden geplaatst, beide afkomstig van Github
   - https://github.com/geerlingguy/ansible-role-pip
   - https://github.com/geerlingguy/ansible-role-docker
5. onder de map 'ansible' wordt tenslotte een bestand 'site.yml' gemaakt waar er enkele variabelen aan worden meegegeven. Observeer figuur 3.19

```
- hosts: all
  become: true
  vars:
    docker_users: ['vagrant']
    pip_install_packages:
      - name: docker
      - name: molecule
      - name: ansible
  posts_tasks:
    - name: "Print Public SSH-key"
      debug: msg="SSH KEY: {{lookup('file', '/home/vagrant/.ssh/id_rsa.pub') }}"
  roles:
    - pip
    - docker
```

Figuur 3.19: Inhoud van site.yml. Enkele belangrijke variabelen worden meegegeven

Onder vars zijn er enkele belangrijke zaken die worden meegegeven aan de Ansible rollen die worden uitgevoerd via dit playbook. 'docker_users' is een lijst van gebruikers die Docker kan gebruiken zonder dat het er root rechten voor nodig heeft. Voor Molecule is dit aangeraden. Zoals al geraden kan worden, zijn de andere 3 variabelen de PIP packages die dienen geïnstalleerd te worden.

Wat site.yml concreet doet is voor elke host, waar het in contact mee komt, PIP zal installeren alsook Docker aan de hand van deze variabelen. De 'become' regel zorgt ervoor dat de role op de virtuele machine als root wordt uitgevoerd, wat nodig is als er packages dienen geïnstalleerd te worden.

Als laatste moet nog enkel de role die getest moet worden in de map 'data' geplaatst worden. Deze wordt dan direct overgezet naar de virtuele machine zodat er onmiddelijk mee aan de slag kan gegaan worden.

Eens de voorbereidingen getroffen zijn, kan de virtuele machine worden opgestart met een simpel ‘vagrant up‘ commando. Het kan enkele minuten duren voor de virtuele machine online is en de juiste software is geïnstalleerd.

3.3.4 Een Molecule test uitvoeren via de virtuele machine

Eens de virtuele machine is opgestart kan er via Vagrant gemakkelijk een SSH verbinding worden opgesteld naar de Ubuntu virtuele machine.

```
$ vagrant ssh
```

Op die machine is er reeds een vagrant account aangemaakt dat over sudo rechten bezit zonder gebruik te maken van een wachtwoord. Het is duidelijk dat Vagrant niet echt aangeraden is voor productieomgevingen tenzij dat men de instellingen hiervoor aanpast.

Als alles goed is verlopen zou er onder de *directory* /vagrant de ansible map staan waaruit de Ansible playbook is uitgevoerd om Docker te installeren alsook een andere map met de role die getest moet worden.

Via de terminal kan er naar deze map genavigeerd worden en als de Ansible playbook succesvol was zou een ‘molecule test‘ geen problemen mogen geven.

Opmerking: als er een foutcode wordt gegeven waarbij er wordt verteld dat de Docker daemon niet bereikbaar is, kan dit twee dingen betekenen. Ofwel is de daemon niet actief, wat gecontroleerd kan worden via ‘sudo systemctl status docker‘, ofwel heeft de huidige gebruiker, vagrant dus, geen rechten om met de Docker daemon te werken. Bij beide gevallen zou de VM herstarten het probleem moeten oplossen.

De playbook die eerder werd getoond op figuur 3.19 zal op het einde na de installatie van de software de SSH key op de terminal tonen. Deze key kan men bij de instellingen op Github toevoegen zodat men vanuit de VM code kan uploaden naar de repository. Dit is tenslotte nodig om travis te activeren.

Om deze methode af te ronden wordt een willekeurig bestand uit de VM geopend en zal er een kleine lijn commentaar worden toegevoegd. Op deze manier kan er dan een commit gemaakt worden. Als deze wordt gepusht naar Github zou Travis CI moeten activeren.

In zowat alle gevallen zou het resultaat van de Molecule test zowel op de VM als Travis CI helemaal hetzelfde moeten zijn. Dit is omdat zowel de virtuele machine als de Travis CI omgeving met hetzelfde besturingsysteem werken, namelijk Ubuntu 16.04. Dus als er zich een fout voordoet is de kans zeer groot dat de oorzaak bij een configuratiefout aan Molecule ligt, en niet het systeem zelf.

3.4 Methode 4: Molecule testen met rootless Podman containers

De voorbije methodes hebben allemaal Docker gebruikt om de containers op te zetten via Molecule, maar tegenwoordig is het ook mogelijk om dit via Podman te doen. Zoals eerder werd aangehaald is Podman een container engine tool die door Red Hat zo ontwikkeld is zodat Docker gebruikers zich meteen zouden thuis voelen met de commando syntax van Podman.

Bij Methode 4 wordt methode 1 gerepliceerd maar deze keer wordt er gebruik gemaakt van Podman in plaats van Docker. Ook zal er dan getest worden of dit ook gebruikt kan worden bij Travis CI.

3.4.1 Requirements en uitleg over Podman

Podman is beschikbaar voor de meeste populaire en recente Linux distrubuties en zou alsook werken op Linux subsysteem voor Windows. MacOS gebruikers dienen gebruik te maken van een virtuele machine met een Linux besturingsssysteem. („Podman Installation Guide”, g.d.)

Podman is beschikbaar voor volgende Debian, Ubuntu en Centos Linux distributies:

- Debian 10, Testing of Unstable
- Ubuntu 18.04 of nieuwer
- Centos/ RHEL 7 of nieuwer

Arch Linux en zijn varianten, Fedora en Gentoo hebben geen specifieke besturingssysteem-versie vereisten.

3.4.2 Installatie

Arch Linux, Gentoo, Fedora en Centos gebruikers kunnen heel gemakkelijk Podman installeren aangezien de software aanwezig is via hun respectievelijke package manager. Voor Ubuntu machines is de installatie iets complexer aangezien er een externe repository moet worden toegevoegd. De commando’s die men moet invoeren worden hier kort meegegeven.

```
. /etc/os-release

#Voegt de podman repository, afhankelijk van ubuntu versie, toe aan de apt
#repository lijst
sudo sh -c "echo 'deb https://download.opensuse.org/repositories/devel:/kubic: \
/libcontainers:/stable/ \
xUbuntu_${VERSION_ID}/ /' > \
/etc/apt/sources.list.d/devel:kubic:libcontainers:stable.list"
```

```
#Dit commando haalt de Release key op.
curl -L https://download.opensuse.org/repositories/devel:/\
kubic:/libcontainers:/stable \
xUbuntu_${VERSION_ID}/Release.key | sudo apt-key add -

#Synchroniseer repositories
sudo apt-get update -qq

#Installeer podman
sudo apt-get -qq -y install podman
```

Om Podman te laten samenwerken met Molecule moet de gebruiker die actief is op het systeem de toelating krijgen om *unprivileged* containers te draaien. Dit wordt verwezenlijkt door de gebruiker toegang te geven tot bepaalde namespaces en deze in te stellen via 2 configuratiebestanden, */etc/subuid* en */etc/subgid*.

Met een tesktverwerker kan men deze bestanden aanpassen en de inhoud daarvan moet het volgende formaat volgen.

Inhoudformaat van /etc/subuid en /etc/subgid:

```
GEBRUIKER:UID:GID
```

Een concreet voorbeeld ziet er als volgt uit. Voor containers zal men typisch voor deze UID en GID paren kiezen. Deze kunnen dus voor alle gebruikers gekozen worden.

Inhoud van /etc/subuid en /etc/subgid:

```
robin:100000:65536
```

Als laatste moet er nog enkel via PIP een package worden geïnstalleerd die de communicatie tussen Molecule en Podman zal behandelen.

```
python3 -m pip install -upgrade setuptools
python3 -m pip install "molecule[podman]" podman
```

3.4.3 Molecule met Podman toepassen

Opnieuw wordt er gebruik gemaakt van de role waar er eerder al mee werd gewerkt zoals bij methode 1. Op zich zou het logisch klinken mocht men in de molecule configuratiefile de driver aanpassen van Docker naar Podman maar dan bestaat het risico dat men een vreemde foutcode krijgt zoals deze:

```
  TASK [Discover local Podman images] ***********************************
failed: [localhost] (item=None) => {"censored": "the output has been hidden due to
the fact that 'no_log: true' was specified for this result", "changed": false}
fatal: [localhost]: FAILED! => {"censored": "the output has been hidden due to the
fact that 'no_log: true' was specified for this result", "changed": false}
```

Wat exact dit probleem doet veroorzaken is tot op heden nog steeds onduidelijk. Zowel de documentatie van Molecule als een zoekopdracht via een zoekmachine geeft geen zinvolle uitleg rond dit probleem. Echter is er wel een simpele *workaround* beschikbaar door Molecule opnieuw te initialiseren maar deze keer podman als driver optie mee te geven.

Deze procedure gaat als volgt (enkel van toepassing als molecule al aanwezig is bij de role):

1. Maak een kopie van map met de Ansible role die getest moet worden.
2. Verwijder in de gekopiëerde map de ‘molecule‘ map.
3. Navigeer met een terminal naar de gekopiëerde map.
4. Initialiseer Molecule opnieuw maar deze keer met Podman driver:

```
molecule init scenario -r 'naam van de map met de role' -d podman
```

Dit betekent wel dat het ‘molecule.yml‘ bestand opnieuw aangepast moet worden zodat de correcte containers worden opgesteld. Bovendien vereist Podman een iets andere configuratie in vergelijking met Docker. Als voorbeeld wordt er een Centos 7 en een Fedora 29 container gemaakt. Bekijk figuur 3.20.

Wat is het grootste verschil vergeleken met een Docker configuratiebestand? Het enige wat hier werd aangepast is de ‘volumes‘ optie. Daar is het laatste woord ‘rw‘ vervangen naar ‘ro‘. Even ter herhaling, de volumes optie bepaald een lijst van volumes die de container mag gebruiken. De woorden op het einde beschrijven met welke toestemmingen dit wordt gedaan. ‘rw‘ staat voor ‘Read and Write‘ terwijl ‘ro‘ daarentegen enkel ‘Read only‘ is. („Docker Guides | Use volumes”, 2020)

Het gebruik van ‘Read and write‘ kan regelmatig foutmeldingen veroorzaken doordat de containers die via Molecule worden opgemaakt *rootless* zijn. Met andere woorden, wellicht hebben de containers geen toestemming om write operaties te doen, ook al wordt de ‘rw‘ optie meegegeven.

3.4.4 Een Molecule test uitvoeren

Deze keer zou ‘molecule create‘ (of molecule test, converge, etc.) geen foutmelding mogen geven zoals deze hierboven werd beschreven. Voor de rest is de functionaliteit van Podman bijna identiek aan die van Docker naar testen toe.

Mocht er zich een fout opdoen is het moeilijk te bepalen wat exact het probleem veroorzaakt. Meestal is het een puzzel bij het ‘molecule.yml‘ configuratiebestand om te achterhalen

```
---
dependency:
  name: galaxy
driver:
  name: podman
platforms:
  - name: srv001
    hostname: srv001
    image: "docker.io/geerlingguy/docker-centos7-ansible:latest"
    command: "/usr/sbin/init"
    pre_build_image: true
    volumes:
      - /sys/fs/cgroup:/sys/fs/cgroup:ro
    privileged: true
    tty: True

  - name: ns2
    hostname: ns2
    image: "docker.io/geerlingguy/docker-fedora29-ansible:latest"
    command: "/usr/sbin/init"
    pre_build_image: true
    volumes:
      - /sys/fs/cgroup:/sys/fs/cgroup:ro
    privileged: true
    tty: True
provisioner:
  name: ansible
verifier:
  name: ansible
```

Figuur 3.20: Een molecule.yml configuratiebestand voor Podman

wat er wel werkt en wat niet. Gebruikers die SELinux op hun systeem draaien moeten misschien Docker of Podman nog de toestemming geven om containers te maken.

3.4.5 Molecule testen met Podman via Travis

Een groot nadeel aan Podman is dat men dit echter niet via Travis kan testen. Waarom? Travis CI draait namelijk op een Ubuntu 16.04 systeem terwijl Podman enkel versies nieuwer dan 18.04 ondersteunt. Dit is een belangrijk punt waar men rekening mee moet houden wanneer men een geavanceerde testing infrastructuur wenst op te zetten.

3.5 Methode 5: Molecule testen met rootfull Podman containers

Rootless Podman containers zijn een uitstekende optie naar veiligheid toe aangezien de container in kwestie via een gebruiker wordt aangemaakt die geen administratie (root) rechten heeft. Dit betekent dat de container op zijn beurt ook verminderde privileges heeft in vergelijking met rootfull Podman- of Docker containers. Op zijn beurt zorgt dit dat de container minder schade kan aanrichten op het systeem zelf mochten criminelen proberen inbreken op deze container.

In sommige gevallen zijn rootless Podman containers echter niet voldoende om een bepaalde role te testen. Een goed voorbeeld hiervan zijn Ansible roles waarbij IP-addressen en networking in het algemeen een belangrijke rol spelen. Containers die vanuit een non-root gebruiker worden uitgevoerd krijgen zelfs geen IP-adres, dit betekent dat Ansible rollen die op IP-addressen vertrouwen niet met rootless Podman kunnen getest worden. Hoewel Podman juist gemaakt is om rootless containers te draaien is er uiteraard ook de optie om dit ook via root te doen, rootfull dus.

Om het verschil tussen rootfull en rootless Podman snel even te verduidelijken, rootfull Podman is het equivalent van wat er bij Docker gebeurt. Rootless ligt op een iets lager niveau aangezien het minder privileges heeft dan de vorige twee. Dit is wat Podman interessant maakt bij productie omgevingen omdat daar altijd verlangd wordt naar een veilig systeem.

3.5.1 Installatie

Het gebruik van rootfull containers is niets anders dan een uitbreiding van methode 4. Er wordt ervoor gezorgd dat root ook een entry krijgt in de ‘/etc/subuid/ en ’/etc/subgid‘ bestanden en daarnaast wordt Molecule voor de root gebruiker geïnstalleerd.

Stappenplan

1. Schakel over naar het root account via ‘sudo su‘
2. Navigeer naar de role die met podman dient getest te worden
3. Installeer de python3 en python3-virtualenv packages via de package manager van het systeem
4. Volgende commando’s moeten daarna uitgevoerd worden:

```
#Maakt een virtuele python environment aan
python3 -m virtualenv molecule_ansible

#Activeer de environment
source molecule_ansible/bin/activate

#installeer benodigde packages
pip install ansible testinfra molecule podman ansible-lint \
flake8 molecule[lint] molecule[podman]
```

```
#Heractiveer de environment zodat Molecule herkent wordt
source molecule_ansible/bin/activate
```

3.5.2 Molecule containers inspecteren

Aangezien het doel van deze methode is om het netwerkgedeelte van rootless containers op te lossen is het interessant om te kijken wat Molecule exact doet met de containers die het aanmaakt.

Eerst wordt er bekeken hoe het netwerkgedeelte gebeurt bij rootless containers. Om dit te realiseren moet men even terug schakelen naar een normaal gebruikersaccount mocht men nog aangemeld zijn met het root account. Daarna moet er aan molecule gevraagd worden om de containers te creëren, maar daar verder nog niets mee te doen. ‘molecule create‘ vervult deze taak.

Controleer zeker dat er twee containers werden aangemaakt. De volgende stap is om in te loggen in de container met behulp van het ‘molecule login –host (hostnaam)‘ commando.

Wanneer men in de terminal van de container dan ‘ip a‘ uitvoert is er te zien dat er een zogenaamde ‘tap0‘ netwerk interface werd gemaakt met als ip adres 10.0.2.100. Als men zich echter afmeldt van deze container (via ‘exit‘) en daarna hetzelfde doet voor de tweede container zal er te zien zijn dat het exact hetzelfde ip adres heeft als de eerste container.

Hoe komt dit? Wanneer een rootless gebruiker een container aanmaakt wordt het netwerkgedeelte automatisch ingesteld. De container zelf krijgt geen eigen IP adres want, omdat het geen root rechten heeft, kan het geen beroep doen op de netwerk services. Dit is één van de eerste beperkingen van rootless Podman containers.

Herneemt men de vorige stappen door gebruik te maken van het root account dan zal er te zien zijn dat de containers deze keer wel een eigen IP-adres krijgen, dit omdat er uiteraard wel root rechten zijn. Het IP-adres komt van het netwerk ’10.88.0.0’. Container 1 zou dus 10.88.0.2 krijgen, container 2 10.88.0.3 enzovoort. Het IP-adres is dus weer voorspelbaar net zoals bij Docker.

Er is echter wel nog een groot probleem. Wanneer men deze containers vernietigt en opnieuw aanmaakt, krijgen ze een nieuw IP-adres. 10.88.0.4 en 10.88.0.5 als we vorig voorbeeld hernemen. Het bewijs is te vinden via volgende 4 afbeeldingen vanaf figuur 3.22. De eerste twee afbeeldingen stellen dit probleem voor bij een Centos 8 server die twee Podman containers als root uitvoert terwijl de andere twee dit illusteren bij een Ubuntu server.

```
(molecule_ansible) [root@centos8 testingrole]# molecule login --host ns1
[root@ns1 /]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
3: eth0@if7: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 52:6f:28:42:07:0b brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.88.0.4/16 brd 10.88.255.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::506f:28ff:fe42:70b/64 scope link
       valid_lft forever preferred_lft forever
[root@ns1 /]# exit
exit
(molecule_ansible) [root@centos8 testingrole]# molecule login --host ns2
[root@ns2 /]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
3: eth0@if8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether c6:9b:bd:5f:b1:84 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.88.0.5/16 brd 10.88.255.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::c49b:bdff:fe5f:b184/64 scope link
       valid_lft forever preferred_lft forever
[root@ns2 /]# exit
exit
(molecule_ansible) [root@centos8 testingrole]# molecule destroy
```

Figuur 3.21: IP-addressen van de containers de eerste keer bij de centos 8 server

```
(molecule_ansible) [root@centos8 testingrole]# molecule login --host ns1
[root@ns1 /]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
3: eth0@if9: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 5a:62:86:9b:d4:4c brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.88.0.6/16 brd 10.88.255.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::5862:86ff:fe9b:d44c/64 scope link
       valid_lft forever preferred_lft forever
[root@ns1 /]# exit
exit
(molecule_ansible) [root@centos8 testingrole]# molecule login --host ns2
[root@ns2 /]# ipa
bash: ipa: command not found
[root@ns2 /]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
3: eth0@if10: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether fe:51:5b:64:c9:28 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.88.0.7/16 brd 10.88.255.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::fc51:5bff:fe64:c928/64 scope link
       valid_lft forever preferred_lft forever
```

Figuur 3.22: IP-adressen van de containers bij de tweede keer bij de centos 8 server

```
(molecule_ansible) root@ubuntu:/vagrant/bindPodman# molecule login --host ns1
[root@ns1 /]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
3: eth0@if13: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 7e:f4:7e:5d:34:e5 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.88.0.12/16 brd 10.88.255.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::7cf4:7eff:fe5d:34e5/64 scope link
       valid_lft forever preferred_lft forever
[root@ns1 /]#
(molecule_ansible) root@ubuntu:/vagrant/bindPodman# molecule login --host ns2
[root@ns2 /]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
3: eth0@if14: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 52:6e:6c:dc:94:63 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.88.0.13/16 brd 10.88.255.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::506e:6cff:fedc:9463/64 scope link
       valid_lft forever preferred_lft forever
[root@ns2 /]#
```

Figuur 3.23: IP-addressen van de containers de eerste keer bij de ubuntu server

```
(molecule_ansible) root@ubuntu:/vagrant/bindPodman# molecule login --host ns1
[root@ns1 /]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
3: eth0@if15: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether f6:34:98:29:e5:4d brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.88.0.14/16 brd 10.88.255.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::f434:98ff:fe29:e54d/64 scope link
       valid_lft forever preferred_lft forever
[root@ns1 /]#
(molecule_ansible) root@ubuntu:/vagrant/bindPodman# molecule login --host ns2
[root@ns2 /]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
3: eth0@if16: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether fa:ca:3c:85:85:02 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 10.88.0.15/16 brd 10.88.255.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::f8ca:3cff:fe85:8502/64 scope link
       valid_lft forever preferred_lft forever
[root@ns2 /]#
```

Figuur 3.24: IP-adressen van de containers bij de tweede keer bij de ubuntu server

Dit betekent dat wanneer men een bepaald IP-adres moet instellen voor een Ansible role, dan zou men dit adres telkens moeten aanpassen zodat deze overeenkomt met de container.

Zowel de Podman en de Molecule documentatie, alsook enkele zoekopdrachten via een browser geven geen resultaten rond dit probleem. Dit betekent dat men voor zulke zaken moet overstappen naar Docker waar er veel meer documentatie en troubleshooting gidsen beschikbaar zijn, waaronder voor Docker netwerken.

Laat het duidelijk zijn dat dit eerder een probleem is dat door Molecule of Ansible wordt veroorzaakt of misschien de driver die de communicatie tussen Molecule en Podman behandelt.

3.6 Toepassing van een andere Ansible role

Tot nu toe is er altijd met dezelfde role gewerkt die bewust redelijk simpel is gehouden om de concepten van Molecule gemakkelijker te kunnen overbrengen. Om de kracht van Molecule echt te demonstreren zal er nu een complexere Ansible role worden toegepast met een uitgebreide Travis CI toepassing. Deze role in kwestie is geschreven door Bert Van Vreckem en heeft als verantwoordelijkheid om een open-source implementatie van een DNS-server[1] te installeren op het systeem.

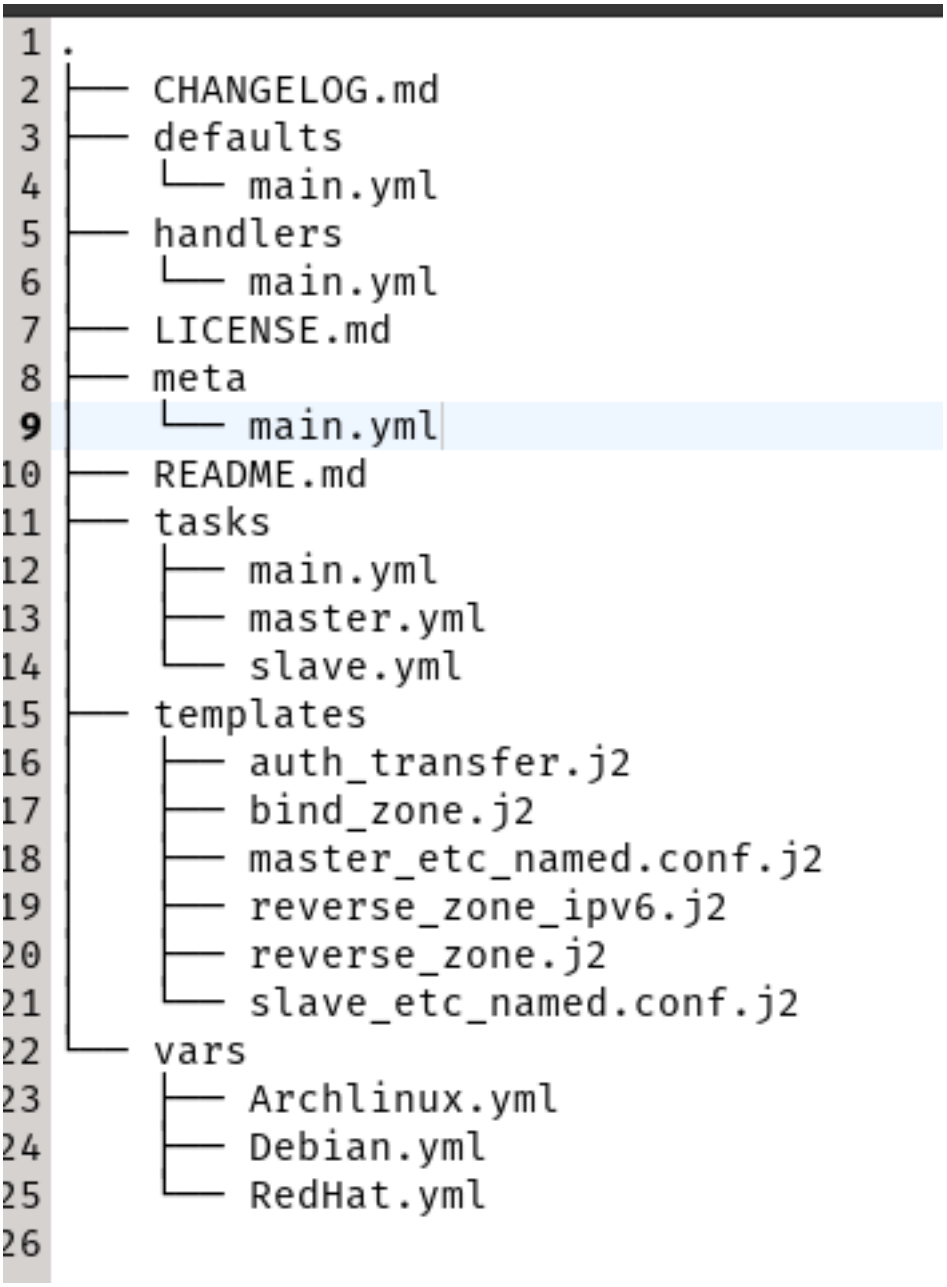
Bert Van Vreckem test deze role op twee manieren: via VM's door gebruik te maken van Vagrant en met Docker containers die via Travis CI worden uitgevoerd. Het verschil ligt echter dat Bert Van Vreckem zelf scripts heeft geschreven die de Docker containers zullen beheren terwijl Molecule deze technische complexiteit op zich neemt.

Er wordt eerst bekeken hoe de workflow van Bert Van Vreckem eruit ziet en daarna wordt een alternatieve methode aangeboden via Molecule. Later zal een vergelijking op basis van enkele criteria zal proberen uitwijzen of Molecule effectief de manuele workflow van Bert Van Vreckem zou kunnen vervangen.

3.6.1 Lokaal testen via Docker scripts

De Ansible BIND role van Bert Van Vreckem kan men vinden via volgende URL: https://www.github.com/bertvv/ansible-role-bind. Als eerst wordt de structuur van deze Ansible role kort besproken zoals te zien is op figuur 3.20. Zo is het snel duidelijk dat deze role veel complexer is dan diegene dat bij de vorige methodes werd toegepast. Zo zijn er een pak meer tasks, templates en vars bestanden en wordt er deze keer alsook gebruik gemaakt van handlers en defaults.

[1] https://www.github.com/bertvv/ansible-role-bind



Figuur 3.25: Structuur van de DNS BIND role

Het is bij het testing gedeelte waar deze role interessant wordt. Zo is er een Github branch voorzien voor testen via Vagrant en een andere branch voor Docker tests. Gezien Molecule via Docker werkt wordt de focus even gelegd op het Docker gedeelte. Dit ziet er als volgt uit:

- dns.bats
- docker-tests.sh
- functional-tests.sh
- test.yml

dns.bats is een script dat commando's uitvoert om te testen of de DNS-server naar behoren

werkt. Het *functional-tests.sh* script helpt hierbij door de benodigde software te installeren om deze zogenaamde BATS testen uit te voeren. *Docker-tests.sh*, zoals de naam al doet vermoeden, maakt de Docker containers aan, roept Ansible op om de role uit te voeren en vernietigt tenslotte de containers. Test.yml bevat de variabelen die moeten gebruikt worden voor de role.

Deze manier van testen kan op twee manieren uitgevoerd worden, lokaal of via Travis CI. Via beide platformen worden ze op dezelfde manier uitgevoerd. Namelijk door de docker-tests.sh uit te voeren waarbij er twee parameters worden meegegeven: het type Linux distributie en een versie daarvan. Voor een Centos 7 container zal de syntax er dus ongeveer zo uitzien:

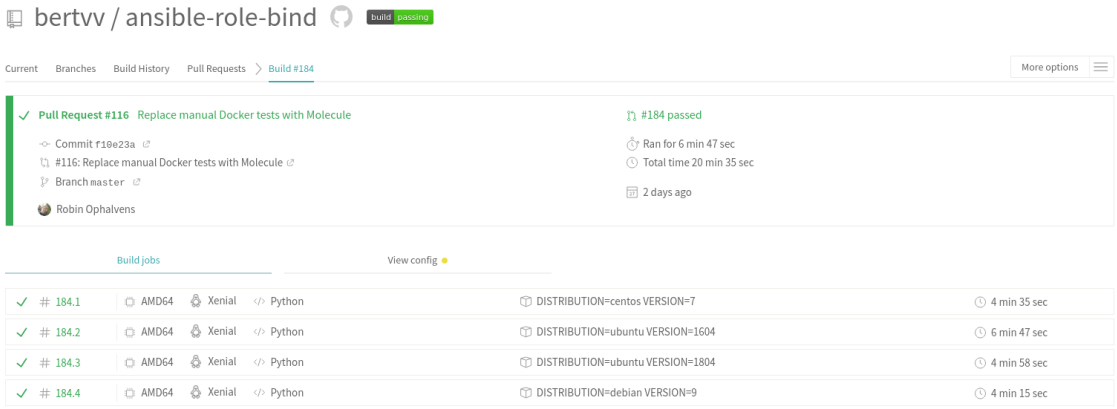
```
DISTRIBUTION=centos VERSION=7 ./docker-tests/docker-tests.sh
```

Waarom wordt dit niet gewoon in de code zelf geplaatst? Dit is omdat deze code speciaal is geschreven om vlot met Travis CI te werken. Het .travis.yml bestand dat te zien is op figuur 3.21 toont dit aan.

Het meest opvallende deel van dit bestand is tussen lijn 3 en 12. Daar wordt een zogenaamde *environment matrix* opgesteld met daarin een verzameling van Linux distributies en hun versies. Wat is het doel hier van? Deze code zorgt ervoor dat Travis CI het ‘script‘ gedeelte meerdere kerel zal uitvoeren. In dit geval vier keer dus, en telkens met een andere type Linux container. Dit zorgt ervoor dat men parallel verschillende testen tegelijkertijd kan uitvoeren zonder hiervoor te hoeven ingrijpen en bovendien oogt het alsook zeer overzichtelijk. Figuur 3.22 bewijst dat Travis CI deze code effectief vier keer uitvoert.

```
sudo: required

env:
matrix:
- DISTRIBUTION: centos
VERSION: 7
- DISTRIBUTION: ubuntu
VERSION: 16.04
- DISTRIBUTION: ubuntu
VERSION: 18.04
- DISTRIBUTION: debian
VERSION: 9

services:
- docker

before_install:
# Install latest Git
- sudo apt-get update
- sudo apt-get install -only-upgrade git
# Allow fetching other branches than master
- git config remote.origin.fetch +refs/heads/*:refs/remotes/origin/*
# Fetch the branch with test code
- git fetch origin docker-tests
- git worktree add docker-tests origin/docker-tests

script:
# Running the test script the first time will set up the master DNS server
# with IP 172.17.0.2. Running it the second time sets up the slave DNS
# server with IP 172.17.0.3.
- ./docker-tests/docker-tests.sh
- ./docker-tests/docker-tests.sh

# Run functional tests on both master and slave server
- SUT_IP=172.17.0.2 ./docker-tests/functional-tests.sh
# Ensure the slave gets the chance to receive zone updates from the master
- sleep 6s
- SUT_IP=172.17.0.3 ./docker-tests/functional-tests.sh

notifications:
webhooks: https://galaxy.ansible.com/api/v1/notifications/
```

Figuur 3.26: Het .travis.yml bestand voor de Ansible BIND role.



Figuur 3.27: Travis CI voert de Docker testen vier keer uit.

3.6.2 Dockertests repliceren met behulp van Molecule

Er wordt nu bekeken hoe men deze Docker testen zou kunnen namaken maar dan met behulp van Molecule. Aangezien Molecule de technische last van Docker containers op zich neemt, kan men de 'docker-tests.sh' in feite schrappen. De andere bestanden kunnen nog steeds van pas komen.

Als eerste moet deze role die op Github te vinden is ge-*cloned* worden naar een lokale PC.

Een stappenplan voor de voorbereiding:

1. Clone de repository:

   ```
   git clone https://github.com/bertvv/ansible-role-bind.git bertvv.bind
   ```

2. Clone alsook de benodigde docker test bestanden:

   ```
   git fetch origin docker-tests
   ```

   ```
   git worktree add docker-tests docker-tests
   ```

3. Verwijder de 'docker-tests.sh' script.

Molecule is nog niet aanwezig in deze role dus het moet nog worden geïnitialiseerd.

```
$ molecule init scenario -r bertvv.bind -d docker
-> Initializing new scenario default...
Initialized scenario in
/home/robin/Projects/bertvv.bind/molecule/default successfully.
```

Nu Molecule is aangemaakt binnen de role moeten de volgende requirements zien vervuld te worden:

- Er moeten twee Docker containers aangemaakt worden
- Deze containers moeten dynamisch kunnen ingesteld worden (Type Linux distributie)
- De BATS testen (of ook wel de acceptatie testen) moeten kunnen uitgevoerd worden op de containers.
- De test code met de variabelen voor de role moet mee opgenomen worden bij Molecule
- De oplossing moet zowel lokaal als via Travis kunnen werken.

Stap per stap wordt er bekeken wat een mogelijke oplossing is voor deze problemen.

Stap 1: Twee Docker containers aanmaken

Zoals nu waarschijnlijk al duidelijk is kunnen de containers worden ingesteld in het configuratiebestand 'molecule.yml'. De oplossing verschilt niet veel met wat er bij de vorige methodes werd gebruikt.

Even een uitleg over de meegegeven opties en de motivaties daarachter:

- **name en hostname**: Het is belangrijk dat de hostnames correct is ingesteld want later zal dit van belang zijn voor de acceptatietesten die zullen uitgevoerd worden.
- **image**: De docker image die gebruikt zal worden
- **command**: Het commando dat uitgevoerd wordt wanneer de container start. Deze is bewust leeg gelaten zodat het de standaard waarde overschrijft die Molecule eraan geeft. Anders kunnen er foutmeldingen ontstaan.
- **volume**: Het type opslag en toestemmingen die de container krijgt.
- **privileged**: Geeft extra privileges aan de container. In test scenario's zal dit de kans op foutmeldingen verminderen. In productieomgevingen is het niet aangeraden om dit te activeren.
- **tty**: Zorgt ervoor dat er gebruik gemaakt kan worden van een TTY omgeving, een terminal binnen de container.
- **environment**: Zowel de Molecule als de Ansible documentatie vertelt weinig over wat het doel is van deze optie, maar het zou kennelijk nodig zijn om systemd gerelateerde zaken te gebruiken.

Stap 2: De container images dynamisch instellen

Bij de oplossing hierboven is de image optie ingevuld met de Centos 7 image gemaakt door Jeff Geerling[1]. Als men echter meerdere images wenst te testen, zoals het geval is bij het .travis.yml bestand van Bert Van Vreckem, zal er een variabele dienen gebruikt te worden om dit te kunnen realiseren. Molecule heeft hiermee gelukkig rekening gehouden („Molecule Configuration Guide - Variable Substitution", 2020)

[1] https://hub.docker.com/u/geerlingguy/

```
dependency:
  name: galaxy
driver:
  name: docker
platforms:
  - name: ns1
    hostname: ns1
    image: "geerlingguy/docker-centos7-ansible:latest"
    command: ""
    volumes:
      - /sys/fs/cgroup:/sys/fs/cgroup:rw
    privileged: true
    tty: True
    environment:
      container: docker
  - name: ns2
    hostname: ns2
    image: "geerlingguy/docker-centos7-ansible:latest"
    command: ""
    volumes:
      - /sys/fs/cgroup:/sys/fs/cgroup:rw
    privileged: true
    tty: True
    environment:
      container: docker
provisioner:
  name: ansible
verifier:
  name: ansible
```

Figuur 3.28: De twee containers die dienen aangemaakt te worden via Molecule

Dit is een stuk uit hun handleiding waar ze schrijven over het gebruik van variabelen:

```
"Both $VARIABLE and ${VARIABLE} syntax are supported.
Extended shell-style features, such as ${VARIABLE-default} and
${VARIABLE:-default} are also supported. Even the default as
another environment variable is supported like ${VARIABLE-$DEFAULT} or
${VARIABLE:-$DEFAULT}. An empty string is returned when both variables
are undefined."
```

Dit stukje hier is zeer interessant. Ontwikkelaars kunnen dus variabelen instellen, maar ook een standaardwaarde mocht de gebruiker vergeten om dit te doen. Als deze werkwijze gebruikt zou worden is het mogelijk om Molecule aan te spreken op ongeveer dezelfde

wijze als docker-tests.sh zoals eerder beschreven, maar dan net iets gemakkelijker weliswaar. Zelfs de standaardwaarde zou men op zijn beurt kunnen vervangen door een andere variabele! Onderstaande code geeft een voorbeeld van hoe dit zou kunnen toegepast worden:

```
#In het molecule.yml  configuratiebestand....
image: "geerlingguy/docker-$(MOLECULE_DISTRO:-centos7)-ansible:latest"

#... die daarna via de terminal dan ingesteld kan worden
MOLECULE_DISTRO=ubuntu1604 molecule test
```

Voordien werd het type image, dat gebruikt dient te worden voor de containers, altijd al op voorhand ingesteld en kon dit nadien niet meer veranderd worden zonder manuele interventie (configuratiebestand weer aanpassen met een tekstverwerker dus). Dankzij de ‘MOLECULE_DISTRO‘ variabele kan men vanuit de terminal makkelijk meegeven via welk besturingssysteem de container moet draaien. Belangrijk om te vermelden bij bovenstaande code is dat slechts een deel nu dynamisch instelbaar is. Als men een besturingssysteem wenst in te stellen via de terminal moet het een type zijn dat beschikbaar is op de Docker hub pagina van Jeff Geerling. Als men de volledige vrijheid wenst van het type image, dus images die niet enkel beperkt zijn tot deze die Jeff Geerling aabiedt, dan zou men de volledige code achter de ‘image‘ optie in het molecule.yml bestand in een variabele moeten stoppen. Ongeveer zo dus:

```
image: "$(MOLECULE_DISTRO:-centos:7)"
```

De ‘centos:7‘ standaardwaarde, deze keer met een dubbele punt symbool ertussen, verwijst naar de officiële Centos 7 image die beschikbaar is via de Docker Hub.

Voor de toepassing van Bert Van Vreckem’s role zal er nog steeds gebruikt worden van Jeff Geerling’s images, aangezien deze gebouwd zijn om met Ansible te werken en daarbovenop is SystemD hier ook bij geïncludeerd. Om dit te realiseren moet het molecule configuratiebestand enkele kleine aanpassingen ondergaan.

Naast de MOLECULE_DISTRO variabele heeft ook de ‘command‘ optie een variabele gekregen. Vele online handleidingen gaan hierbij bijvoorbeeld ‘/usr/sbin/init‘ meegeven, maar dit zal problemen geven bij Debian en Ubuntu images. Vandaar dat de gebruiker de optie krijgt om deze in te stellen binnen de terminal. In de meeste gevallen zal de lege aanhalingstekens optie perfect functionerend zijn.

```
dependency:
  name: galaxy
driver:
  name: docker
platforms:
  - name: ns1
    hostname: ns1
    image: "geerlingguy/docker-$(MOLECULE_DISTRO:-centos7)-ansible:latest"
    command: $(MOLECULE_COMMAND:-"")
    volumes:
      - /sys/fs/cgroup:/sys/fs/cgroup:rw
    privileged: true
    tty: True
    environment:
    container: docker
  - name: ns2
    hostname: ns2
    image: "geerlingguy/docker-$(MOLECULE_DISTRO:-centos7)-ansible:latest"
    command: $(MOLECULE_COMMAND:-"")
    volumes:
      - /sys/fs/cgroup:/sys/fs/cgroup:rw
    privileged: true
    tty: True
    environment:
    container: docker
provisioner:
  name: ansible
verifier:
  name: ansible
```

Figuur 3.29: Een licht aangepaste versie van molecule.yml

Stap 3: De acceptatie testbestanden op de juiste plaats zetten en deze uitvoeren

Bert Van Vreckem heeft acceptatietesten geschreven die vanuit de host machine (het apparaat dat de Docker containers aanmaakt) enkele testen zal uitvoeren op de containers om te achterhalen of alle gevraagde functionaliteiten operationeel zijn. Zulke testen worden ook wel ‘black-box testen‘ genoemd.

Als eerste dient er onder ‘molecule/default‘ een nieuwe map aangemaakt te worden met de naam files. In deze map komen de bestanden ‘functional-tests.sh‘ en ‘dns.bats‘ te staan.

Nu deze bestanden op de juiste plek staan moet men Ansible vertellen om deze effectief ook uit te voeren. Via Molecule zal dit doormiddel van de verify.yml playbook gebeuren. Zoals terug bij Methode 1 werd beschreven is de verify playbook bedoeld om via Ansible of TestInfra de toestand van de containers te testen. Met andere woorden, in dit geval kan de verify playbook gebruikt worden om de black-box testen uit te voeren op de containers.

Het gebruik van Verify heeft bovendien nog een interessant voordeel. Als men snel een black-box test wil uitvoeren kan men dit gemakkelijk via het ‘molecule verify‘ commando doen. Stel dat de testen in plaats van bij Verify, bij de Converge fase worden aangeroepen, dan zou men, afhankelijk van hoe groot en complex de role is, enkele minuten verliezen doordat de role telkens weer opnieuw wordt uitgevoerd op de containers.

Figuur 3.30 toont voor deze situatie een mogelijke oplossing.

```
1  ---
2  - name: Verify
3    hosts: all
4    tasks:
5      # We run the BATS tests from the localhost,
6      #since they are black box tests
7      - name: "Run BATS tests for {{ ansible_hostname }}"
8        shell: >
9          SUT_IP={{ ansible_default_ipv4.address }}
10         bats {{ playbook_dir }}/files/dns.bats
11       delegate_to: localhost
12       changed_when: false
```

Figuur 3.30: Het verify.yml configuratiebestand waar er opnieuw enkele variabelen worden gebruikt.

Ansible verzamelt bij elk systeem waar het op zal uitgevoerd worden enkele ‘feitjes‘ of ‘weetjes‘ over het systeem. Of zoals Ansible ze noemt: ‘Facts‘. Dankzij deze Facts kan men variabelen gebruiken en deze toepassen in playbooks. Bij lijn 7 bijvoorbeeld op figuur 3.30 wordt de naam van de container opgehaald en bij lijn 9 het IP-adres van de container. Dit is namelijk nodig wanneer men de black-box testen wenst uit te voeren op de container.

De acceptatietesten van Bert Van Vreckem vragen aan de gebruiker om bij het script een IP-adres mee te geven waarop de test dient uitgevoerd te worden. Ansible Facts automatiseert dit en bovendien zorgt het gebruik van deze variabelen ervoor dat men geen meerdere tasks dient te schrijven om de black-box testen op andere containers uit te voeren. Op lijn 9 is te zien hoe het IP-adres automatisch zal ingevuld worden.

Het ‘>‘ teken na de ‘shell‘ optie bij lijn 8 zorgt ervoor dat Ansible lijn 9 en 10 als één lijn zal samenvoegen en zo uitvoeren. Dit wordt voornamelijk gedaan om lange lijnen in het bestand of op de pagina te vermijden.

De ‘delegate_to‘ optie vertelt Ansible dat deze taak op het host systeem moet uitgevoerd worden, en niet op de containers.

Stap 4: de testvariabelen aan Molecule geven

Bert Van Vreckem heeft voor zijn role een playbook gemaakt met daarbij een verzameling van variabelen zodat hij op deze manier kan testen of zijn role naar behoren werkt.

Dit playbook kan men via Molecule perfect opnemen door de variabelen toe te voegen aan het ‘molecule/default/converge.yml‘ bestand. Naast deze variabelen worden hier alsook de tasks die de testbestanden overzetten naar de containers toegevoegd.

```
---
- name: Converge
  hosts: all
  vars:
    bind_zone_dir: /var/local/named-zones
    bind_zone_file_mode: '0660'
    bind_allow_query:
      - any
    bind_listen_ipv4:
      - any
    bind_listen_ipv6:
      - any
    bind_acls:
      - name: acl1
        match_list:
          - 172.17.0.0/16
    bind_forwarders:
      - '8.8.8.8'
      - '8.8.4.4'
    bind_recursion: true
    bind_query_log: 'data/query.log'
    bind_check_names: 'master ignore'
    bind_zone_master_server_ip: 172.17.0.2
    bind_zone_minimum_ttl: "2D"
    bind_zone_ttl: "2W"
    bind_zone_time_to_refresh: "2D"
    bind_zone_time_to_retry: "2H"
    bind_zone_time_to_expire: "2W"
    bind_zone_domains:
      - name: 'example.com'
        networks:
          - '192.0.2'
        ipv6_networks:
          - '2001:db9::/48'
        name_servers:
          - ns1.acme-inc.com.
          - ns2.acme-inc.com.
        hostmaster_email: admin
        hosts:
          - name: srv001
            ip: 192.0.2.1
            ipv6: '2001:db9::1'
            aliases:
              - www
          - name: srv002
            ip: 192.0.2.2
            ipv6: '2001:db9::2'
          - name: mail001
            ip: 192.0.2.10
            ipv6: '2001:db9::3'
```

Figuur 3.31: De congerge.yml playbook met de testvariabelen, deel 1

```
    mail_servers:
      - name: mail001
        preference: 10
  - name: 'acme-inc.com'
    networks:
      - '172.17'
      - '10'
    ipv6_networks:
      - '2001:db8::/48'
    name_servers:
      - ns1
      - ns2
    hosts:
      - name: ns1
        ip: 172.17.0.2
      - name: ns2
        ip: 172.17.0.3
      - name: srv001
        ip: 172.17.1.1
        ipv6: 2001:db8::1
        aliases:
          - www
      - name: srv002
        ip: 172.17.1.2
        ipv6: 2001:db8::2
        aliases:
          - mysql
      - name: mail001
        ip: 172.17.2.1
        ipv6: 2001:db8::d:1
        aliases:
          - smtp
          - mail-in
      - name: mail002
        ip: 172.17.2.2
        ipv6: 2001:db8::d:2
      - name: mail003
        ip: 172.17.2.3
        ipv6: 2001:db8::d:3
        aliases:
          - imap
          - mail-out
      - name: srv010
        ip: 10.0.0.10
      - name: srv011
        ip: 10.0.0.11
      - name: srv012
        ip: 10.0.0.12
```

Figuur 3.32: Converge.yml, deel 2

```
        mail_servers:
          - name: mail001
            preference: 10
          - name: mail002
            preference: 20
        services:
          - name: _ldap._tcp
            weight: 100
            port: 88
            target: srv010
        text:
          - name: _kerberos
            text: KERBEROS.ACME-INC.COM
          - name: '@'
            text:
              - 'some text'
              - 'more text'
roles:
  - role: bertvv.bind
```

Figuur 3.33: Converge.yml, deel 3

Stap 5: Molecule zowel lokaal als via Travis CI laten werken

De code langs de kant van Molecule is nu in orde, het travis.yml bestand dient nog enkel aangepast te worden zodat het gebruik maakt van de environment matrix en Molecule zelf. Deze code is te zien op figuur 3.29.

Lijn 1 en 5 zijn standaard. Er wordt beschreven dat python gebruikt zal worden en dat sudo rechten noodzakelijk zijn. Vanaf lijn 7 worden de variabelen ingesteld in matrix vorm die Travis CI dan één voor één zal uitvoeren.

De nieuwkomers bij dit bestand is de globale 'ROLE_NAME' variabele en alsook het 'before_script' gedeelte. Bert Van Vreckem's Ansible role staat op Github bekend als 'ansible-role-bind' terwijl dat dit op Ansible Galaxy anders is, namelijk 'bertvv.bind'. Het is aangeraden om de Ansible role zoveel mogelijk te ontwikkelen met Ansible Galaxy in gedachten dus het before_script gedeelte zal de folder van de Ansible role even van 'ansible-role-bind' naar 'bertvv.bind' hernoemen. Op deze manier kan de Ansible role op het Ansible Galaxy platform een betere score krijgen naar structuur van de code toe. Uniformiteit is bovendien ook overzichtelijker.

Het install gedeelte is vanaf nu redelijk vanzelfsprekend. De juiste python packages worden via PIP geïnstalleerd zodat Molecule aan de slag kan met de Docker containers. 'netaddr' is een specifieke package die nodig is om de role van Bert Van Vreckem in te stellen.

Opnieuw wordt de versie van Molecule en Ansible getoond voor mocht debuggen nodig zijn en het Ansible configuratiebestand wordt snel ingesteld.

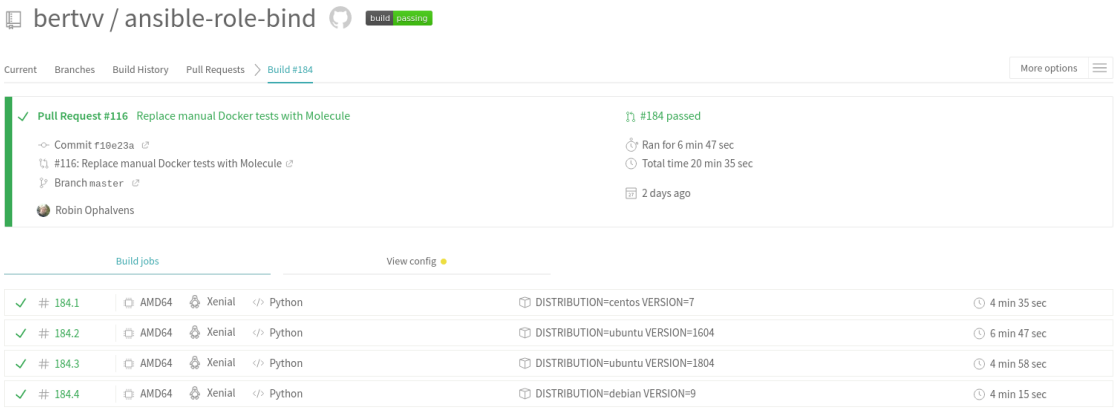
Wat wel opvalt is dat het script zelf slechts uit één lijn bestaat. Inderdaad, de manier waarop Molecule werkt en is ingesteld laat toe om de containers op te stellen, uit te voeren met Ansible en deze te testen met de BATS acceptatie testen, allemaal met één commando!

Bij de bespreking van de resultaten bij het volgende deel wordt er in detail ingegaan op wat de gelijkenissen en verschillen zijn tussen de manuele Docker testen en het gebruik van Molecule.

Wanneer men deze code pusht naar Github zal men zien dat Travis CI wordt geactiveerd, opnieuw worden er vier zogenaamde jobs uitgevoerd, bij elke job wordt een andere variabele gebruikt zoals beschreven in de environment matrix.

```
---
language: python

# Use the new container infrastructure
sudo: required

env:
  global:
    - ROLE_NAME: bind
matrix:
  - MOLECULE_DISTRO: centos7
  - MOLECULE_DISTRO: debian9
  - MOLECULE_DISTRO: ubuntu1604

# Enable docker support
services:
  - docker

install:
  - sudo apt-get update
  - sudo apt-get install bats curl dnsutils
  # Install dependencies for Molecule test
  - python3 -m pip install molecule yamllint ansible-lint docker netaddr
  # Check ansible and molecule version
  - ansible -version
  - molecule -version

  # Create ansible.cfg with correct roles_path
  - printf '[defaults]\nroles_path=../' >ansible.cfg

before_script:
  # Renames ansible-role-bind to bertvv.bind to make it match with Ansible
  # Galaxy
  - cd ../
  - mv ansible-role-$ROLE_NAME bertvv.$ROLE_NAME
  - cd bertvv.$ROLE_NAME

script:
  # Run molecule test
  - molecule test

notifications:
  webhooks: https://galaxy.ansible.com/api/v1/notifications/
```

Figuur 3.34: De inhoud van het nieuwe .travis.yml configuratiebestand



Figuur 3.35: Opnieuw is er te zien dat er viers jobs werden uitgevoerd. Deze zijn allemaal geslaagd

3.6.3 Extra: manueel de IP-addressen instellen van de containers

Tot nu toe werd er vertrouwd op Docker en Molecule om de IP-addressen in te stellen van de containers. De eerste container die wordt aangemaakt krijgt altijd als IP 172.17.0.2, de tweede 172.17.0.3, enzovoort. Aangezien het altijd via dit patroon gebeurt, kan men dit toepassen bij de Ansible roles. De BIND role van Bert Van Vreckem vraagt de ontwikkelaar om altijd een zogenaamde ‘master_ip‘ IP in te stellen zodat de role een onderscheid kan maken welke server/container de Master rol krijgt en welke de Slave rol. Het is echter ook mogelijk om zelf een netwerk aan te maken en de IP-addressen in te stellen, maar dit vereist toch enige moeite naar configuratie toe. Het doel van dit onderzoek is om de workflow te verbeteren, en niet om het complexer te maken, maar voor de ontwikkelaars die *per sé* een eigen netwerk wensen in te stellen, wordt dit deel snel even toegelicht.

Om dit te realiseren dienen er twee stappen uitgevoerd te worden. Eerst moet er een docker netwerk aangemaakt worden en daarop zal de ‘molecule.yml‘ aangepast worden zodat de containers deelnemen aan dit netwerk met het gewenste IP-adres, en niet het ’default’ docker netwerk.

Stap 1: Docker netwerk aanmaken

Een nieuw docker netwerk kan gemaakt worden met het ‘docker network create‘ commando. Een voorbeeld wordt hier getoond:

```
docker network create \
-driver=bridge \
-subnet=172.28.0.0/16 \
-ip-range=172.28.5.0/24 \
-gateway=172.28.5.254 \
br0
```

Er wordt dus een bridge netwerk aangemaakt met de naam ‘br0‘. Dit netwerk bevindt zich in het subnet 172.28.0.0 en de containers kunnen een IP-adres krijgen tussen 172.28.5.0 en 172.28.5.254.

Stap 2: Containers laten deelnemen aan br0 netwerk

De tweede stap bestaat eruit om het molecule.yml configuratiebestand aan te passen zodat de containers zullen deelnemen aan dit netwerk. Om dit te doen moet men bij elke container een ‘networks‘ optie toevoegen met daaronder de naam van het netwerk en het gewenste IP-adres. Optioneel kan men ook een alias aan de container geven en specifiëren met welke andere containers het ‘gelinkt‘ moet zijn. Bekijk figuur 3.36 en lijn 14 tot en met 16 waar dit gerealiseerd is voor de eerste container.

```yaml
dependency:
  name: galaxy
driver:
  name: docker
platforms:
  - name: ns1
    hostname: ns1
    image: "geerlingguy/docker-centos7-ansible:latest"
    command: "/usr/sbin/init"
    volumes:
      - /sys/fs/cgroup:/sys/fs/cgroup:rw
    privileged: true
    tty: True
    networks:
      - name: br0
        ipv4_address: '172.28.5.4'
    environment:
      container: docker
  - name: ns2
    hostname: ns2
    image: "geerlingguy/docker-centos7-ansible:latest"
    command: "/usr/sbin/init"
    volumes:
      - /sys/fs/cgroup:/sys/fs/cgroup:rw
    privileged: true
    tty: True
    networks:
      - name: br0
        ipv4_address: '172.28.5.5'
    environment:
      container: docker
provisioner:
  name: ansible
verifier:
  name: ansible
```

Figuur 3.36: Molecule.yml, nu met aangepast netwerk gedeelte

Stap 3: aanpassingen testen

Na een ‘molecule create‘ commando uit te voeren kan met inloggen op de container via het ‘molecule login‘ commando. Eens er aangemeld is op de container kan er via ‘ip a‘ bevestigd worden dat de container effectief het gespecifieerde IP-adres heeft meegekregen. Bekijk vooral lijn 11 en 25 bij figuur 3.37.

```
1  (bachelor) robin@pop-os:~/Projects/ansible-role-bind$ molecule login -host ns1
2  [root@ns1 /]# ip a
3          1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state
4          UNKNOWN group default qlen 1000
5          link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
6          inet 127.0.0.1/8 scope host lo
7          valid_lft forever preferred_lft forever
8          22: eth0@if23: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500
9          qdisc noqueue state UP group default
10         link/ether 02:42:ac:12:00:04 brd ff:ff:ff:ff:ff:ff link-netnsid 0
11         inet 172.28.5.4/24 brd 172.18.255.255 scope global eth0
12         valid_lft forever preferred_lft forever
13 [root@ns1 /]# exit
14 exit
15 (bachelor) robin@pop-os:~/Projects/ansible-role-bind$ molecule login -host ns2
16 [root@ns2 /]# ip a
17         1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state
18         UNKNOWN group default qlen 1000
19         link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
20         inet 127.0.0.1/8 scope host lo
21         valid_lft forever preferred_lft forever
22         24: eth0@if25: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500
23         qdisc noqueue state UP group default
24         link/ether 02:42:ac:12:00:05 brd ff:ff:ff:ff:ff:ff link-netnsid 0
25         inet 172.18.0.5/16 brd 172.18.255.255 scope global eth0
26         valid_lft forever preferred_lft forever
27 [root@ns2 /]# exit
```

Figuur 3.37: De containers hebben inderdaad een nieuw IP-adres gekregen

Potentiële problemen met deze werkwijze

Hoewel deze methode toelaat om zelf IP-addressen in te stellen voor de containers kan het resultaat heel inconsistent zijn. Zo werd dit enkele keren getest op het Pop!_OS hostsysteem, een Centos 8 VM en een Ubuntu 16.04 VM. 1 of 2 keer slaagde Molecule en Docker erin om dit te realiseren maar andere keren werd deze foutmelding meegegeven:

```
failed: [localhost] (item=None) => {
"attempts": 1,
"censored": "the output has been hidden due to the fact that
'no_log: true' was specified for this result",
"changed": false
}
fatal: [localhost]: FAILED! => {
"censored": "the output has been hidden due to the fact that
'no_log: true' was specified for this result",
"changed": false
}

PLAY RECAP *********************************************************************
localhost                  : ok=7    changed=1    unreachable=0    failed=1
```

Figuur 3.38: Een redelijk onbruikbare foutmelding van Ansible

Hoe dit opgelost kan worden is op dit moment nog niet gevonden. Dat is iets wat een update vanuit Molecule of Ansible hopelijk kan oplossen en in een ideaal geval wordt er hier een onderzoek naar uitgevoerd.

3.7 Een tweede toepassing van een andere Ansible role

Er wordt een tweede toepassing gedemonstreerd, maar deze keer met een andere, en iets specifiekere role. De Ansible role in kwestie die getest zal worden via Molecule is Bert Van Vreckem's rh-base, die algemene administratieve zaken kan uitvoeren op Red Hat gebaseerde besturingssystemen. Een belangrijke factor waarom deze role werd gekozen om er een tweede test mee uit te voeren is omdat het alsook SELinux zal instellen. Zoals al eerder bij de Stand van zaken werd aangehaald is een SELinux een applicatie die bovenop de standaard Linux toestemmingen een extra laag aan controle en bescherming toevoegt.

Dit brengt een interessant probleem met zich mee. Om even kort te herhalen, containers zijn geïsoleerde processen die naast het besturingssysteem bovenop de kernel van de machine draaien. Deze isolatie wordt gerealiseerd door zogenaamde namespaces en cgroups. Het namespaces gedeelte is hier belangrijk. Een namespace laat namelijk toe om de middelen (processen, netwerk, gebruikers, opslag) van het systeem virtueel te verdelen in partities. („Linux Programmer's Manual - Namespaces", 2020). Docker kan dus aan de kernel vragen om een namespace aan te maken voor een container waarbij het proces dan een limiet heeft van welke middelen het kan gebruiken die het systeem aanbiedt. Waarom is dit belangrijk en wat heeft dit te maken met SELinux?

SELinux is namelijk niet namespaced wat betekent dat SELinux policies niet kunnen opereren binnen de containers zelf, alsof het een besturingssysteem op zich zou zijn. Dit betekent dat SELinux fundamenteel incompatibel zou zijn om in Docker containers te draaien dus dit deel gaat uitzoeken wat de mogelijke impact hiervan zal zijn op de werking van Molecule.

3.7.1 Voorbereiding

De rh-base role kan men vinden via deze URL: https://www.github.com/bertvv/ansible-role-rh-base.

Om deze stappen te herproduceren dient de role gedownload of *gecloned* te worden zodat hiermee gewerkt kan worden. RH-base is bedoeld om uit te voeren op een virtuele machine of een server zelf, dus er zullen enkele zaken moeten aangepast worden aan deze role zodat deze werkt via een container. Bij containers kan het gebeuren dat enkele software tools ontbreken die standaard geïnstalleerd op een volwaardige besturingssysteem.

Stap 1: Molecule initialiseren en instellen

Molecule is nog niet aanwezig in deze role dus het moet nog worden geïnitialiseerd.

```
$ molecule init scenario -r ansible-role-rh-base -d docker
-> Initializing new scenario default...
Initialized scenario in
/home/robin/Projects/ansible-role-rh-base/molecule/default successfully.
```

Eerst en vooral wordt de ‘molecule.yml‘ file ingesteld en wordt er gekozen om met een Centos 7 container te werken. Figuur 3.39 toont deze oplossing. Hoewel eerder werd vermeld dat SELinux incompatibel is met Docker containers wordt de functionaliteit zo veel mogelijk nagebootst. Daarom wordt een optie ‘capabilities‘ gegeven met daarbij ‘SYS_ADMIN‘. Opnieuw is dit enkel voor testing doeleinden. Het is ten sterkste afgeraden om deze configuratie in een productieomgeving te gebruiken.

```
--
dependency:
  name: galaxy
driver:
  name: docker
platforms:
  - name: Centos7
    image: "geerlingguy/docker-centos7-ansible:latest"
    command: "/usr/sbin/init"
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock:rw
      - /sys/fs/cgroup:/sys/fs/cgroup:ro
    privileged: true
    capabilities:
      - SYS_ADMIN
    pre_build_image: true
    tty: True
    environment:
      container: docker
provisioner:
  name: ansible
verifier:
  name: ansible
```

Figuur 3.39: Molecule.yml voor de centos 7 container.

Stap 2: ontbrekende software tools toevoegen

De volgende stap is om ‘vars/RedHat.yml‘ aan te passen zodat de ontbrekende software tools worden geïnstalleerd.

```
# roles/rh-base/vars/RedHat.yml
---
rhbase_systemd_services:
  - systemd-journald
  - systemd-tmpfiles-setup-dev
  - systemd-tmpfiles-setup

rhbase_dependencies:
  - libselinux-python
  - libsemanage-python
  - firewalld
  - openssh-server
  - selinux-policy.noarch

rhbase_package_manager: yum
rhbase_package_manager_configuration: /etc/yum.conf

rhbase_updates_packages:
  - yum-cron
rhbase_updates_service: yum-cron
rhbase_updates_config:
  - yum-cron.conf
  - yum-cron-hourly.conf
```

De voornaamste aanpassing hierbij is dat er twee packages zijn toegevoegd aan ‘rhbase_dependencies‘. Namelijk ‘openssh-server’ en ‘selinux-policy.noarch’.

Stap 3: Test bestanden klaar zetten

Opnieuw is er een acceptatietest voorzien die zal controleren of de voornaamste features zijn ingesteld door de role. De broncode hiervan is te vinden aan het einde van dit hoofdstuk en alle credits behoren nogmaals tot Bert Van Vreckem. In de ‘molecule/default‘ folder wordt een files map aangemaakt waar deze twee bestanden in terecht komen.

Stap 4: Converge.yml en verify.yml instellen

Net zoals bij de eerste toepassing van een andere Ansible role zullen in converge.yml de testvariabelen vermeld worden alsook enkele tasks die de testbestanden overzetten naar de container. Figuur 3.40 toont het resultaat. Deze variabelen zullen RH-base de opdracht geven om een gebruikersaccount aan te maken met de naam ‘robin’ en zorgt ervoor dat de

SSH-sleutel wordt toegepast zodat hij zich direct kan aanmelden zonder een wachtwoord te hoeven ingeven. De sleutel zelf is om veiligheidsredenen uiteraard hier niet te vinden. Alsook wordt de firewall geactiveerd en zullen er enkele packages worden geïnstalleerd en daarbovenop wordt SELinux geactiveerd.

```
---
- name: Converge
  hosts: all
  vars:
    rhbase_users:
      - name: robin
        comment: 'Robin Ophalvens'
        groups:
          - wheel
        password: "$6$ScOBREutjXejyFLt$leFt6qP.d.wtLyOOhkbeqaEgWsJe2Ik
        DPtEh.j5jO/0zmo1F9AjXw.9yP.FyQBVv6i.8FZNVmReiqyU29jE3g1"
    rhbase_install_packages: ['epel-release','openssh-server','firewalld',
    'bash-completion','bind-utils','git','nano','tree','vim','wget','traceroute']
    rhbase_start_services: ['firewalld']
    rhbase_selinux_state: 'enforcing'
    rhbase_ssh_key: 'wordt hier even niet getoond'
    rhbase_ssh_user: 'robin'
    rhbase_dynamic_motd: true
  tasks:
    - name: "Include ansible-role-rh-base"
      include_role:
        name: "ansible-role-rh-base"
  post_tasks:
    - name: "Transfer testscript to container"
      copy:
        src: runbats.sh
        dest: /runbats.sh
        owner: root
        group: root
        mode: 773

    - name: "transfer bats tests to container"
      copy:
        src: common.bats
        dest: /common.bats
        owner: root
        group: root
        mode: 773
    - name: "transfer bats tests to /tmp"
      copy:
        src: common.bats
        dest: /tmp/common.bats
        owner: root
        group: root
        mode: 773
```

Figuur 3.40: Inhoud van converge.yml. Common.bats wordt deze keer ook in de /tmp map gestopt om fouten te voorkomen.

Het verify.yml configuratiebestand is deze keer zeer eenvoudig. De verify playbook zal er voor zorgen dat de acceptatie testen worden uitgevoerd.

```
- name: Verify
  become: true
  hosts: all
  tasks:
    - name: "Run test bats against containers"
      shell: /runbats.sh
      register: shell_result

    - name: "Display bats results of BATS"
      debug:
        var: shell_result.stdout_lines
```

Stap 5: Molecule activeren en resultaten observeren

Nu alles gereed staat kan er snel getest worden met Molecule via het ‘molecule test’ commando. In normale omstandigheden zullen er twee merkwaardige resultaten te zien zijn.

Eerst en vooral zal RH-base de waarschuwing geven dat de ’machine’ heropgestart moet worden zodat SELinux kan overschakelen naar de ‘enforcing’ modus, waarbij het effectief zijn controle zal uitvoeren naar toestemmingen toe.

```
TASK [ansible-role-rh-base : Security |
Make sure SELinux has the desired state (enforcing)] ***
[WARNING]: Reboot is required to set SELinux state to 'enforcing'
    ok: [Centos7]
```

Een container is niet een besturingssysteem dus het is onmogelijk om deze zomaar te herstarten. Echter is het wel interessant dat deze waarschuwing geen effectieve foutmelding is. Moest dit het geval zijn dan zou Ansible het proces stoppen en niet verder de taken uitvoeren.

Wanneer de converge fase is afgerond zal Molecule de verify playbook aanroepen die de acceptatietesten zal uitvoeren. Opnieuw slaagt alles behalve de controle naar de SELinux status.

```
[root@Centos7 /]# ./runbats.sh
Running test /common.bats
 FAILED: SELinux should be set to 'Enforcing'
   (in test file common.bats, line 7)
     `[ 'Enforcing' = $(getenforce) ]' failed
 PASSED: Firewall should be enabled and running
 PASSED: EPEL repository should be available
 PASSED: Bash-completion should have been installed
 PASSED: bind-utils should have been installed
 PASSED: Git should have been installed
 PASSED: Nano should have been installed
 PASSED: Tree should have been installed
 PASSED: Vim-enhanced should have been installed
 PASSED: Wget should have been installed
 PASSED: Admin user robin should exist
 PASSED: An SSH key should have been installed for robin
 SKIPPED: Custom /etc/motd should have been installed (skipped)

13 tests, 1 failure, 1 skipped
```

Dit betekent dat de role wel effectief de incompatibele zaken kan uitvoeren op de container maar dat de controle en effectieve functionaliteit hiervan beperkt is.

Opmerking: bij de eerste toepassing van een andere Ansible role werden de acceptatie testen vanuit het host systeem uitgevoerd terwijl het hier rechtstreeks in de container plaats vindt. Dit is omdat deze testen niet vanop afstand kunnen uitgevoerd worden en enkel op het systeem zelf. Dit verklaart ook waarom bij ‘converge.yml‘ enkele tasks werden toegevoegd die de testbestanden zal overzetten naar de container. Vele stukken code zijn hier opzettelijk *hard coded*, tegen *best practices* in, om opnieuw de essentie van deze methode makkelijker te kunnen aantonen. Namelijk of Ansible bepaalde functionaliteiten, zoals SELinux, kan testen op containers terwijl dit in theorie niet mogelijk zou mogen zijn.

3.7.2 Bijlagen voor deze methode

Bijlage 1: runbats.sh acceptatietest

```
#! /usr/bin/bash
#
# Author:   Bert Van Vreckem <bert.vanvreckem@gmail.com>
#
# Run BATS test files in the current directory, and the ones in the subdirectory
# matching the host name.
#
# The script installs BATS if needed. It's best to put ${bats_install_dir} in
# your .gitignore.

set -o errexit  # abort on nonzero exitstatus
set -o nounset  # abort on unbound variable

#{{{ Variables

test_dir="$( cd "$( dirname "${BASH_SOURCE[0]}" )" && pwd )"

bats_version="1.1.0"
bats_archive="v${bats_version}.tar.gz"
bats_url="https://github.com/bats-core/bats-core/archive/${bats_archive}"
bats_install_dir="/usr/local"
bats="${bats_install_dir}/bin/bats"

test_file_pattern="*.bats"

# color definitions
blue='\e[0;34m'
yellow='\e[0;33m'
reset='\e[0m'

#}}}
#{{{ Functions

# Predicate that checks whether BATS is installed
is_bats_installed() {
test -f "${bats}"
}

# Usage: install_bats_if_needed
install_bats_if_needed() {
if ! is_bats_installed; then
echo -e "${blue}Installing BATS${reset}"
```

```
cd /tmp
wget "${bats_url}"
tar xf "${bats_archive}"
(
cd "./bats-core-${bats_version}/"
./install.sh "${bats_install_dir}"
)
rm "${bats_archive}"
rm -r "/tmp/bats-core-${bats_version}"
fi
}

# find_tests DIR [MAX_DEPTH]
find_tests() {
local max_depth=""
if [ "$#" -eq "2" ]; then
max_depth="-maxdepth $2"
fi

local tests

tests=$(find "$1" ${max_depth} -type f -name "${test_file_pattern}" -printf '%p\n'

echo "${tests}"
}
#}}}
# Script proper

install_bats_if_needed

# List all test cases (i.e. files in the test dir matching the test file
# pattern)

# Tests to be run on all hosts
global_tests=$(find_tests "${test_dir}" 1)

# Tests for individual hosts
host_tests=$(find_tests "${test_dir}/${HOSTNAME}")

# Loop over test files
for test_suite in ${global_tests} ${host_tests}; do
echo -e "${blue}Running test ${yellow}${test_suite}${reset}"
${bats} "${test_suite}"
done
```

Bijlage 2: common.bats

```
#! /usr/bin/env bats
#
# Acceptance test script for srv
admin_user=robin

@test "SELinux should be set to 'Enforcing'" {
[ 'Enforcing' = $(getenforce) ]
}

@test "Firewall should be enabled and running" {
systemctl is-active firewalld.service
systemctl is-enabled firewalld.service
}

@test "EPEL repository should be available" {
[ -n "$(yum repolist | grep 'epel')" ]
}

@test "Bash-completion should have been installed" {
# This file should exist if bash-completion was installed
[ -f /etc/profile.d/bash_completion.sh ]
}

@test "bind-utils should have been installed" {
# Ask version of dig, this should execute with exit status 0
dig -v
}

@test "Git should have been installed" {
git -version
}

@test "Nano should have been installed" {
nano -version
}

@test "Tree should have been installed" {
tree -version
}

@test "Vim-enhanced should have been installed" {
vim -version
}
```

```
@test "Wget should have been installed" {
wget -version
}

@test "Admin user ${admin_user} should exist" {
# The user should exist
getent passwd ${admin_user}
# and should be member of the `wheel` group
[ -n "$(groups ${admin_user} | grep wheel)" ]
}

@test "An SSH key should have been installed for ${admin_user}" {
local keyfile="/home/${admin_user}/.ssh/authorized_keys"
[ -f "${keyfile}" ]
[ -s "${keyfile}" ] # should be nonempty
[ $(stat -format="%a" "${keyfile}") = '600' ]

}

@test "Custom /etc/motd should have been installed" {
skip
[ -f /etc/motd ] # is a regular file
[ -s /etc/motd ] # is nonempty
}
```

4. Bespreking van resultaten

4.1 Toegankelijkheid en performantie

De besproken methodes bieden vanuit verschillende standpunten manieren om met Molecule te werken of hoe men het programma kan toepassen bij andere workflows. Zo werd er bij methode 1 besproken hoe men Ansible rollen op een eigen machine lokaal kan testen via Molecule

De methodes die eerder werden aangehaald bieden verschillende manieren en use-cases rond hoe men Molecule kan gebruiken om Ansible rollen te testen.

Methode 1 besprak het 'normale' geval waarbij er lokaal op een machine Docker containers worden aangemaakt en daarop Ansible wordt uitgevoerd. Wat deze manier van werken vooral met zich meebrengt is dat het zeer gemakkelijk is om dit op te zetten. Men hoeft enkel Docker op het systeem geïnstalleerd te hebben en met behulp van twee commando's kan men direct aan de slag met Molecule. Bovendien is het ook veel lichter. Tabel 4.1 geeft een vergelijking hoeveel *system resources* Molecule en Docker verbruiken in tegenstelling tot een virtuele machine.

Deze statistieken zijn de gemiddeldes die werden opgemeten tijdens twee experimenten waarbij de metrieken werden opgenomen. Het verloop gebeurde als volgt: eerst werd een VM (of container) opgestart en werd er zo'n 15 seconden gewacht zodat het RAM en CPU verbruik zou stabiliseren. Via een generieke system monitoring tool werden de metrieken in het oog gehouden en na de wachtperiode werd een screenshot genomen van het programma. De bovenstaande waarden zijn inclusief de resources die nodig zijn voor de monitoring- en het screenshottool dus in principe kan men een 100 MiB aftrekken voor een iets preciezer resultaat waarbij de monitoring tools niet inbegrepen zijn. Belangrijk om

Situatie	CPU vebruik	RAM verbruik
Idle	Tussen 0 en 10 %	1.8 GiB
1 VM	10 á 20 %	2.2 GiB
2 VM's	10 á 20 %	3 GiB
2 containers	10 á 20 %	2 GiB

Tabel 4.1: Een tabel met performantie metrieken

te vermelden is dat deze virtuele machines en containers enkel werden opgestart, Ansible werd dus nog niet uitgevoerd.

In sommige gevallen is het voor bepaalde ontwikkelaars niet mogelijk om lokaal Ansible rollen te testen, allezins toch niet rechtstreeks. Zo kan het zijn dat hun pc daar niet krachtig genoeg voor is, ook al is Docker/Podman een relatief lichte applicatie. Of in andere gevallen gebruiken zij een Linux distributie waarbij het een grote opgave is om Docker te installeren. Voor hun zijn er dan twee opties: men kan de Ansible roles exclusief via Travis CI testen of als de hardware het toelaat is er ook de mogelijkheid om een virtuele machine te gebruiken waarbij ofwel Docker of Podman gemakkelijk is om te installeren.

De nieuwste update van Ubuntu, die uitgebracht is in april, heeft Docker rechtstreeks beschikbaar gesteld in hun software repositories. Daar is het dus mogelijk om met één commando Docker te installeren. Hetzelfde verhaal geldt ook voor recente Red Hat gebaseerde Linux distributies (Centos, Fedora, RHEL) waarbij Podman ook direct beschikbaar is hun software repositories.

Molecule heeft als voordeel dat het zowat op elke distributie gemakkelijk verkrijgbaar is doordat het via de Python package manager (PIP) wordt geïnstalleerd.

4.2 Podman

Over Podman gesproken, het heeft helaas wel nog zijn beperkingen, wat kan verklaren waarom Docker de standaard driver is voor Molecule. Zo is het netwerkgedeelte bij rootfull Podman containers semi-onvoorspelbaar als men alles automatisch laat afhandelen. Het is natuurlijk mogelijk om dit manueel allemaal in te stellen maar dat is een redelijk uitgebreid proces en het is iets dat buiten de scope van dit onderzoek valt, aangezien het doel is om de workflow te verbeteren, en niet om het verder uit te breiden en complexer te maken.

Is Podman dan helemaal onbruikbaar bij Molecule? Het hangt af van de Ansible role die getest wenst te worden. Als voorbeeld wordt even de Ansible BIND role van Bert Van Vreckem genomen. Zijn role kan twee types van DNS-servers installeren: een zogenaamde Master server en een Slave server. Een master server is de hoofd DNS-server en een slave server neemt alle data over van deze Master, maar kan zelf geen aanpassingen doorvoeren. Hoe wordt er beslist wie de Master of Slave rol krijgt? Dit gebeurt door bij de variabelen de optie 'bind_master_server_ip' in te vullen. Wanneer Ansible wordt uitgevoerd zal het

de IP-addressen vergelijken van alle machines waar deze role op wordt uitgevoerd. Komt het IP-adres overeen met die van de ingestelde variabele, dan krijgt het de Master role en in andere gevallen de Slave role.

Via Rootfull Podman containers kan dit nagebootst worden maar zoals eerder werd bewezen verschuift het IP-adres dat verkregen wordt elke keer er een nieuwe container wordt aangemaakt. Dit betekent dat als men *per se* met Podman wenst te werken, men dan de bind_master_server_ip variabel constant zou moeten aanpassen om te voorkomen dat alle servers de slave role toegewezen kregen. Een poging om zelf een netwerk te maken via Podman en de containers daar aan te laten deelnemen werd meerdere keren ondernomen tijdens dit onderzoek, maar helaas is er geen consistente oplossing voor dit probleem gevonden.

Bij andere Ansible rollen waar netwerken en IP-addressen geen rol spelen is Podman een perfect alternatief voor Docker, zij het met zijn limieten uiteraard. Zo is Podman niet beschikbaar voor Travis CI testen aangezien het besturingssysteem daarachter te verouderd is.

4.3 Docker

In de meeste gevallen is het dus aangeraden om Docker te gebruiken aangezien de kans dat het fouten zal veroorzaken veel kleiner is dan bij Podman. Het feit dat er met Docker netwerken kan gewerkt worden (in beperkte mate weliswaar) is een interessante trekpleister voor iets meer geavanceerdere Ansible rollen, vooral naar netwerken toe. De grootste troef van Docker in samenwerking met Molecule is dat het zeer vlot kan omgezet worden naar een werkbare Travis CI integratie, hoofdzakelijk omdat de activatie van Docker afgehandeld kan worden met een simpele optie.

Echter, het grootste voordeel van Docker is dat het aanzienlijk veel meer documentatie, handleidingen en FAQ materiaal ter beschikking stelt in tegenstelling tot Podman. Dit geldt ook voor de combinatie met Molecule. Een zeer oppervlakkige vergelijking is een zoekopdracht via Google. De zoekterm ‘molecule docker‘ levert zo’n 247 duizend resultaten op terwijl ‘molecule podman‘ slechts 66.800 matches weet te vinden.

Ook videos rond Docker zullen veel meer te vinden zijn op videoplatformen zoals Youtube.

Een mogelijke verklaring voor deze snelle vergelijkingen is dat Docker nu eenmaal veel ouder en volwassener is dan Podman. Docker werd in 2013 gelanceerd terwijl Podman nog maar sinds 2018 actief is. Dit betekent dat Docker een 5 jaar voorsprong heeft naar documentatie en handleidingen toe. Het feit dat Docker nu een marktleiderspositie heeft aangemomen speelt uiteraard ook een grote rol waarom er veel meer leermateriaal hierrond beschikbaar is.

4.4 De beperkingen van Molecule

Wanneer men met Molecule in het begin gaat experimenteren zal het onvermijdbaar zijn dat er foutmeldingen zich zullen voordien. Dit is namelijk één van de nadelen van Molecule, de foutmeldingen die worden gegeven zijn meestal beperkt of zelfs onbruikbaar. Het ‘molecule –debug (commando)‘ kan iets meer informatie geven, maar in de meeste gevallen zal het weinig verlichting brengen. Het probleem is namelijk dat de ontwikkelaars van Molecule gekozen hebben om vele details niet op het scherm te tonen die Ansible uitvoert, dit ziet men vaak via volgende lijn die regelmatig wordt getoond:

```
"censored": "the output has been hidden due to
the fact that 'no_log: true' was specified for this result"
```

Hoogstwaarschijnlijk kan men dit omzeilen door de broncode van Molecule aan te passen. Het probleem is echter dat de broncode nogal beperkt gedocumenteerd is waardoor men al een hele zoektocht moet doen om het specifieke deel te vinden waar men meer informatie bij wenst.

Ook de documentatie rond Molecule zou iets uitgebreider mogen zijn. Zo moet men vaak meer beroep doen op de docker_container module van Ansible zelf om te achterhalen wat de functie is van alle mogelijke configuratieopties. De issues pagina op de Github repository van Molecule kan enkele problemen helpen oplossen maar het zou interessant zijn mocht de FAQ pagina op de website van Molecule zelf up-to-date worden gehouden met gekende problemen en hoe men deze kan oplossen.

Methode 3 werd uitgevoerd om te achterhalen of Molecule op een Ubuntu 16.04, hetzelfde besturingssysteem dat Travis CI gebruikt, een merkbaar verschil zou opleveren naar stabiliteit en foutmeldingen toe. De observatie is echter dat er in feite geen merkbaar verschil is. De methodes die tijdens dit onderzoek gebruikt maakten van Travis CI hebben op 4 verschillende type systemen (Pop!_OS, Centos 7 en 8 en Ubuntu 16.04) telkens hetzelfde resultaat opgeleverd.

4.5 Travis CI en de Ansible role van Bert Van Vreckem

Hoofdstuk 3.6 heeft aangetoond dat het Travis CI verhaal gemakkelijk van manuele Docker tests overschakelt naar een Molecule implementatie. Sterker nog, mocht men de Docker scripts vanaf 0 moeten opstellen dan zou Molecule immens veel tijd, moeite en complexiteit besparen aangezien het programma dit allemaal op zich neemt. Een YAML bestand is bovendien ook veel leesbaarder dan een een Bash script.

Vergelijk even de manuele docker testen met het Molecule alternatief. ‘docker-tests.sh‘ is een redelijk groot bestand van 217 lijnen dat, hoewel het uitstekend gedocumenteerd is, toch minder begrijpbaar en aantrekkelijk eruitziet in vergelijking met het ‘molecule.yml‘ bestand dat hetzelfde resultaat realiseert, maar dan met slechts 48 lijnen. Het ‘test.yml‘

bestand is bij Molecule ook niet meer nodig aangezien het geïntegreerd wordt binnen het converge.yml configuratiebestand. De bestanden die verantwoordelijk zijn voor de acceptatietesten worden overgenomen en een klein verify.yml Ansible playbook zorgt ervoor dat deze uitgevoerd worden.

Hetzelfde geldt ook voor de configuratie van Travis CI. Het ‘install‘ gedeelte is naar omvang ongeveer hetzelfde, maar het indrukwekkende is dat het ‘molecule test‘ commando exact hetzelfde resultaat zal geven als de 5 lijnen bij het huidige travis.yml bestand. En inderdaad, met behulp van variabelen die Molecule ondersteunt, kan er ook gebruik gemaakt worden van de Travis CI’s environment matrix.

Manueel docker testen opstellen brengt uiteraard meer flexibiliteit en mogelijkheden met zich mee naar testen toe maar voor de meeste generieke use-cases, waaronder de BIND-role van Bert Van Vreckem, kan Molecule een uitstekend alternatief zijn.

5. Conclusie

Dit onderzoek heeft aangetoond dat Molecule in de meeste gevallen een uitstekend medium is om Ansible rollen te testen. Voor Ansible rollen die gebruik maken van Linux of andere programma's die niet namespaced zijn kan men beter een virtual machine orchestrator zoals Vagrant overwegen om deze specifieke zaken te testen via VM's. Het nadeel aan Molecule is echter dat de documentatie nog enkele vormen van verbetering kan gebruiken en het troubleshooting proces zou ook geoptimaliseerd kunnen worden door meer output informatie te voorzien via de console, wat op dit moment nog vaak gecensoreerd wordt.

Podman is alsook in sommige gevallen een waardig alternatief voor Docker, vooral bij simpele use cases zoals een webserver. Het rootless aspect van Podman helpt hier ook bij door het systeem iets meer weerstand te bieden dankzij de verminderde privileges die deze containers hebben, in tegenstelling tot rootfull containers en Docker. Een onderzoek dat kan verklaren waarom Molecule of de Podman driver ervoor zorgt dat de meegegeven IP-adressen altijd verschuiven in plaats van opnieuw in te stellen zou Podman nog aantrekkelijker maken om deze boven Docker te gebruiken.

Docker blijft in ieder geval wel de aangeraden driver om met Molecule te werken gezien de overweldigende documentatie en handleidingen die beschikbaar zijn in tegenstelling tot Podman. Hoogstwaarschijnlijk zal Podman wel meer gebruik zien in de toekomst wanneer het enkele kinderziektes achter zich kan laten alsook de documentatie en handleidingen kan verhogen en vooral toegankelijker maken.

Ook naar CI/CD omgevingen kan Molecule redelijk eenvoudig geïntegreerd worden. Dit onderzoek heeft zich vooral op Travis CI gefocust waarbij er twee aanpakken werden gebruikt. Een klassieke aanpak met één job die uitgevoerd wordt en een environment matrix waardoor verschillende jobs tegelijkertijd meerder Linux distributieimages kunnen

testen. De documentatie van Molecule geeft ook een beknopt voorbeeld van hoe een Jenkins Pipeline opgebouwd zou kunnen worden. Het voorbeeld met de BIND role van Bert Van Vreckem heeft alleszins aangetoond dat het travis configuratiebestand iets korter is, maar de grootste verbetering vindt men in de code die instaat voor het beheer van Docker containers, waarbij Molecule toch wel duidelijk veel eenvoudiger is voor de meeste doelgroepen. Mocht men vanaf 0 beginnen is Molecule bovendien ook veel gemakkelijker en sneller om op te zetten vergeleken met een manuele aanpak via scripts.

Tijdens dit onderzoek werd er ook achterhaald of het gebruik van een Ubuntu 16.04 VM evenveel of minder afwijkingen zou opleveren wanneer het werd vergeleken met de resultaten van een Travis CI omgeving. Daaruit werd duidelijk dat het type Linux distributie geen rol speelde. Met andere woorden, een bepaalde Molecule configuratie behaalde bij alle gebruikte distributies (Arch Linux, Manjaro Linux, Pop!_OS 20.04 LTS, Ubuntu 16.04, Centos 7 & 8) telkens dezelfde resultaten.

In een vervolgstudie zou er onderzocht kunnen worden hoe de gekende limitieten en pijnpunten van Podman kunnen opgelost worden zodat Molecule niet enkel beperkt is tot Docker omwille van zijn stabiliteit. Voor sommige gebruikers is het nu eenmaal veel gemakkelijker om Podman te installeren of willen zij namelijk gewoon geen gebruik maken van Docker. Hoewel dit eerder een taak is voor de ontwikkelaars van Molecule zou een onderzoek naar de herziening van de manier waarop Molecule foutcodes geeft ook zeer interessant zijn. Nu is het namelijk zo dat foutcodes te vaak worden gecensoreerd en in de meeste gevallen moet men dan beroep doen doen op externe bronnen en alsook een vorm van gokwerk om een bepaald probleem binnen Molecule te kunnen oplossen.

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 en State-of-the-Art

Systeem automatisatie en configuratie is nog nooit zo eenvoudig geweest. Wat men enkele decenia geleden manueel moest instellen is gevolueerd tot scripting tot op de dag van vandaag waar er software zoals Ansible is ontwikkeld om speciaal deze taak op zich te nemen met behulp van herbruikbare code. Andere applicaties zoals Github [1] maakt het eenvoudig om code te delen met de wereld en dankzij Travis CI kan deze nieuwe code ook onmiddelijk getest worden om op zoek te gaan naar bugs. (McAllister, 2017) en (Geerling, 2015)

Wat men tegenwoordig vaak doet is deze nieuwe software via Travis CI [2] laten testen en vervolgens zelfs inzetten op volwaardige virtuele machines of op zogenaamde containers, geïsoleerde applicaties die direct draaien bovenop de kernel. De meest populaire methode om deze containers te bouwen is ongetwijfeld met behulp van Docker dat wordt georchestreerd door Kubernetes [3]. (Sysdig, 2019)
Dankzij Ansible [4] is het ook heel eenvoudig geworden om complexe operaties en confi-

[1] https://www.github.com/
[2] https://travis-ci.com/
[3] https://kubernetes.io/
[4] https://www.ansible.com/

guraties te reduceren tot het invullen van enkele variabelen met behulp van zogenaamde rollen. De rollen zelf opmaken vergt al iets meer tijd en kan soms als frustererend worden ervaren door de strikte yaml opmaak. Om die reden is het voor een developer vereist een omgeving te hebben waar hij of zij snel nieuwe code kan testen met zo weinig mogelijk tussenstappen. Een mogelijkheid is om Vagrant [5]met een vooraf geconfigureerde Ansible-Skeleton te gebruiken waar een *vagrant provision* commando voldoende is om een Ansible role te testen. Een andere methode dat meer populariteit wint is het gebruik maken van Travis CI dat voordien hier ook al werd vermeld. Een developer uploadt zijn aanpassingen naar github waarna een mechanisme van Travis CI in werking treedt die de aanpassingen uitvoert op een omgeving. Zoals eerder vermeld kan zo een omgeving een virtuele machine zijn dat draait op een volwaardig besturingsysteem. Anderzijds zal tegenwoordig de keuze eerder een container worden aangezien het lichter is VM (Virtuele Machine), betrouwbaarder en meestal ook beter beveiligd.

Deze tweede manier van werken brengt wel enkele problemen met zich mee, zeker als men Docker gebruikt voor de containers op te zetten. Eerst en vooral zijn er de stability issues die toch wel onaangename gevolgen kan dragen voor de gebruiker zijn systeem. Anderszijds zijn er ook enkele *by-design* eigenschappen van Docker die een negatieve invloed hebben op de workflow van Travis CI. Het feit dat er een inefficiënte shell script nodig is om Docker containers te bouwen frusteerd vele DevOps tegenwoordig omdat zulke praktijken "niet meer van deze tijd"zijn.

Het kan ook interessant zijn om te bekijken of het voor onze onderzoeksvraag wel zinvol is om Travis CI te gebruiken. Hoewel de populariteit van CI/CD platformen, waaronder Travis, aanzienlijk is toegenomen de laatste jaren wordt er soms te veel de voorkeur gegeven aan deze manier van werken. (Gallaba & McIntosh, 2020)

Via dit onderzoek wordt er bekeken of er manieren zijn om de workflow te vereenvoudigen of zelfs compleet te vervangen met een nieuwe stack van software die hierboven nog niet allemaal werden beschreven. Anderzijds wordt er tegelijkertijd getest of het gebruik van Docker tegenwoordig nog steeds altijd de norm kan zijn als we denken aan containers. Misschien zijn er ook alternatieven die men nog niet genoeg kansen heeft gegeven? Misschien zijn er inmiddels software pakketten ontwikkeld die beschreven problemen vermindert of zelf compleet elemineert.

A.2 Methodologie

Voor dit onderzoek zullen enkele scenario's opgesteld worden. Bij elke scenario worden er andere stukken software en technologieën gebruikt waarbij de workflow en features worden getest op basis van volgende criteria:

[5]https://www.vagrantup.com/

- Gebruiksvriendelijkheid
- Documentatie en community
- Schaalbaarheid
- Benodigde resources
- Stabiliteit

Elke scenario gaat alsook door een bepaalde stappenplan:

1. Er wordt een aanpassing aangebracht aan de Ansible rol
2. De aanpassingen worden naar github opgeladen
3. Travis CI merkt de aanpassingen op en begint met het opbouwen van de opgeving
4. De role wordt uitgevoerd op de omgeving

Voorbeelden van scenario's waarbij de gebruikte technologieën worden vermeld:

- S(0): Ansible, Github, Virtuele machine en shellscript via Travis CI
- S(1): Ansible, Github, Docker container en shellscript via Travis CI
- S(2): Ansible, Github, Virtuele machine en Molecule [6] via Travis CI
- S(3): Ansible, Github, Docker container en Molecule via Travis CI
- S(4): Ansible, Github, Podman [7], Buildah [8] en Molecule via Travis CI
- Extra: experimenterne met Kubernetes, Cri-O en/of Openshift

Met behulp van deze scenario's en criteria hopen we op zoek te gaan naar een workflow die het hoogste scoort op deze punten. Daarnaast is het perfect ook mogelijk dat we een alternatieve scenario vinden die hier niet beschreven staat die ook goed zou kunnen scoren. We gaan er ook telkens van uit dat elke bovenstaande scenario ook uitvoerbaar is op een lokale host. Elke scenario een score geven van 1 tot en met 5 kan gebruikt woorden om de scenario's te beoordelen.
In het geval dat scenario 4 of een alternatieve scenario als winnaar uit de bus komt kan het zinvol zijn om dit te laaten testen door een deskundig persoon die vertrouwd is met het onderwerp. In dit geval kan dit de co-promotor en opdrachtgever zijn, Bert Van Vreckem.

A.3 Verwachte resultaten

Op basis van opgestelde scenario's wordt er verwacht dat Scenario 0 en 1 altijd een werkbaar resultaat zal geven maar dat ze niet het niet de ideale workflow zal zijn voor deze situatie. Scenario 0 gebruikt een VM wat zwaarder is dan een container en zowel Scenario 0 als 1 gebruikt een shellscript in Travis CI om de aanpassingen van github aan te brengen

[6] "https://molecule.readthedocs.io"
[7] 'https://podman.io/'
[8] https://buildah.io/

naar de omgevingen, iets wat niet altijd een gebruiksvriendelijke opgave is.
Het gebruik van Molecule probeert dit probleem te elimineren maar dit onderzoek zal tevens ook uitzoeken of dit effectief de gebruiksvriendelijkheidsprobleem oplost.

Hoewel scenario 4 nieuwe tools gebruikt die niet zo bekend zijn in vergelijking met de andere software kunnen we niet uitsluiten dat ze verassend uit de hoek kan komen. Redhat, het bedrijf achter Ansible, Podman en Buildah heeft met deze technologieën bewezen dat applicaties steunen die gebruikvriendelijk, schaalbaar en snel zijn.

A.4 Verwachte conclusies

Uit dit onderzoek verwachten we te concluderen dat gebruik van nieuwe tools zoals Podman, Buildah en Molecule een verrassend en zelfs een volwaardig alternatief kan zijn tegenover de workflow die vandaag de dag gehanteerd wordt. Hoewel virtuele machine's in sommige niche gevallen nog steeds de voorkeur kunnen krijgen zal men naar de toekomst toe zich meer en meer richten naar containers. Ook is het mogelijk een conclusie te stellen dat Docker zal nog een lange tijd populair zal blijven omdat deze relatief gezien gemakkelijk op te zetten is samen met Kubernetes omwille van veelzijdige mogelijkheden eens men de complexe bediening onder de knie heeft. Tenslotte kan men de voordelen van tools zoals Openshift en Cri-O ook niet uitsluiten.

Bibliografie

About pull requests. (2020). Verkregen 26 april 2020, van https://help.github.com/en/github/collaborating-with-issues-and-pull-requests/about-pull-requests

Arch Wiki - SELinux. (2020, februari 16). Verkregen 19 mei 2020, van https://wiki.archlinux.org/index.php/SELinux

BMitch. (2016, augustus 26). *Why is Systemd not present in Ubuntu Docker Images?* Verkregen 31 maart 2020, van https://stackoverflow.com/questions/39169403/systemd-and-systemctl-within-ubuntu-docker-images

Datadog. (2018, juni). *8 surprising facts about real Docker adoption*. Verkregen 17 april 2020, van https://www.datadoghq.com/docker-adoption/

Docker Guides | Use volumes. (2020, mei 18). Verkregen 19 mei 2020, van https://docs.docker.com/storage/volumes/

Fork and Exec. (g.d.). Verkregen 23 april 2020, van https://www.bottomupcs.com/fork_and_exec.xhtml

Fowler, M. (2016, mei 1). *Continuous Integration*. Verkregen 29 april 2020, van https://martinfowler.com/articles/continuousIntegration.html

Gallaba, K. & McIntosh, S. (2020). Use and Misuse of Continuous Integration Features: An Empirical Study of Projects That (Mis)Use Travis CI. *IEEE Transactions on Software Engineering*, *46*(1), 33–50.

Gardner, C. (2019, augustus 7). *The Forrester Wave: Infrastructure Automation Platforms, Q3 2019*. Verkregen 30 maart 2020, van https://www.redhat.com/cms/managed-files/ma-forrester-wave-infrastructure-automation-analyst-paper-f18968-201908-en.pdf

Geerling, J. (2015, oktober 4). *Ansible for DevOps: Server and configuration management for humans*. Verkregen 1 december 2019, van https://leanpub.com/ansible-for-devops

Geerling, J. (2020, maart 13). *Ansible for Kubernetes: automate app deployment on any scale with Ansible and K8s*. Verkregen 3 april 2020, van http://leanpub.com/ansible-for-kubernetes

Graber, S. (2014, januari 17). *Introduction to unprivileged containers*. Verkregen 10 mei 2020, van https://stgraber.org/2014/01/17/lxc-1-0-unprivileged-containers/

Holst, A. (2020, maart 2). *Ansible Usage Statistics*. Verkregen 20 maart 2020, van https://www.statista.com/statistics/511293/worldwide-survey-cloud-devops-tools/

Linux Programmer's Manual - Namespaces. (2020, april 11). Verkregen 19 mei 2020, van https://man7.org/linux/man-pages/man7/namespaces.7.html

McAllister, J. (2017, juli 20). *Implementing DevOps with Ansible 2*. Verkregen 2 december 2019, van https://www.packtpub.com/networking-and-servers/implementing-devops-ansible-2

Molecule Configuration Guide - Variable Substitution. (2020, februari 23). Verkregen 17 mei 2020, van https://molecule.readthedocs.io/en/latest/configuration.html

Patton, R. (2006). *Softare testing* (2de ed.). Indianapolis: Sams Publishing.

Podman Installation Guide. (g.d.). Verkregen 9 april 2020, van https://podman.io/getting-started/installation.html

Sysdig. (2019, oktober 29). *2019 Container Usage Report*. Verkregen 25 november 2019, van https://sysdig.com/resources/papers/2019-container-usage-report