



Faculteit Bedrijf en Organisatie

Titel

Kenzie Coddens

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

Promotor:
Olivier Rosseel
Co-promotor:
Raf Boon

Instelling: Aucxis

Academiejaar: 2019-2020

Tweede examenperiode

Faculteit Bedrijf en Organisatie

Titel

Kenzie Coddens

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

Promotor:
Olivier Rosseel
Co-promotor:
Raf Boon

Instelling: Aucxis

Academiejaar: 2019-2020

Tweede examenperiode

Woord vooraf

In de eerste plaats wil ik Aucxis bedanken voor het aanbieden van de use case en voor mij de mogelijkheid te bieden om hierover een bachelorproef te schrijven. Ik wil in het bijzonder Raf Boon bedanken voor zijn tijd en uitleg over de werking binnen het bedrijf. Ook wil ik Olivier Roseel bedanken voor de begeleiding van mijn bachelorproef.

Ook wil ik mijn dank uitdrukken naar de gezondheidszorg voor hun werk tijdens de Corona crisis. Dit onderzoek is grotendeels geschreven tijdens de lockdown. Dit heeft mij een bijzonder gevoel gegeven tijdens het schrijven van dit onderzoek. Het was een bijzondere situatie.

Initieel was het niet de bedoeling om zo diep op CI/CD in te gaan. Ik wou vooral aan de slag gaan met Cloud platformen aangezien we hier weinig aanraking mee hadden tijdens onze studie. Dit onderwerp is er gekomen door een voorstel dat ik naar Aucxis heb gedaan. Dit onderwerp hebben we dan in samenspraak preciezer gemaakt zodanig dat het onderwerp niet te ruim werd. Devops procedures interesseren mij enorm. Naar mijn mening is Devops een van de belangrijkste verzameling procedures in de IT-sector. Zonder deze procedures zouden projecten veel meer tijd en geld kosten. Ik had ook nooit gedacht dat CI/CD zo een populair thema is binnen de IT-sector. Ook ben ik geschrokken van hoe weinig onderzoek er is uitgevoerd naar de services en producten voor CI/CD op Cloud platformen. Er bestaan talloze studies over de prestatie van de platformen en hoe deze zijn opgebouwd. Ook bestaan er veel case studies. Echt onderzoek naar een algemeen antwoord op welk Cloud platform er nu het best mogelijkheden voor CI/CD heeft, is moeilijk te vinden. Met dit onderzoek hoop ik de sector helpen een antwoord te bieden aan dit probleem. Hopelijk komt de ervaring van dit onderzoek van pas op een latere datum.

Samenvatting

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus.

Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetur adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Inhoudsopgave

1	Inleiding	17
1.1	Probleemstelling	18
1.2	Use Case	18
1.3	Onderzoeksvraag	19
1.4	Onderzoeksdoelstelling	19
1.5	Opzet van deze bachelorproef	19
2	Stand van zaken	21
2.1	Continuous Integration & Continuous Deployment	21
2.1.1	Software release management voor component gebaseerde software	22
2.1.2	Uitdagingen en problemen in het Release Management Process: een Casestudie	24
2.1.3	Methodes en Systemen voor Software Release Management	25
2.1.4	Continuous Integration en de Tools	27

2.1.5	DeVops	27
2.1.6	Continuous Integration en Release Pipelines bouwen door het toepassen DevOps Principes: een Casestudie bij Varidesk	28
2.2	Amazon AWS	29
2.2.1	Prestatie Analyse van hoge prestatie reken applicaties op Amazon Web Services Cloud	29
2.3	Microsoft Azure	30
2.3.1	Microsoft Azure en Cloud computing	30
2.3.2	Microsoft Azure Documentation	31
3	Methodologie	35
4	Kandidaat Selectie	37
4.1	Azure DevOps	37
4.2	Google Cloud	39
4.3	Amazon Web Services	41
4.4	IBM Cloud	44
4.5	Andere	45
4.6	Gebruiksvriendelijkheid	46
4.6.1	Java Amazon AWS	49
4.6.2	Java Google Cloud Platform	50
4.6.3	Java Azure DevOps	52
4.7	Kandidaat	54
5	Proof Of Concept	57
5.0.1	Google Cloud Platform	60
5.0.2	Azure DevOps	65

6	Conclusie	73
A	Onderzoeksvoorstel	75
A.1	Introductie	75
A.2	Literatuurstudie	76
A.2.1	Conventional Software Testing Vs. Cloud Testing	76
A.2.2	Software Testing Based on Cloud Computing	76
A.2.3	Benchmarking in the Cloud: What It Should, Can, and Cannot Be	77
A.2.4	When to Migrate Software Testing to the Cloud?	77
A.3	Methodologie	77
A.4	Verwachte resultaten	78
A.4.1	Vergelijking van platformen	78
A.4.2	Proof of concept	78
A.4.3	beveiliging experiment	78
A.5	Verwachte conclusies	79
A.5.1	Vergelijking van platformen	79
A.5.2	Proof of concept	79
A.5.3	beveiliging experiment	79
	Bibliografie	81

Lijst van figuren

2.1 Figuur uit (van der Hoek & Wolf, 2002). Figuur toont een simpel diagram van hoe een software release management systeem opgebouwd kan zijn.	23
2.2 Figuur uit (Methodes and systems for software release management, 2005). Diagram van een niet geautomatiseerde versie beheer procedure.	26
2.3 Figuur van (https://p2zk82o7hr3yb6ge7gzxx4ki-wpengine.netdna-ssl.com/wp-content/uploads/azure-vm-types-comparison-1.jpg). Chart met alle VM tiers van azure en hun specifieke code.	32
4.1 Prijzen van Azure DevOps. Afbeelding van Azure DevOps website. Figuur toont de prijzen voor Azure DevOps op een beknopte manier.	39
4.2 GCB prijzen. Figuur van Google Cloud website. Figuur toont de prijzen voor Google Cloud Platform op een beknopte manier.	41
4.3 Figuur van AWS website. Figuur toont de prijzen van AWS per rekenkracht, OS en verstrekken minuut.	43
4.4 compileer prijzen AWS.Figuur van AWS website. Figuur toont berekening van prijzen per compileer minuut voor AWS.	43
4.5 extra kosten. Figuur van AWS website. Figuur toont prijs voor opslag op AWS.	44

4.6 BronCode locatie seleceteren op AWS. Deze figuur toont het seleceteren van een GitHub repositorie bij het aanmaken van een pipeline op Amazon AWS.	50
4.7 Een ingebouwd dashboard op Azure DevOps. Dit dashboard geeft de resultaten van de uitgevoerde testen tijdens de laatste compilatie.	54
4.8 Overzicht van datacenter locaties Google Cloud. Figuur van Google Cloud website. Figuur toont de connectiviteit van de verschillende datacenters verspreid over de wereld.	55
5.1 New Project scherm op Google Cloud. Figuur toont het scherm om een nieuw project te creëren op Google Cloud.	60
5.2 Storage Bucket scherm op Google Cloud. Figuur toont het scherm met een gecreëerde Storage Bucket en de geconfigureerde opties op Google Cloud.	61
5.3 Scherm met Trigger voor een bepaalde pipeline op Google Cloud. Figuur toont het scherm van Google Cloud met een gecreëerde Trigger op de tak 'Build' op GitHub.	63
5.4 Resultaat van de applicatie. Figuur toont de uitvoer van de gecompileerde applicatie op een lokale Windows Server 2019. Applicatie is gekopieerd door middel van <i>script 5.3</i>	64
5.5 Console uitvoer van een pipeline op Google Cloud. Figuur toont het resultaat van de compilatie stappen op Google Cloud. Ook zijn er een aantal andere logs te bekijken op dit scherm.	64
5.6 Compileer resultaten dashboard op Google Cloud. Figuur toont een simpel dashboard met gegevens over de compilatie pipeline.	65
5.7 Hoofd scherm Azure DevOps. Figuur toont een nieuw aangemaakt project op Azure DevOps.	66
5.8 Test-pass dashboard Azure DevOps. Figuur toont een simpel dashboard met gegevens over de uitgevoerde testen.	68
5.9 Release pipeline configuratie. Figuur toont hoe een release pipeline op Azure DevOps wordt geconfigureerd. In dit geval wordt er een Ubuntu machine geconfigureerd met een script.	70
5.10 Grafische weergave Azure DevOps pipeline. De figuur toont een kort overzicht van de totale pipeline op Azure DevOps.	71
5.11 Project overzicht dashboard Azure DevOps. Figuur toont een simpel dashboard met gegevens over de compilatie pipeline.	71

Lijst van tabellen

- 4.1 toont de gespendeerde tijd voor de gebruiksvriendelijkheid test per Cloud platform. De tijden staan in minuten. De Cloud platformen vergelijken goed met elkaar. Ook schaalt de complexiteit mee met de gespendeerde tijd. 54

Lijst van Codeblokken

4.1	Java bestand MessageUtil.java. Hoofd klasse van de applicatie <i>MessageUtil</i>	46
4.2	Java Test klasse MessageUtilTest.java, voor het testen van <i>MessageUtil</i> 4.1.	47
4.3	XML-bestand voor de Maven compiler. Definieert stappen voor het compileren van een java applicatie.	47
4.4	Output van het tree commando in een java applicatie bestanden structuur.	48
4.5	YAML-bestand voor het configureren van een pijpleiding op Amazon AWS.	49
4.6	YAML-bestand voor het configureren van een pijpleiding voor java op Google Cloud.	51
4.7	YAML-bestand voor het configureren van een pipeline voor java op Azure DevOps.	52
5.1	C# bestand MessageUtil.cs. Hoofd klasse van de applicatie <i>MessageUtil</i>	57
5.2	C# Test klasse MessageUtilTest.cs, voor het testen van <i>MessageUtil</i> 5.1.	58
5.3	Bash script filetrans.sh. Script installeert de juiste Linux packages en kopieert de bestanden met SFTP.	59
5.4	Tree van de bestanden structuur voor de Proof Of Concept op Google Cloud.	61
5.5	Bash script om een bepaalde folder te comprimeren tot een .tar.gz.	62
5.6	YAML-bestand voor de configuratie van de .Net pijpleiding op Google CLoud.	62
5.7	Tree van de bestanden structuur voor de Proof Of Concept op Azure DevOps.	66
5.8	YAML-bestand voor de configuratie van de .Net pipeline op Azure DevOps.	67
5.9	PowerShell script dat een map compresseert. Wordt gebruikt om alle .Net Framework afhankelijkheden mee te verpakken.	68
5.10	Verbeterd YAML-bestand voor de configuratie van de .Net pipeline op Azure DevOps.	69

1. Inleiding

Heden ten dage zijn er veel bedrijven die grote delen van hun server infrastructuur digitaliseren in de Cloud. Dit kan variëren van eenvoudige webapplicaties tot een volledige workflow. Deze situaties zijn vaak zeer use case specifiek. Daarbovenop helpt het ook niet dat er verschillende Cloud aanbieders zijn met elk hun specifieke serie van producten en oplossingen. Deze werken dan ook nog eens met verschillende prijzenstelsels. Daarbovenop verschillen de Cloud aanbieders ook in functionaliteit. Het kan dus vaak zeer complex zijn om een gepaste oplossing te vinden.

Een bepaalde workflow die belangrijk is binnen een softwarebedrijf, is continuous integration & continuous deployment (CI/CD). Deze moet ervoor zorgen dat elk stukje software dat door een programmeur of een team van programmeurs wordt opgeleverd, voldoet aan een aantal opgestelde kwaliteit eisen en uiteindelijk bij de klanten in een productieomgeving uitgerold kan worden. CI/CD automatiseert dit proces volledig.

CI/CD is ondertussen bij veel bedrijven geïmplementeerd. Desondanks is er maar weinig info te vinden over welke platformen of oplossingen nu specifiek de beste zijn. Als er al informatie of onderzoeken over bestaan, dan zijn deze meestal zeer specifiek en enkel in dat geval toe te passen. Daarom is het interessant om voor deze use case te bekijken wat er het best past. Wat is de beste strategie om deze pijpleiding te implementeren.

Om deze automatisatie te bereiken zijn er een aantal strategieën. Een daarvan is met containers werken. Dit zorgt ervoor dat de pijpleiding flexibel en herhaaldelijk is op zo goed als elk Cloud platform dat 'Platform as a Service' (PaaS) aanbiedt. Ook zorgt het ervoor dat er containers op maat gemaakt kunnen worden om optimaal van de functies van een bepaald platform gebruik te maken. Dit onderzoek zal op maat gemaakte compileer en test containers buiten beschouwing houden. Dit omdat deze oplossing vaak arbeidsintensief

is om te ontwikkelen en minder gebruiksvriendelijk. Dit onderzoek zal dus zo veel mogelijk gebruikmaken van ingebakken en voor gemaakte oplossingen.

1.1 Probleemstelling

CI/CD Procedures zijn geen nieuw idee. Beschrijving over wat deze procedures zijn en hoe ze het best worden opgesteld zijn goed uitgediept. CI/CD wordt gebruikt om snel kwalitatieve software op te leveren en uit te rollen naar productie omgevingen bij klanten. Ook Aucxis wil zich meer toeleggen op betere en kwalitatievere software, zodanig dat er sneller kwalitatieve support aan de klanten aangeboden kan worden.

Echter, hoe deze nu het best worden geïmplementeerd op zowel praktisch als technisch vlak, bestaat er nog geen eenduidig algemeen antwoord. De meeste recente onderzoeken beschrijven vaak specifieke gevallen en gebruiken vaak compleet op maat gemaakte oplossingen per Cloud platform. Dit is ook niet zo onlogisch. Onderzoeken over welk Cloud platform optimaal is voor CI/CD zijn vrijwel niet te vinden. Ook zijn er weinig tot geen onderzoeken te vinden over de vergelijking van verschillende mogelijkheden van Cloud platformen. Er bestaan wel talloze onderzoeken over de prestatie van de verschillende platformen. Zo een soort onderzoeken zullen ook in acht worden genomen in dit onderzoek.

Deze paper tracht een duidelijk beeld te vormen voor Aucxis over welke mogelijkheden er nu bestaan, welk platform specifieke CI/CD mogelijkheden heeft en wat er nu het meest gebruiksvriendelijk lijkt.

1.2 Use Case

Aucxis werkt momenteel veel met ‘.NET’ projecten. Als IDE wordt er Visual Studio gebruikt. Als versiebeheer gebruiken ze hiervoor ‘Team Foundation Server’ (TFS). Dit was een voor een lange tijd een goede oplossing. De testen die worden uitgevoerd zijn vooral functionele testen. Dit is niet ideaal. Er worden veel fouten en bugs vastgesteld op moment van implementatie bij klanten. Dit kan leiden tot veel frustraties bij de mensen verantwoordelijk voor het implementeren van de software op locatie. Anderzijds gaat er veel tijd en geld verloren met het verplaatsen tussen klant en bedrijf. Aucxis heeft recent de keuze gemaakt om meer te investeren in kwaliteit. Aangezien Aucxis al een Microsoft partner is, was de keuze voor Azure DevOps niet moeilijk. Ook heeft dit minder impact op hun huidige werkwijze.

Zoals eerder aangehaald worden er vooral functionele testen uitgevoerd. Met de stap naar kwalitatievere oplossingen is er ook nood voor meerdere soorten testen. Het ideale zou zijn dat er naast functionele testen ook in een gecontroleerde omgeving getest kan worden. Daarna zou het programma bij een test omgeving van de klant uitgerold worden om daar getest te worden. In een finale stap zou het programma dan in productie uitgerold worden. Dit alles zou zo geautomatiseerd mogelijk moeten verlopen.

Azure DeOps lijkt hier het meest geschikt om deze functionaliteit te verkrijgen. Aucxis is reeds aan de overstap naar Azure Devops begonnen. Hierbij is niet echt stilgestaan bij andere mogelijkheden. Daarom de vraag om toch nog het aanbod van Azure met de andere Cloud platform aanbieders te vergelijken en het beste alternatief te selecteren.

1.3 Onderzoeksvraag

Welk Cloud platform heeft het meest geschikte aanbod voor CI/CD te implementeren, vertrekend vanuit een specifieke use case van Aucxis, naast het Azure Devops platform. Dit zonder arbeidsintensieve containers te maken voor ieder specifiek platform. Zo zal in dit onderzoek het Tekton framework buiten beschouwing gehouden worden. Dit framework staat immers toe dat ieder Cloud platform een geschikte kandidaat zo zijn. De criteria om een vergelijking te kunnen maken zijn: aanbod, gebruiksvriendelijkheid, haalbaarheid en prijs.

1.4 Onderzoeksdoelstelling

Welk Cloud platform nu mogelijk een beter alternatief is voor CI/CD zal worden aangevoerd in twee stadia. In een eerste fase worden alle kandidaten naast elkaar gelegd voor vergelijking. Er zal worden gekeken naar hun mogelijkheden, hoe de prijsstelsels werken, eerdere implementaties van andere toepassingen en de gebruiksvriendelijkheid. Daaruit wordt dan het beste alternatief voor Azure Devops gekozen. In de tweede fase zal met Azure Devops en het beste alternatief een Proof Of Concept uitgewerkt worden. Voor deze Proof Of Concept wordt er een pijpleiding voor een Microsoft specifieke applicatie opgesteld. Deze pijpleiding moet kunnen automatisch geactiveerd worden. Ook moet deze pijpleiding een gecompileerde applicatie uploaden naar een lokale test omgeving. Deze twee platformen worden dan enigszins met elkaar vergeleken. Dit alles zou een vergelijkend beeld moeten vormen over welk platform het meest geschikt is om met bestaande functies een pijpleiding te implementeren.

1.5 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 onderzoeksdomen, op basis van een literatuurstudie.

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

In Hoofdstuk 4 wordt besproken hoe gebruiksvriendelijk de Cloud platformen zijn aan de hand van een simpele opstelling. In dit hoofdstuk wordt de kandidaat voor de Proof Of

Concept geselecteerd.

in Hoofdstuk ?? wordt met de geselecteerde kandidaat uit voorgaande sectie, een Proof Of Concept opgesteld voor een Microsoft gerelateerde use case.

In Hoofdstuk 6 wordt een conclusie gegeven en een antwoord geformuleerd op de onderzoeks vragen. Daarbij wordt ook een aanzet gegeven voor toekomstig onderzoek binnen dit domein.

2. Stand van zaken

In het vorige hoofdstuk is de onderzoeksraag voor dit onderzoek voorgesteld. Ook is de use case van Aucxis kort besproken. In dit hoofdstuk worden eerder gemaakte studies besproken die aansluiten bij de scope van dit onderzoek. Deze literatuur studie moet dienen als vertrek punt voor de volgende hoofdstukken. Er wordt in dit hoofdstuk vooral gedefinieerd wat Continuous Integration & Continuous Deployment inhoud. Zo kan er een goed beeld worden gevormd van het belang van deze procedures. Ook probeert dit hoofdstuk de volgende hoofdstukken duidelijker te maken met deze studies.

2.1 Continuous Integration & Continuous Deployment

Een van de eerste vragen die opdook was: '*Hoe wordt software release management nu eigenlijk verwezenlijkt?*'. '*Zijn er bepaalde procedures of praktijken die worden gevolgd om dit in goede banen te leiden?*'. '*Welke technieken worden er gebruikt?*'. Dit leidde ertoe om in de eerste plaats in de richting van de ITIL-processen te zoeken (ITIL, Information Technology Infrastructure Library). Dit omdat het nauw aansluit bij software release management. Zeker als men denkt in de richting van support doeleinden.

ITIL is een verzameling van een aantal voorwaarden waaraan moeten worden voldaan in bepaalde situaties. Het zijn de procedures en technieken die een aantal van deze voorwaarden vervullen, die naar Continuous Integration & Continuous Deployment (CI/CD) hebben geleid.

2.1.1 Software release management voor component gebaseerde software

De paper (van der Hoek & Wolf, 2002) is een verslag over een jarenlange observatie van software release management praktijken. Deze paper is niet meer van de jongste maar bespreekt toch nog een aantal belangrijke kernideeën. De technische kant en de gebruikte software is minder relevant.

De paper (van der Hoek & Wolf, 2002) begint met de problematiek uit te leggen van software release management voor zowel de gebruiker als de ontwikkelaar. In de eerste plaats stelt de paper dat de eindgebruiker altijd het slachtoffer is. Tegenwoordig wordt er veel component gebaseerde software ontwikkeld. Een goed voorbeeld hiervan, zijn Linux packages. Dit maakt het niet altijd gemakkelijk voor de eindgebruiker om de juiste softwarecomponenten te vinden. Laat staan dat het de juiste versies zijn. Voeg daar dan nog aan toe dat sommige softwarebedrijven niet altijd oudere versies aanbieden, dat de eindgebruiker meerdere websites moet afspoelen en dat de eindgebruiker ook nog eens verantwoordelijk is voor het installeren en up-to-date houden van al die componenten. Dit is dus ver van een optimale situatie.

Het samenvoegen van verschillende componenten, het uitrollen naar de gebruikers en het updateen ervan, wordt beschreven als software release management. Software release management is enkel verantwoordelijk voor het beheren en het opslaan van de verschillende benodigde componenten en is dus niet bedoelt om de verschillende componenten zelf te compileren. Dit zorgt ervoor dat software release management tools compleet platform onafhankelijk kunnen zijn. Het limiteert daardoor wel de functionaliteit. In het bijzonder zijn de tools niet instaat om aan validatie of versie beheer te doen.

Om aan goede software release management te kunnen doen, is goede documentatie van alle verschillende componenten nodig. De paper stelt een aantal vereisten voor software release management voor. Dit is voor zowel de eindgebruiker als de ontwikkelaar. Deze hebben ze samengesteld uit hun eigen tien jaar lange ervaring. (van der Hoek & Wolf, 2002).

Minimale vereisten voor ontwikkelaars:

- Afhankelijkheden moeten expliciet zijn en gemakkelijk kunnen worden vastgelegd.
- Releases moeten consistent worden gehouden.
- De reikwijdte van een release moet controleerbaar zijn.
- Het releaseproces zou minimale inspanning aan de kant van de ontwikkelaar moeten inhouden.
- Er moet een geschiedenis van opvragen worden bijgehouden.

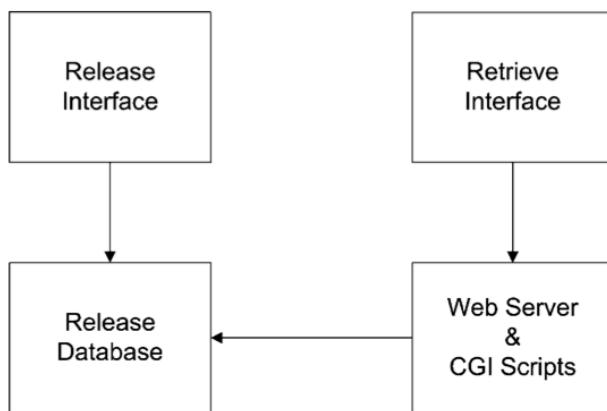
Minimale vereisten voor eindgebruiker:

- Beschrijvende informatie moet beschikbaar zijn.
- Er moet transparantie over de locatie worden geboden.
- Een component en zijn afhankelijkheden moeten als één archief kunnen worden opgehaald.
- Software-implementatie hulpmiddelen moeten de software-releasebeheertool kun-

nen gebruiken als bron voor componenten die moeten worden geïnstalleerd en geconfigureerd.

Hierna geeft de paper (van der Hoek & Wolf, 2002) verder uitleg over een software release management tool die de schrijvers van de paper zelf hebben ontwikkeld. Dit is minder interessant voor de doeleinden van dit onderzoek maar de ideeën achter deze tool kunnen een meerwaarde bieden bij de motivatie waarom software release management zo belangrijk is.

De bedoeling van deze tool is om enerzijds de informatie die gebruikt wordt in het releasebeheer proces te structureren en anderzijds locatietransparantie aan te bieden. De



Figuur 2.1: Figuur uit (van der Hoek & Wolf, 2002). Figuur toont een simpel diagram van hoe een software release management systeem opgebouwd kan zijn.

figuur 2.1 illustreert de architectuur van de software release management tool. Deze bestaat uit vier delen. Een logisch gecentraliseerde, maar fysiek gedistribueerde releasedatabase, een interface waarmee ontwikkelaars componenten in de releasedatabase plaatsen, een interface waarmee gebruikers componenten uit de releasedatabase halen en een webserver voor op afstand toegang te krijgen tot de releasedatabase en de componenten.

Deze paper (van der Hoek & Wolf, 2002) stelt dus dat software release management ervoor moet zorgen dat componenten die op verschillende locaties, door verschillende bedrijven worden ontwikkeld, gemakkelijk gecentraliseerd toegankelijk moeten zijn. De bedrijven zelf hebben nog altijd volledige controle in handen over het versie beheer en het groeperen van de verschillende extern benodigde componenten. Tevens is de eindgebruiker nog altijd verantwoordelijk voor de installatie ervan maar dit begint ook minder het geval te zijn aangezien er meer release tools op de markt komen.

2.1.2 Uitdagingen en problemen in het Release Management Process: een Casestudie

Zoals eerder aangehaald spelen ITIL-achtige procedures een belangrijke rol in software release management. Deze paper (Lahtela & Jantti, 2011) is een korte casestudie met de bedoeling om kort een aantal problemen aan het licht te stellen in verband met software release binnen een bedrijf of organisatie en hiervoor een oplossing aan te bieden.

De casestudie (Lahtela & Jantti, 2011) beschrijft allereerst een definitie voor release management. “Release management omvat mensen, functies, systemen en activiteiten om software- en hardware versies effectief te plannen, verpakken, bouwen, testen en implementeren in een productie omgeving.” Lahtela en Jantti (2011) Deze definitie sluit goed aan bij wat dit onderzoek tracht te verduidelijken. Ook stelt de studie een belangrijk algemeen probleem. Helaas is in de praktijk bij veel bedrijven nog geen sprake van een implementatie van ITIL, ISO, enz. procedures. Dit omdat dit zeer moeilijk te implementeren is in reeds bestaande processen. Het hebben van zo een procedures is niet alleen zeer belangrijk om kwalitatief goede software op te leveren maar ook voor de supportafdeling van een bedrijf. Meestal zijn dit de mensen die het meeste te maken krijgen met deze procedures. Hierbij moet wel gezorgd worden dat er duidelijk verschil gemaakt wordt tussen het beheren van veranderingen en het release management. Vervolgens beschrijft de studie de vastgestelde problemen en een mogelijk oplossing ervoor. Deze zijn letterlijk overgenomen uit de studie Lahtela en Jantti (2011).

- Er is geen gespecificeerd releasebeheerproces.
 - Het releasebeheerproces moet worden beschreven en gestroomlijnd om ervoor te zorgen dat iedereen in de organisatie het proces kent.
- De rol van releasesmanager is onduidelijk.
 - Iemand moet worden genoemd voor de rol van releasesmanager. Daarnaast moeten de rollen, toewijzingen en verantwoordelijkheden worden beschreven.
- De klant weet niet wat de release bevat.
 - Meestal worden niet alle aangebrachte veranderingen beschreven of worden deze te technische beschreven waardoor de klant deze niet verstaat. De boodschap is om deze goed te documenteren op een verstaanbare manier.
- De uitgifte distributie snelheid is te hoog.
 - De release-vensters, die het tijdstip bepalen waarop de release in de productieomgeving van de klant moet worden geïnstalleerd, moeten worden overeengekomen tussen de serviceprovider en de klant, bijvoorbeeld grote releases maandelijks en kleinere releases wekelijks.
- De klant denkt dat de serviceprovider niet alle test gevallen kan testen of inspecteren.
 - Het testproces, dat deels wordt gedaan binnen het release managementproces, moet worden ontwikkeld. Het proces moet worden beschreven en aan producttesters worden geleerd. Daarnaast moeten de testresultaten aan de klant worden voorgelegd.
- Er moeten meer testomgevingen in het testproces zijn.
 - Er moet worden onderzocht of er meer testomgevingen nodig zijn. De ideale situatie is dat er voor elke productieomgeving een identieke testomgeving zou zijn.

- Het verandermanagement van testomgevingen is onvoldoende.
 - Elke wijziging die in een bepaalde testomgeving wordt aangebracht, moet correct worden gedocumenteerd. Op deze manier zijn alle wijzigingen traceerbaar en up-to-date.
- Problemen bij versiebeheer.
 - Alle versies van verschillende producten moeten worden gedocumenteerd in een klant specifieke lijst, die de serviceprovider vertelt welke geïnstalleerde productieversies de klant heeft.
- De caseorganisatie heeft geen specifieke 'release jury'.
 - Er is behoefte aan een specifieke jury die releases inspecteert voordat ze in productie worden genomen.

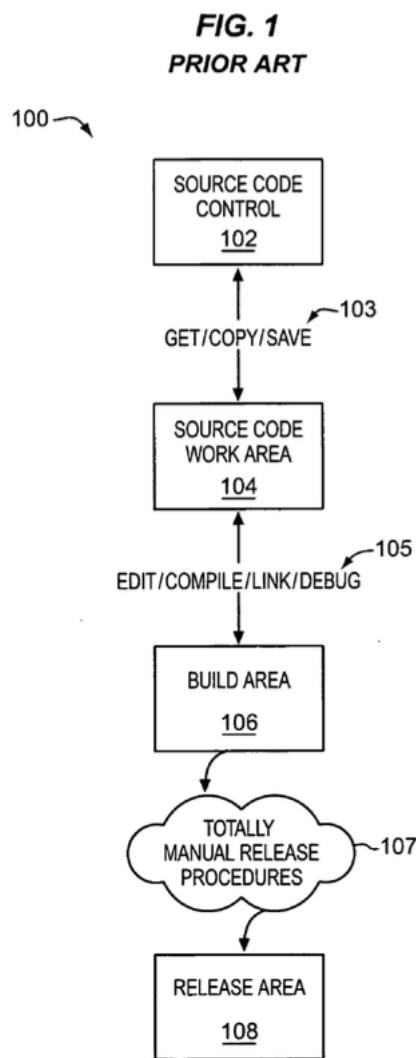
Deze studie (Lahtela & Jantti, 2011) biedt een unieke inkijk op een specifiek geval en biedt oplossingen aan voor bepaalde problemen. Sommige van deze problemen sluiten zeer goed aan bij dit onderzoek. Zo is er nood aan een zeer goede test omgeving zodanig dat er niet nodeloos moet worden gereden tussen klant en bedrijf.

2.1.3 Methodes en Systemen voor Software Release Management

Het volgende document (Methodes and systems for software release management, 2005) is een patent dat een bepaalde problematiek met normale software release management probeert op te lossen. Op zich is dit patent niet zo interessant voor dit onderzoek maar de figuren en ideeën waarvoor het bedrijf in kwestie een patent heeft aangevraagd, kunnen wel helpen in het beter begrijpen van software release management en hoe deze wordt toegepast.

Dit document (Methodes and systems for software release management, 2005) begint met zeer algemeen uit te leggen hoe software release management werkt en wat de verschillende stappen zijn die doorlopen worden. Het begint met versie beheer van de software. Waarna deze in een ontwikkelomgeving wordt gebracht. Hierna zal de software meestal naar een test omgeving gaan vooraleer het in productie wordt geplaatst. Al deze verschillende stappen zijn meestal verschillende mensen, in verschillende grote teams, die niets anders dan hun specifieke stap uitvoeren. *Zie diagram 2.2.* Het document (Methodes and systems for software release management, 2005) beschrijft echter een aantal problematieken bij het idee uit *figuur 2.2*. Zo is dit een zeer manueel proces waarbij verschillende mensen deelnemen. Het is dus zeer gemakkelijk om fouten te maken tijdens een van deze stappen. Bijvoorbeeld een merge conflict. Deze problemen hebben ze proberen verhelpen door middel van computer geassisteerde release. *Zie figuur 2 Methodes and systems for software release management (2005).* Ook is er in dit document nagedacht over de verschillende files en methodes om software te compileren en hoe deze moeten worden beheerd. Zo stellen ze gescheiden opslagplaatsen voor, voor inventory bestanden en compiler bestanden, enz. *Zie figuur 3 Methodes and systems for software release management (2005) voor details.*

Dit document (Methodes and systems for software release management, 2005) toont dus een nogal algemeen beeld over de software release cycli en wordt meer ter info beschouwd in dit onderzoek.



Figuur 2.2: Figuur uit (Methodes and systems for software release management, 2005). Diagram van een niet geautomatiseerde versie beheer procedure.

2.1.4 Continuous Integration en de Tools

Bij release management hoort continuous integration en continuous deployment (CI/CD). Het is namelijk een toepassing om op een snelle & kwalitatieve manier software te gaan ontwikkelen en uitrollen. Het volgende korte artikel (Meyer, 2014) bespreekt kort waarom een organisatie aan continuous integration moet doen en wat voor tools er allemaal bestaan. Ook worden er een aantal aanwijzingen gegeven rond het gebruik van continuous integration.

Het artikel (Meyer, 2014) vertrekt vanuit het standpunt dat een versie beheer tool in ieder softwarebedrijf een gegeven moet zijn. Dit om situaties te vermijden dat software lokaal werkt maar op andere toestellen niet. Ook staat dit toe om volledige geautomatiseerde pipelines te maken die automatisch de code compileert en controleert op fouten door meegeleverde testen uit te voeren of door een uitrol in een test omgeving. Het artikel gaat verder met te stellen dat het bij iedere organisatie een prioriteit moet zijn om ten aller tijden de opgeleverde code werkende te houden. Dit staat dan niet alleen toe dat er ten aller tijden een uitrol kan worden uitgevoerd van werkende code naar een test omgeving, maar ook dat er geen fouten of niet werkende code wordt gepusht door de programmeurs. Het artikel beschrijft verder een aantal tools zoals onder andere 'Jenkins' voor het automatisch uitrollen en testen van nieuwe software.

Dit artikel (Meyer, 2014) is minder belangrijk voor dit onderzoek maar toont wel het belang aan van een compiler pipeline en een goede test omgeving voor code werkende te houden en de kwaliteit te behouden.

2.1.5 DevOps

Devops is de gebruikte term om de samenwerking te beschrijven tussen softwareteams en hardwareteams binnen een organisatie of project. Deze term wordt meestal gebruikt binnen een release management of CI/CD werk sfeer. Met Devops is het de bedoeling om de verschillende teams onderling te doen communiceren en samenwerken. Dit met doel om beter en sneller software te ontwikkelen en op te leveren. Het volgende artikel (Ebert, Gallardo, Hernantes & Serrano, 2016) legt in meer detail Devops uit. De 2 luiken van Devops worden besproken alsook waarom Devops belangrijk is. Ook worden veel tools en technieken aangehaald.

Het artikel (Ebert e.a., 2016) begint met uit te leggen wat Devops voor een organisatie kan betekenen. Het stelt dat Devops staat voor een betere samenwerking tussen ontwikkelen, kwaliteit, zekerheid en operaties. Het is lang niet zo dat Devops voor ieder softwareproject kan worden gebruikt, maar het wordt toch aangeraden. Ook vertrekt het artikel uit een gegeven dat er al software versiebeheer in enige vorm aanwezig is.

Verder stelt het artikel (Ebert e.a., 2016) dat er 2 facetten zijn bij Devops. Enerzijds de compiler kant van de software en anderzijds het deployment gedeelte van de software. Compileren wordt volgens het artikel hoofdzakelijk op twee manieren uitgevoerd. Aan de ene kant zijn er de traditionele compiler tools die meestal nog wat handmatig werk

vereisen. Om in deze tools te compileren moet er meestal een XML-bestand worden gemaakt met specifieke onderdelen. Dit kan nogal arbeidsintensief zijn. Langs de andere kant zijn er continuous integration procedures. Hierbij wordt getracht om op ieder moment testbare, werkende code te hebben. Deze tools zijn een stuk gemakkelijker om te gebruiken en zijn meestal Cloud gebaseerd. Dit omdat het de bedoeling is om op een vlot en sneller tempo updates te kunnen uitrollen. Het artikel beschrijft een tabel Ebert e.a. (2016) waarin een aantal tools met wat rand info beschreven staan.

Het artikel (Ebert e.a., 2016) stelt dat meestal de software moet worden getest hierna, zodanig dat kwaliteit kan worden gegarandeerd. Dit moet zo efficiënt en snel mogelijk gebeuren waardoor er dus nood is aan 'infrastructure as code'. Herbruikbare code die snel kan worden uitgevoerd over meerdere platformen. Verschillende automatisatie tools worden door het artikel besproken. De meeste zijn simpel in gebruik en gebruiken YAML of XML voor de test omgevingen te definiëren. Het uitrollen van test omgevingen voor de software wordt meestal in de Cloud gedaan of met virtuele machines. Dit is afhankelijk van wat de vereisten zijn van de organisatie.

Ook bespreekt het artikel (Ebert e.a., 2016) een van de nieuwere manieren om software te testen. Ze bespreken het gebruik van microservices in de Cloud. Hoofdzakelijk bespreekt het artikel de producten van Amazon AWS. Deze zouden zeer gemakkelijk te automatiseren zijn en gemakkelijk te gebruiken.

Dit artikel (Ebert e.a., 2016) is een meerwaarde voor dit onderzoek omdat er een hele hoop tools, naast elkaar, kort worden besproken. De nadruk ligt niet op gebruiksvriendelijkheid maar eerder op wat de tools ondersteunen en of dat ze bruikbaar zijn voor Devops procedures. Dit artikel bevestigt dat er in de use case van dit onderzoek een nood is aan een goed gebouwde omgeving omdat kwaliteit een nummer één prioriteit geworden is.

2.1.6 Continuous Integration en Release Pipelines bouwen door het toepassen DevOps Principles: een Casestudie bij Varidesk

Zoals de vorige artikelen aantonen is het gebruik van CI/CD binnen software release management aangeraden. Er zijn reeds een aantal problemen en moeilijkheden beschreven. Het volgende artikel (Debroy, Miller & Brimble, 2018) is een studie over een specifieke use case van een bedrijf dat bepaalde web services aanbiedt aan klanten. Het beschrijft hoe de ervaring was om een CI/CD pipeline te migreren naar de Cloud. Welke valkuilen er zijn. Welke moeilijkheden er zijn vastgesteld.

De casestudie (Debroy e.a., 2018) begint met de nood voor CI/CD uit te leggen. Het beschrijft dat er verschillende voordelen zijn aan CI/CD. Een voordeel is dat er veel sneller kan worden gereageerd op support vragen en mogelijke problemen of updates. Ook beschrijft het artikel dat er in sommige gevallen, zoals bij 'HP Laserjet Firmware Division', een kost reductie tot 78 procent is. Vervolgens verklaart de studie dat ondanks CI/CD een wijd verspreide procedure is, er nog weinig onderzoek naar is gebeurd. Ook beschrijft het artikel dat tool ondersteuning niet al te best is.

Vervolgens beschrijft het artikel (Debroy e.a., 2018) de oorspronkelijke werkwijze van het bedrijf. Het bedrijf gebruikte voor release en compilatie van hun volledige web service 'Visual Studio Team Service' (VSTS). Dit is een betalend platform van Microsoft. Het platform heeft een marktplaats waarop voor gedefinieerde stappen staan om bepaalde taken te voltooien. Bijvoorbeeld compileren, automatische testen, kwaliteit stappen, enz. Dit voor allerlei soorten projecten en programmeertalen. Ook bestaan er stappen voor automatische uitrol met behulp van bepaalde bestaande tools. Ook bestaat er een mogelijkheid dat de klanten die gebruik maken van het platform, zelf bepaalde taken definiëren en dan uploaden op de marktplaats.

Het artikel (Debroy e.a., 2018) beschrijft dat dit voor de totale website meer dan voldoende is. Al werd er volgens het artikel een aantal beperkingen vastgesteld. Zo is de compilatietijd op dit platform niet schitterend. Het duurt vaak lang. Ook is dit een Cloud specifieke oplossing waardoor dit niet schaalbaar is over verschillende Cloud platformen. Daarboven op is het ook nog eens niet schaalbaar op het platform zelf. Dit heeft het artikel proberen oplossen door meerdere instanties te doen draaien op hetzelfde platform maar de resultaten vielen tegen.

Daarna beschrijft het artikel (Debroy e.a., 2018) de gevonden oplossing. Er is een architecturale verandering gebeurd aan de site waardoor er met componenten wordt gewerkt. Dit heeft een nood ontwikkeld voor meerdere pipelines tegelijk. Deze oplossing moet dan ook nog eens redundant zijn door het te verspreiden over verschillende platformen. Een oplossing hiervoor zouden containers kunnen zijn. Dit is een redelijk gemakkelijk te implementeren oplossing. Alleen viel de schaalbaarheid op een hetzelfde platform tegen waardoor de compileertijden niet veel verbeterden. Dus al snel werd er gekeken naar microservices. Dit is gemakkelijk te configureren en extreem schaalbaar op de Cloud platformen. Het enige probleem is het uploaden naar de Cloud is bij ieder platform anders en zeer manueel. Als oplossing hiervoor is een script ontwikkeld dat gemakkelijk aanpasbaar is voor naar de verschillende Cloud platformen te uploaden. Dit heeft, zoals in het artikel aangegeven Debroy e.a. (2018), de compilatietijden drastisch verlaagd.

Deze studie (Debroy e.a., 2018) sluit zeer nauw aan bij dit onderzoek. Er wordt beschreven wat mogelijke valkuilen kunnen zijn en waar de problemen liggen. Het toont ook aan dat zo goed als ieder Cloud platform kan worden gebruikt voor CI/CD door het gebruik van containers. Al is dit niet altijd de meest efficiënte oplossing. Ondanks dat dit artikel al een groot deel onderzocht heeft, is er nog altijd nood om te onderzoeken hoe de verschillende platformen CI/CD mogelijk maken op een niet containerized manier.

2.2 Amazon AWS

2.2.1 Prestatie Analyse van hoge prestatie reken applicaties op Amazon Web Services Cloud

Dit onderzoek wil ook de Cloud platformen en hun aanbod naast elkaar leggen voor vergelijking. Ook wilt dit onderzoek de lezer een idee geven over hoe de verschillende

datacenters van de Cloud platformen samengesteld zijn, op hardware vlak, en hoe de verbindingen hiermee zijn. De volgende paper (Jackson e.a., 2010) is een prestatie test van een Amazon AWS datacenter.

De paper (Jackson e.a., 2010) gebruikt synthetische wetenschappelijke testen om de prestatie van een aantal Amazon producten te testen. Een Amazon datacenter is wat raar samengesteld aangezien de eindgebruiker geen enkel idee op voorhand heeft over welke hardware hij krijgt toegewezen in het datacenter (Op het moment van schrijven van (Jackson e.a., 2010)). Bovendien is Amazon benadeelt op vlak van connectiviteit omdat het niet directe lijnen heeft naar de datacenters zoals Microsoft met Azure. Belangrijk is dat alle testen uitgevoerd door de paper, zijn uitgevoerd in geografisch hetzelfde datacenter.

In deze paper (Jackson e.a., 2010) testen de onderzoekers vooral of dat een Amazon datacenter geschikt zou zijn voor wetenschappelijk gebruik. Vandaar dat er vooral synthetische wetenschappelijke tools worden gebruikt. De tools zijn zorgvuldig geselecteerd zodanig dat de verschillende aspecten van een datacenter worden getest. Zowel de processor, als RAM, de harde schijven en ook de verbinding met de servers worden getest.

De paper (Jackson e.a., 2010) concludeert dat de prestatie van de systemen wel goed is maar dat dit afhankelijk is, van welk merk CPU de gebruiker krijgt. Aangezien hier geen controle over is, is het moeilijk om voor dezen redenen Amazon voor rekenintensieve doeleinden aan te raden. Ook is vastgesteld dat de connectie met het datacenter een aanzienlijke beperking is voor taken die zeer transactie intensief zijn.

Deze paper (Jackson e.a., 2010) is niet zo zeer een meerwaarde voor dit onderzoek. Het artikel is redelijk verouderd en ondertussen is Amazon een van de grotere Cloud platformen. Ook bespreekt deze paper niet zo goed de producten van Amazon. Ook is de use case die de paper beschrijft totaal verschillend van de use case die dit onderzoek gebruikt.

2.3 Microsoft Azure

2.3.1 Microsoft Azure en Cloud computing

Aangezien binnen de use case van dit onderzoek Microsoft Azure een grote rol speelt, is het belangrijk om een zeer goed idee te vromen van hoe dit Cloud platform is opgebouwd en wat de verschillende services en producten zijn. Daarom is het volgende hoofdstuk uit het boek (Copeland, Soh, Puca, Manning & Gollob, 2015) redelijk informatief, zeker als een initiatie.

Het eerste hoofdstuk uit het boek (Copeland e.a., 2015) is vooral bedoelt als een eerste kennismaking met het Azure platform. Azure is eigenlijk ontstaan uit Microsoft hun office 365 aanbiedingen. Microsoft biedt een grote hoeveelheid aan applicaties aan als onderdeel van dit pakket. Omdat die applicaties ergens moeten draaien, is Microsoft begonnen met het bouwen van datacenters om daar hun SaaS (Software as a Service) in te hosten. Al snel beseften ze dat er geld viel te verdienen met het aanbieden van remote services. Het duurde dan ook niet lang vooraleer Microsoft ook begon met IaaS (Infrastructure

as a Service) aan te bieden. Dit was toen een unicum volgens het boek (Copeland e.a., 2015), Aangezien tot dan de meeste Cloud platformen ontstaan zijn vanuit het verhuren van overschot aan rekenkracht uit een mengelmoes van toestellen uit een bepaald datacenter. Dit was onder andere een van de conclusies van een verouderd artikel over Amazon aws (Jackson e.a., 2010). Bij Microsoft waren dat speciaal gebouwde datacenters met die services in gedachten. Ook hun PaaS (Platform as a service) wordt kort aangehaald door het boek (Copeland e.a., 2015).

Voorbeelden van Azure IaaS:

- Azure virtual machines
- Azure virtual networks
- Azure virtual networks gateways
- Azure storage solutions
- ...

Voorbeelden van Azure Paas:

- Azure SQL database
- Azure website
- Azure content delivery network
- Azure DevOps
- ...

Verder legt het boek (Copeland e.a., 2015) kort uit wat vanuit Azure wordt gedaan op vlak van privacy en wetgevingen. Dit is minder interessant voor dit onderzoek. Ook gaat het boek kort in op waarom It professionals voor Azure Cloud of een ander Cloud platform zouden moeten kiezen. Zo haalt het boek (Copeland e.a., 2015) aan dat een Cloud platform weinig onderhoud inhoud, minder dan een lokale opstelling. Ook is de eindgebruiker niet verantwoordelijk voor de hardware. Hun datacenter zijn redundant. Dus er is eigenlijk vrij weinig down time. Ook de aangeboden producten zijn vrij compleet en gemakkelijk onderhoudbaar.

Verder geeft het boek (Copeland e.a., 2015) een korte introductie in het Azure web portaal. Deze heeft dit onderzoek voor kennismaking doeleinden doorgelopen. Veel interessants is hier voor dit onderzoek niet uit te melden. Het hoofdstuk is dus een snelle kennismaking met Azure. Dit dient als een basis voor verder onderzoek. Zo gaan we in deze literatuurstudie nog wat dieper in op Azure DevOps en een kleine hands-on. Ook IaaS van Azure wordt nog wat meer uitgediept.

2.3.2 Microsoft Azure Documentation

Zoals eerder aangehaald in deze literatuurstudie, speelt Microsoft Azure een belangrijke rol binnen de use case van dit onderzoek. Het is het vertrek punt voor de vergelijkingen. In het kader van dit onderzoek, is de website van Azure eens uitgeplozen. Dit omdat er een beeld kan gevormd worden over welke Azure services interessant kunnen zijn. Het volgende is een kort verslag. Er worden twee service categorieën aangehaald. Deze zijn

IaaS en PaaS (Infrastructure as a Services en Platform as a Service). Specifiek is er gekeken naar Azure virtual machine, Azure virtual networks, Azure virtual gateways en Azure DevOps. Het laatste is een samenvatting van een onlinecursus en informatie op Microsoft Docs.

In het kader van software release management en dan vooral de stap om de kwaliteit van software te controleren, is er gekeken of er misschien een mogelijkheid is om deze specifieke infrastructuur eventueel in de Cloud te maken. Hiervoor is er een kort concept uitgewerkt. Dit concept beschrijft een hybride infrastructuur waarbij eigenlijk alle niet use case specifieke toestellen in de Cloud zitten. In theorie stond dit toe dat alle infrastructuur dan als code zou kunnen worden gedefinieerd.

Het aanbod qua mogelijkheden voor hardware om een virtuele machine aan te maken op Azure is enorm. Ook in tegenstelling tot de concurrenten wordt er zeer transparant omgesprongen met welke hardware er per optie beschikbaar wordt gesteld. De mogelijkheden verschillen enorm en zijn meestal voor specifieke doeleinden. Ook de prijzen schalen mede met de use cases. Zo zijn er specifieke opties voor machine learning. Of een optie voor machines met enorme hoeveelheden ram en CPU-kracht om met bepaalde databases overweg te kunnen. Ook de goedkopere basis opties zijn redelijk uitgebreid. Al deze opties worden door Azure onderverdeelt in categorieën die door middel van een letter worden aangeduid. Zie figuur 2.3. Verder is Azure de oudere manier van het definiëren



The chart is a comparison table for Azure VM types, showing categories: General Purpose, Compute Optimized, Memory Optimized, Storage Optimized, GPU, and High Performance Compute. It includes columns for Type, Description, and Uses.

	General Purpose	Compute Optimized	Memory Optimized	Storage Optimized	GPU	High Performance Compute
Type	DC, Av2, Dv2, Dv3, B, Dsv3	Fsv2, F	M, Dv2, G, DSv2, GS, Ev3	Ls	NC, NCv2, ND, BV, NVv2	H
Description	Balanced CPU and memory	High ratio of compute to memory	High ratio of memory to compute	High disk throughput and IO	Specialized with single or multiple NVIDIA GPUs	High memory and compute power – fastest and most powerful
Uses	Testing and dev, small-med databases, low traffic web servers	Medium traffic web servers, network appliances, batch processing, app servers	Relational database services, analytics, and larger caches	Big Data, SQL, NoSQL databases	Compute intensive, graphics-intensive, and visualization workloads	Batch processing, analytics, molecular modeling, and fluid dynamics, low latency RDMA networking



Figuur 2.3: Figuur van (<https://p2zk82o7hr3yb6ge7gzxx4ki-wpengine.netdna-ssl.com/wp-content/uploads/azure-vm-types-comparison-1.jpg>). Chart met alle VM tiers van azure en hun specifieke code.

van een Azure virtual machine aan het afraden. Hiernaast zijn er ook nog een tal van mogelijkheden op vlak van virtual networking. Het belangrijkste is dat er een mogelijkheid bestaat om een virtueel netwerk volledig te isoleren van het internet. Dit staat toe dat enkel verkeer dat op dat specifieke netwerk is, aan de virtuele machines kan. Er wordt

dan wel meestal een Azure network firewall node voorzien die in dit virtueel netwerk zit, zodanig dat de connectiviteit met de machines ten aller tijden kan worden gegarandeerd. Om dan connectiviteit te hebben met het virtueel netwerk wordt er gebruikgemaakt van een Azure Gateway node. Deze staat een point to point VPN (Virtual Private Network) tunnel toe. Dit is eigenlijk een directe, geëncrypteerde verbinding met het Azure netwerk. Hiervoor is wel specifieke hardware nodig lokaal in het netwerk. De kost van deze services is eigenlijk bijna niks. Azure werkt immers met een pay-to-run, pay-on-the-go systeem. Dit wil zeggen dat er dus moet worden betaald voor de aanmaak van de services en daarna voor het verbruik. Microsoft heeft op zijn documentatie portaal tal van stap voor stap handleiding voor het instellen van deze services. Ook concepten voor hybride opstellingen staan hier uitgelegd. Maar dit zou deze literatuurstudie te veel doen afwijken.

Het zou dus in theorie mogelijk moeten zijn om een hybride Cloud infrastructuur te voorzien die dynamisch is voor de specifieke te testen projecten. De vraag is of dit handig is in gebruik en niet noodloos complexiteit toevoegt.

Azure Devops is een platform van Azure dat organisaties toestaat om bepaalde procedures te definiëren voor het uitrollen van projecten. Het platform staat dan ook integratie toe met andere project follow-up tools van Microsoft. Deze procedures kunnen juist zoals de virtuele machines op Azure via code worden gedefinieerd of via een duidelijk en gemakkelijk te gebruiken GUI. De bedoeling van Azure DevOps is eigenlijk om het oude TFS-systeem te vervangen en een verbeterde interface aan te bieden. Meestal wordt DevOps gebruikt om aan Continuous Integration te doen. Dus er wordt een repository waar de code opstaat gedefinieerd. Hierna bestaat er een mogelijkheid om de code te compileren, automatisch testen te runnen en deze code dan weer door te schuiven naar een volgend stadium. Hier wordt de code dan uitgerold naar een omgeving voor verder kwaliteitscontrole. Het mooie aan dit platform is dat de gebruiker of organisatie niet verplicht is om deze code in een virtuele machine in de Cloud te compileren of testen. Er is volledige controle over de procedures.

In het kader van dit onderzoek is dit zeer belangrijk aangezien dit platform op dit moment in gebruik genomen wordt. Er is ook een onlinecursus gevolgd met een basis uitleg en een labo. Dit heeft duidelijk gemaakt dat het mogelijk is om vanuit DevOps op een lokale test omgeving uit te rollen.

3. Methodologie

In dit hoofdstuk wordt besproken hoe dit onderzoek zal verlopen. Wat de criteria zijn die beslissend zijn om een kandidaat te selecteren. Wat het uiteindelijke doel is van de 'Proof Of Concept' met deze kandidaat. Wat de criteria zijn voor een succesvolle 'Proof Of Concept'.

In een eerste fase moet er een alternatief Cloud platform worden geselecteerd voor de 'Proof Of Concept'. Deze selectie gebeurt op drie manieren. Eerst wordt er per Cloud platform bekeken wat de algemene mogelijkheden zijn op het platform. Er wordt gekeken naar de standaard mogelijkheden op ieder Cloud platform zoals: virtuele machines, opslag in de Cloud, enz. Dit wordt gedaan om een beeld te kunnen vormen over: hoe complex het platform is, hoe groot het Cloud platform is en hoe vergelijkt het platform in het algemeen. Hierna worden de mogelijkheden voor het versiebeheersysteem besproken. Dit kan belangrijk zijn. Als een Cloud platform de meest populaire systemen niet ondersteund, zou het in de toekomst problemen kunnen geven. Ook zoekt dit onderzoek naar een flexibel platform. Tot slot worden alle mogelijkheden voor CI/CD besproken en vergeleken. Dit is de belangrijkste criteria voor het selecteren van een kandidaat. Als een kandidaat geen ingebouwde oplossingen heeft, dan valt deze buiten beschouwing.

In een tweede fase wordt er bij de kandidaten die in aanmerking komen, een test op gebruiksvriendelijkheid uitgevoerd. Voor deze testen worden er handleidingen gevuld van de Cloud platformen zelf om een Java applicatie te compileren. Het meten van gebruiksvriendelijkheid gebeurt door middel van tijd. Hoe meer tijd nodig was om op een platform een pipeline te configureren hoe minder gebruiksvriendelijk het platform is. Ook zijn deze testen pas succesvol wanneer de applicatie op een lokale computer kan worden uitgevoerd en de meegeleverde testen zijn uitgevoerd. Ook wordt bij ieder platform opnieuw begonnen met het schrijven van de applicatie. Deze applicatie is hetzelfde voor

alle testen.

In de laatste fase wordt een 'Proof Of Concept' gemaakt met de geselecteerde kandidaat. Deze opstelling moet een .Net console applicatie compileren en testen. Hierna moet deze applicatie naar een lokale omgeving worden gekopieerd. Deze applicatie moet dan worden uitgevoerd. Deze opstelling is pas volledig wanneer deze functionaliteit bereikt is. Deze 'Proof Of Concept' wordt ook nagemaakt in Azure DevOps om een duidelijke conclusie te kunnen maken. Hiervoor zal rekening worden gehouden met de voorgaande fases.

4. Kandidaat Selectie

Het is nodig om een aantal Cloud platformen hun aanbod te bespreken en te vergelijken. Omdat het uiteindelijke doel van dit onderzoek een Proof Of Concept met het best mogelijke alternatief voor Azure DevOps is. In dit hoofdstuk worden alle mogelijkheden van de Cloud platformen kort besproken en vergeleken. Ook worden de mogelijkheden voor CI/CD vergeleken. Daarnaast wordt er in dit hoofdstuk kort besproken hoe het prijsenstelsel werkt. Bij elk Cloud platform wordt er verantwoord waarom dit platform een goede kandidaat zou zijn of waarom niet. Ook wordt er een test uitgevoerd op gebruiksvriendelijkheid. Hiervoor zal een Java applicatie gebruikt worden. Op het einde van dit hoofdstuk wordt er een sterke kandidaat naar voren geschoven. Niet alle Cloud platformen zullen aan bod komen. Daarvoor zijn er teveel mogelijkheden. Dit hoofdstuk vergelijkt de meest populaire Cloud platformen.

4.1 Azure DevOps

Microsoft Azure is gelanceerd in 2010 en bevat een hele serie producten. Azure biedt vooral producten aan in de categorieën 'Software as a Service' (SaaS), 'Infrastructure as a Service' (IaaS) & 'Platform as a Service' (PaaS). Azure heeft een aantal goede producten voor virtualisatie. Zo heeft Azure veel mogelijkheden voor verschillende soorten machines voor verschillende doeleinden. Ook hun virtuele netwerk mogelijkheden zijn enorm. Dit alles is mooi geordend en gemakkelijk in gebruik. De Azure datacenters zijn over heel de wereld verspreid. Deze zijn altijd het nieuwste van het nieuwste. Ook zijn de datacenters goed verbonden onderling en met de buitenwereld. Omdat Azure van Microsoft is, is Azure ook perfect te integreren met gebruikersaccounts in domeinen die reeds bestaan. Dit geeft de gebruiker volledige controle over wie, wat kan gebruiken en zien. Azure biedt

ook een aantal services aan. Daarvan is Cloud gebaseerde 'Active Directory' er een van. Ook bieden ze een volledig aanbod aan services aan om CI/CD te realiseren.

Azure DevOps was vroeger bekend als 'Visual Studio Team System' (VSTS) of 'Team Foundation Server' (TFS). Het is een versie beheer, rapportering, vereisten beheer, project beheer, automatisch compileer, test en uitrol tool gemaakt door Microsoft. De tool maakt gebruik van 'Team Foundation Version Control' (TFVC) of Git. De tool is gemaakt om de volledige levenscyclus van een programma te controleren en beheren. Ook biedt de tool de mogelijkheid aan programmeerteams om in een DevOps sfeer samen te werken. Het mooie aan deze tool is dat het bijna in iedere 'Integrated Development Environment' (IDE) te integreren is.

Het is mogelijk om deze tool zowel lokaal als in de Cloud te implementeren. Microsoft heeft deze tool toegevoegd aan hun Azure aanbod onder Azure DevOps. Microsoft heeft de verschillende componenten van deze tool opgesplitst op het platform. Dit maakt het mogelijk dat de gebruiker niet alle componenten tegelijk hoeft te gebruiken of te implementeren. De gebruiker kan zo naar voorkeur functies kiezen.

Azure DevOps kan gebruikmaken van twee soorten versie controle in een project. Het kan gebruikmaken van de door Microsoft speciaal ontwikkelde versie beheer framework TFVC voor Azure DevOps of het wereld befaamde Git.

TFVC ondersteund twee manieren van werken, met een centraal systeem of lokaal met check-out/ check-in op de computer van een programmeur. Bij het gebruik van een centraal systeem worden bestanden die door een andere programmeur worden gebruikt als 'alleen lezen' bestempeld. Dit kan leiden tot problemen als andere programmeurs deze bestanden nodig hebben. Dit heeft Microsoft proberen oplossen door het mogelijk te maken om volledig lokaal te werken. De programmeur kan dan alle bestanden aanpassen waar nodig. Eventuele problemen met verschillende bestanden moeten dan worden opgelost bij check-in. Dit maakt het mogelijk dat er veel minder conflicten ontstaan. Een ander voordeel is dat de gebruiker de mogelijkheid heeft om met TFVC, regels te configureren die bij check-in worden uitgevoerd.

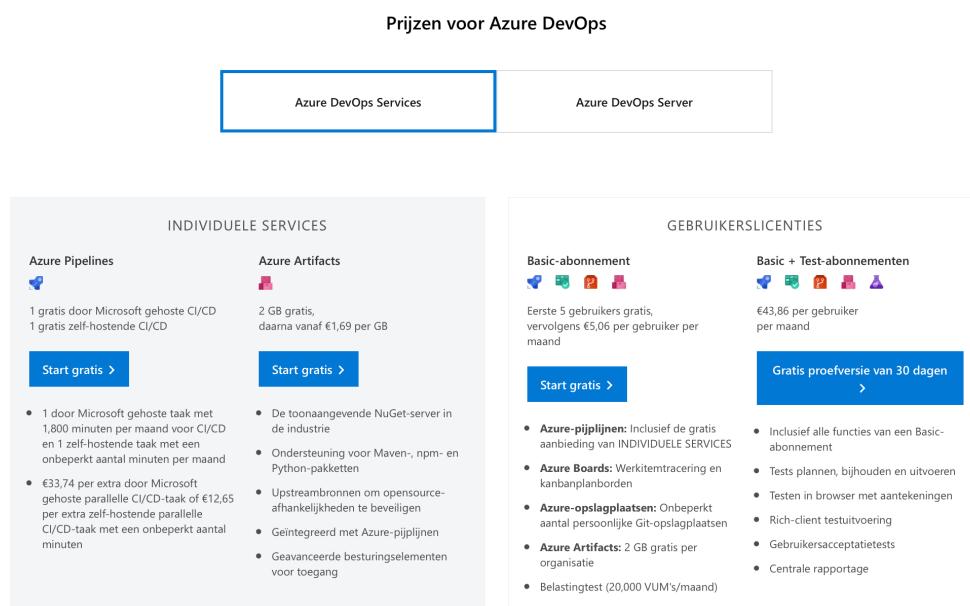
Git is een veel gebruikt versiebeheersysteem. Bijna alle IDE's bieden ondersteuning aan voor dit systeem. Het werkt gelijkaardig zoals TFVC. Alleen kan de gebruiker met Git geen regels configureren die worden uitgevoerd bij check-in. Git is wel volledig compatibel met Azure DevOps. Zo kan er rechtstreeks met Git op Azure DevOps worden gepubliceerd. Dit alles zorgt ervoor dat gebruikers door gebruik van Git, met bijna iedere IDE of programmeertaal, Azure Devops kan gebruiken.

Een ander voordeel van Azure DevOps is de uitgebreide rapportering ingebouwd in de tool. Deze maakt het mogelijk om uitgebreide verslagen te genereren van de uitgevoerde testen, een uitgevoerde check-in, compileer problemen, enz. Ook staat het de gebruiker toe om gepaste meldingen te versturen of automatisatie te configureren om bepaalde problemen op te lossen. Daarnaast heeft Azure Devops ook een ingebouwde tool om planningen voor projecten bij te houden.

Azure Boards is een volledige implementatie rechtstreeks in Azure DevOps van een scrum-

bord. Zo kan de gebruiker bij het opleveren van een uitgevoerde taak automatisch de compileer pipeline doen starten. Ook is het volledig geïntegreerd met GitHub waardoor de gebruiker geen extra kosten moet maken om een Azure Repository aan te maken. Ook bestaat de mogelijkheid om Azure boards naadloos te laten samenwerken met de populairste chat applicaties voor ontwikkelaars, zoals Slack.

Als een gebruiker een CI pipeline wil configureren in Azure DevOps, dan kan de gebruiker dat via de web interface doen of met de 'Azure powershell' module. Azure DevOps gebruikt zoals eerder vermeld, twee soorten versiebeheersystemen. Om met Git een pipeline te bouwen, selecteert de gebruiker als bron een GitHub repository. Daarna kan de gebruiker kiezen uit een hele serie voor gemaakte engines voor het compileren van de code. Ook door gebruik te maken van een marktplaats kunnen er zelfs volledig aangepaste engines gebruikt worden. Hierna wordt de gebruiker gevraagd om een 'azure-pipelines.yml' aan te maken. Dit bestand wordt opgenomen in de broncode. In dit bestand staat de configuratie geschreven voor de uit te voeren compilatie. De gebruiker kan dan nog stappen toevoegen voor testen uit te voeren. Azure DevOps heeft de mogelijkheid om op verschillende manieren de software uit te rollen. Gaande van in de Cloud tot specifieke lokale omgevingen. Azure Devops prijzen worden beschreven in *figuur 4.1*.



Figuur 4.1: Prijzen van Azure DevOps. Afbeelding van Azure DevOps website. Figuur toont de prijzen voor Azure DevOps op een beknopte manier.

4.2 Google Cloud

Google Cloud Platform (GCP) is gelanceerd in April 2008. Het draaide des tijds in dezelfde datacenters als 'google search', 'youtube' en 'gmail'. Het was slechts in 2011 dat GCP beschikbaar was voor het brede publiek. GCP is een onderdeel van de Google Cloud. Google Cloud biedt een enorme serie van producten aan waarvan GCP maar een

klein onderdeel is. GCP specifiek, biedt ‘infrastructure as a service’ (IaaS), ‘platform as a service’ (PaaS) en ‘serverless computing’ aan.

Omdat het doel van dit onderzoek specifiek de ondersteuning voor CI/CD vergelijken is, wordt er hier een specifiek onderdeel van GCP bekeken. De ‘Cloud Developper Tools’. Onder deze categorie vallen er een aantal zeer interessante hulpmiddelen. Hier vindt men onder andere ‘Cloud Build’, ‘Cloud-SDK’, ‘Tools voor Powershell’, ‘Tools voor Visual Studio’, enz.

Cloud Build is Google zijn antwoord op een volledige geautomatiseerde CI/CD pipeline. Het is dan ook volledig mede met de moderne vereisten. Met Google Cloud Build (GCB) is een organisatie instaat om snel en gemakkelijk een volledige pijpleiding te configureren. Het gelijkt dan ook op Azure DevOps.

GCB werkt hoofdzakelijk met Git en GitHub om een pijpleiding te bouwen. Google heeft ook zijn eigen ‘Cloud repository service’ die naadloos integreert met GCB, maar deze is helaas betalend. Daarom is het gebruik van Git met GitHub een beter en goedkoper alternatief. Aangezien deze ook perfect integreren met GCB en omdat Git wijdverspreid en simpel in gebruik is. Een organisatie kan dan configureren op GCB dat op het moment van een code update, automatisch een compiler pipeline wordt gestart. Om GCB te laten weten wat er specifiek moet worden uitgevoerd, moet er op de GitHub repository een YAML-bestand voorzien worden waarin regels moeten worden gedefinieerd. Dit maakt het gemakkelijk om snel aanpassingen te maken.

De pipeline op GCB werkt op basis van Docker images. Deze worden in de ‘cloud-build.yml’ gedefinieerd. Ook wordt er per Docker container gedefinieerd wat er moet worden uitgevoerd in de vorm van commando’s, script, enz. Dit maakt het mogelijk dat iedere stap in het CI gedeelte van de pipeline volledig kan worden aangepast naar de noden van de organisatie. Zo kan de organisatie beslissen om voor gemaakte containers te gebruiken van de ‘DockerHub’ pagina. Ook kan de organisatie zelf containers maken met aangepaste scripts om bijvoorbeeld in lokale omgevingen testen uit te voeren. GCB kan dus in een hybride opstelling geïmplementeerd worden. Wat ook de bedoeling zou zijn aangezien dit de use case van Aucxis is. Daarnaast biedt GCB ook de mogelijkheid om de gecompileerde code rechtstreeks vanuit Google Cloud beschikbaar te maken voor verdere verdeling.

Met behulp van deze containers kunnen dan functionele testen worden uitgevoerd op de gecompileerde code. Het programma kan dan op basis van de uitkomst van deze uitgevoerde taak, naar de volgende stap zijn wachtrij worden geplaatst. Hier kan de volgende taak starten. GCB genereert rapporten en statistieken van de uitgevoerde taken zodanig dat de gebruiker van het platform inzicht kan krijgen in de uitgevoerde taken.

De compiler en uitvoer-tijden van GCB zijn zeer goed aangezien het platform automatisch schaalt naarmate er meer rekwesten tegelijk worden verstuurd. Dit maakt mogelijk dat verschillende programmeurs tegelijk aan hetzelfde project werken of aan meerdere projecten tegelijk. Ook biedt GCB de mogelijkheid om redundantie te voorzien. GCB maakt het mogelijk om naar andere Cloud platformen uit te rollen of zelfs om de werklast te

verdelen over verschillende Cloud platformen. Dit is mogelijk door gebruik te maken van Tekton. Tekton is een open-source framework voor Kubernetes. Dit maakt het mogelijk dat een organisatie over verschillende Cloud platformen heen kan werken. Kubernetes is een clustering hypervisor voor Docker containers. Tekton zou een goede oplossing kunnen zijn voor CI/CD in de Cloud maar valt hier buiten beschouwen vermits het doel de verschillende aanbiedingen van de Cloud platformen vergelijken is.

De prijzen in Google Cloud worden berekend zoals bij ieder moderne Cloud aanbieder. De gebruiker betaald wat hij verbruikt. De volgende *figuur 4.2* moet dienen om een beeld te vormen over wat men kan verwachten te betalen voor het gebruik van GCB. Dit zonder netwerk kosten voor het transfereren van gegevens. Er wordt ook niks in rekening gebracht voor zaken die in een wachtrij staan of pipelines die niet worden gebruikt. De Google Cloud Developper Tools hebben een voordeel. Een groot deel van de hulpmiddelen voor ondersteuning met het platform zijn gratis. Er zijn ook weinig hulpmiddelen van derden nodig om de gewilde functionaliteit te bereiken.

Type machine	Virtuele CPU's	Snelle start ¹	Prijs (USD)
n1-standard-1	1	✓	\$ 0,003 / buildminuut. Eerste 120 buildminuten per dag zijn gratis. ²
n1-highcpu-8	8		\$ 0,016 / buildminuut
n1-highcpu-32	32		\$ 0,064 / buildminuut

¹ Een build met snelle start begint zonder leveringsvertraging.

² De aanbieding van 120 gratis buildminuten per dag geldt per factureringsaccount en kan worden gewijzigd.

Als u in een andere valuta dan USD betaalt, gelden de prijzen die in uw valuta op [Cloud Platform SKU's](#) worden vermeld.

[BuildOptions](#) ondersteunt een `diskSizeGb`-veld waarmee u extra SSD's tot maximaal 1000 GB kunt aanvragen boven de standaard 100 GB. De prijs voor extra SSD's is **\$ 0,17** per GB per maand. SSD-kosten worden op secondebasis berekend.

Figuur 4.2: GCB prijzen. Figuur van Google Cloud website. Figuur toont de prijzen voor Google Cloud Platform op een beknopte manier.

Met andere woorden is GCB dus een volwaardig alternatief voor Azure DevOps. Er is ondersteuning voor verschillende tools, biedt uitgebreide functionaliteiten aan en bovendien adverteert Google dat het gemakkelijk samenwerkt met andere Cloud platformen. Het zei door het gebruik van Tekton. Andere Cloud platformen ondersteunen dit ongetwijfeld ook. Alleen wordt er niet mee reclame gemaakt zoals bij GCP. Daarnaast is het ook relatief simpel in gebruik door dat GCB gebruikmaakt van Docker containers voor te compileren.

4.3 Amazon Web Services

Amazon web services (AWS) is gelanceerd in 2006. Gedurende lange tijd heeft Amazon stukken van hun datacenter verhuurd aan het brede publiek. Tegenwoordig kan een gebruiker op AWS een Cloud computer samenstellen juist zoals een gewone server zou worden samengesteld. Het is maar in recente tijden dat Amazon zich meer beginnen focussen is op services voor de gebruiker. Zo bevat AWS nu meer dan 212 services

en producten. Bijvoorbeeld: virtuele computerkracht, netwerking, opslag in de Cloud, databases, statistieken, programma services, uitrol op de Cloud, beheer van bepaalde zaken, ontwikkelingshulpmiddelen en hulpmiddelen voor 'Internet Of Things' (IoT). De populairste tegenwoordig zijn 'Amazon Elastic Compute Cloud' (EC2) en 'Amazon Simple Storage Service' (Amazon S3). Deze laatste zijn in feite Cloud computerkracht en opslag voor verschillende doeleinden. Deze zijn volledig schaalbaar en zijn gratis om te creëren.

Ondanks dit groot aanbod resteert er toch nog altijd de vraag hoe het zit met de huidige prestatie van Amazon datacenters over de hele wereld. Zeker na het lezen van deze paper (Jackson e.a., 2010). Voor dit onderzoek is er ijverig gezocht naar recentere prestatie onderzoeken maar zonder resultaat. Er kan alleen maar worden afgegaan van Amazon zijn website.

Al deze producten en services maken het niet gemakkelijk voor een gebruiker om snel te weten welke producten voor hem geschikt zijn. Ook in dit onderzoek is er vastgesteld dat het lastig is om een duidelijk beeld te krijgen wat er allemaal mogelijk is. Dat terzijde, heeft Amazon toch een specifiek aanbod om CI/CD te implementeren op hun Cloud platform. Zo heeft Amazon, AWS CodePipeline. Dit is een service waarbij de gebruiker of organisatie via het web portaal gemakkelijk een CI/CD pijpleiding kan definiëren. Deze is volledig aanpasbaar naar de noden van de gebruiker. AWS CodePipeline gebruikt AWS CodeBuild voor de compilatie en het testen van projecten in CI/CD en AWS CodeDeploy voor het automatische uitrollen van projecten.

Zoals alle grote Cloud platformen ondersteund AWS CodePipeline ook het gebruik van Git en GitHub. De gebruiker hoeft dus geen speciale zaken te doen. Amazon heeft ook zijn eigen Cloud repositories voor code op te bewaren. Deze zijn ook gebaseerd op Git. Het zijn eigenlijk privé Git repositories die door Amazon worden aangeboden. Het gebruik verschilt niet tussen GitHub en Amazon zijn privé Git servers. De gebruiker kan gemakkelijk via het web portaal de gewenste Git-projecten toevoegen aan AWS CodePipeline.

AWS CodeBuild is een CI service die code compileert, testen uitvoert en als resultaat software oplevert. AWS CodeBuild is speciaal omdat er geen nood is om zelf de server infrastructuur te configureren voor de compilatie van code. AWS CodeBuild doet dit allemaal voor de gebruiker en schaalt mede naarmate de belasting of het project groter wordt. AWS CodeBuild maakt gebruik van voorverpakte compiler omgevingen maar de gebruiker heeft wel de mogelijkheid om zelf zijn compiler omgevingen te configureren. Dit maakt mogelijk dat de pipeline volledig aan te passen is naar de noden van de gebruiker.

Zodat Amazon weet wat voor compiler omgeving er moet worden gebouwd, moet de gebruiker aan de project folder een BuildSpec.yml toevoegen waarin staat welke compiler engine er gebruikt moet worden met welke files.

Dit kan ook worden gedefinieerd in het web portaal zodanig dat de gebruiker niet de hele tijd broncode moet updaten bij wijzigingen aan de configuratie. Ook heeft AWS CodeBuild een voordeel. Na de compilatie en testen geslaagd zijn, kan er direct een zip gemaakt worden die dan downloadbaar is van Amazon zijn Cloud opslag. AWS CodeBuild zijn prijzen worden op dezelfde manier berekend als Google Cloud Build. Er wordt betaald per

minuut dat er computerkracht gebruikt wordt. Zie de *figuur 4.3* en *figuur 4.4*

Compute types

AWS CodeBuild offers three compute instance types with different amounts of memory and CPU. Charges vary by the compute instance type that you choose for your build.

Region: Europe (Ireland) ▾

Compute instance type	Memory	vCPU	Linux price per build minute	Windows price per build minute
general1.small	3 GB	2	\$0.005	N/A
general1.medium	7 GB	4	\$0.01	\$0.018
arm1.large	16 GiB	8	\$0.015	N/A
general1.large	15 GB	8	\$0.02	\$0.036
general1.2xlarge	144 GiB	72	\$0.20	N/A
gpu1.large	244 GiB	32	\$0.65	N/A

Figuur 4.3: Figuur van AWS website. Figuur toont de prijzen van AWS per rekenkracht, OS en verstreken minuut.

Pricing example

If you execute 100 builds in one month using build.general1.small where each build runs for 5 minutes, then your charges would be calculated as follows:

Monthly build charges

Build minutes = 100 builds * 5 minutes = 500 build minutes

Build minutes – Free tier build minutes = Monthly billable build minutes = 500 – 100 = 400 build minutes

Monthly build charges = 400 build minutes * \$0.005 = \$2

Figuur 4.4: compileer prijzen AWS.Figuur van AWS website. Figuur toont berekening van prijzen per compileer minuut voor AWS.

Een klein detail dat maar zichtbaar was bij het bekijken van de prijzenstelsels. De gebruiker heeft de mogelijkheid om een besturingssysteem (OS) te kiezen bij het aanmaken van een AWS CodeBuild pipeline. Dit onderzoek heeft vastgesteld dat een Windows OS wel beschikbaar is maar niet in iedere datacenter locatie. Vaak is die optie ook duurder dan de Linux variant. Dit kan eventueel problemen veroorzaken met Windows specifieke voorbeelden. Ook is het moeilijk om informatie te vinden over hoe de gebruiker nu juist zelf een aangepaste compileer omgeving definieert.

AWS CodeDeploy is het CD gedeelte van de AWS CodePipeline. Het is volledige te beheren en aanpasbaar naar de noden van de gebruiker. Zo is het mogelijk om rechtstreeks vanuit de pipeline uit te rollen naar eender welke Amazon Cloud service of naar lokale omgevingen. Aangezien het mogelijk is om een zip met de software in te downloaden is het ook gemakkelijk te integreren met bestaande uitrol tools of procedures. Deze feature is ook niet gratis. Volgende *figuur 4.5* toont dit aan. Amazon rekent per update van een instantie een prijs aan. Daarbovenop moet er ook nog betaald worden voor de verbruikte opslag.

Naast een hele serie ontwikkelingshulpmiddelen heeft Amazon ook hulpmiddelen toegevoegd om gedetailleerde rapporten en analyses te genereren van de uitgevoerde taken op

For CodeDeploy on EC2/Lambda: There is no additional charge for code deployments to Amazon EC2 or AWS Lambda through AWS CodeDeploy.

For CodeDeploy On-Premises: You pay \$0.02 per on-premises instance update using AWS CodeDeploy. There are no minimum fees and no upfront commitments. For example, a deployment to three instances equals three instance updates. You will only be charged if CodeDeploy performs an update to an instance. You will not be charged for any instances skipped during the deployment.

You pay for any other AWS resources (e.g. S3 buckets) you may use in conjunction with CodeDeploy to store and run your application. You only pay for what you use, as you use it; there are no minimum fees and no upfront commitments.

Figuur 4.5: extra kosten. Figuur van AWS website. Figuur toont prijs voor opslag op AWS.

AWS CodePipeline. Dit geeft de gebruiker goede inzichten in wat er juist allemaal gebeurt en verkeerd loopt. Ook kan de gebruiker op basis van foutmeldingen of status rapporten, bepaalde acties instellen en laten uitvoeren.

Het zou mogelijk zijn om met AWS de gewilde functionaliteit te realiseren. Het zal wel zeer arbeid intensief zijn aangezien informatie over een aangepaste compilatie omgeving moeilijk te vinden is. Ook is het een stuk duurder. Dit onderzoek heeft ook vastgesteld dat Amazon een hele reeks van producten heeft waardoor het soms moeilijk is om door het bos de bomen te zien. Ook de prestatie van het platform blijft een vraagteken door de paper (Jackson e.a., 2010).

4.4 IBM Cloud

IBM Cloud is mogelijk een van de oudere Cloud platformen. Nog voor de term Cloud veel gebruikt werd. IBM was al vroeg bezig met het idee om hardware open te stellen om dan meerdere machines of services op te laten draaien. In 1972 heeft IBM de eerste stappen gezet naar IaaS door voor hun mainframe een hypervisor te bouwen die toestond dat er meerdere instanties van een besturingssysteem op hetzelfde systeem draaide (VM's). Dit is dan later geëvolueerd naar een meer typische Cloud infrastructuur. IBM heeft in de vroege jaren van hun Cloud systeem, vooral hardware voorzien aan klanten. De zo genoemde privé Cloud. In 2007 werden dan de eerste stappen gezet naar de typisch Cloud infrastructuur door het verhuur van rekenkracht vanuit hun datacenters met hun hardware.

Heden ten dage is IBM Cloud een stuk uitgebreider. Het valt onder te verdelen in drie grote categorieën. 'SmartCloud Foundation', 'SmartCloud Service's en 'SmartCloud Solutions'. Volgens IBM is het hun bedoeling om de gaten in het aanbod van andere Cloud platform aanbieders op te vullen.

'SmartCloud Foundation' is een serie producten die privé Cloud en Hybride Cloud mogelijk moeten maken. Het biedt de infrastructuur, beheer, beveiliging, hardware en integratie aan. 'SmartCloud Services' zijn dan de verschillende hulpmiddelen om dit te bereiken of te gebruiken. (IaaS of PaaS.) 'SmartCloud Solutions' is dan meer een pakket dat samenwerking, statistieken enz. moet mogelijk maken binnen de services en aanbiedingen van IBM.

Ook IBM Cloud heeft producten om een CI/CD pipeline te maken. Al is er toch een addertje onder het gras. IBM Cloud voorziet infrastructuur om vooral CD te kunnen realiseren. Dit met mogelijkheden om de infrastructuur te definiëren. Hulpmiddelen om de

uitrol te beheren en te analyseren. Dit alles kan worden gecontroleerd, zoals alle andere Cloud platform aanbieders, door middel van een speciaal ontwikkelde command line interface (CLI) of door hun web portaal. Voor CI biedt IBM niks specifiek aan. Er bestaat wel de mogelijkheid om het Tekton framework te gebruiken op de Cloud infrastructuur van IBM maar dat kan bij ieder Cloud platform. Dit framework valt ook buiten de scope van dit onderzoek.

Op basis van hun producten en services die ze aanbieden valt IBM Cloud uit de boot. Het zou zeer omslachtig zijn om IBM Cloud te gebruiken voor een Microsoft georiënteerde pipeline. Aangezien er geen specifieke compiler technieken aanwezig zijn. Naast het Tekton framework. Ook is de prestatie van de IBM-datacenters niet slecht. Het zijn speciaal ontworpen centers met eigen hardware en voorzieningen van IBM. Wat wel blijkt uit onderzoek van de ontwikkelingshulpmiddelen van IBM Cloud, is dat IBM zich inzet om gemakkelijk te gebruiken hulpmiddelen te ontwikkelen die weinig moeite kosten om te implementeren en te configureren. Ook hebben ze als enige specifiek een Cloud aanbod voor Apple georiënteerde applicaties.

4.5 Andere

Naast de alom bekende giganten zoals Azure, Google Cloud, IBM Cloud en AWS zijn er nog een aantal andere Cloud platform aanbieders. Zo bestaat er nog Oracle Cloud en Digital Ocean. Er bestaan waarschijnlijk nog wel maar deze laat dit onderzoek buiten beschouwing.

Oracle Cloud zijn aanbod van producten en services liggen vooral in vier categorieën. IaaS, PaaS, 'Software as a Service' (SaaS) en 'Data as a Service'. Oracle Cloud hun aanbod is hoofdzakelijk hetzelfde als alle andere Cloud platformen. Juist zoals IBM Cloud heeft Oracle hun eigen hardware en eigen datacenters. Oracle Cloud biedt oplossingen en producten aan om DevOps te realiseren maar deze zijn vooral gefocust op het Java platform. Om deze reden valt ook Oracle Cloud uit de boot. Aangezien we in dit onderzoek trachten om een Microsoft georiënteerde pipeline te realiseren. Het is wel mogelijk door gebruik te maken van het Tekton Framework. Maar dit laten we buiten beschouwing in dit onderzoek.

Digital Ocean is een Cloud platform speciaal gemaakt voor ontwikkelaars. Het Cloud platform is een van de jongere spelers tussen de Cloud platformen. Digital Ocean is opgericht in 2011. Hun doel is om een omgeving aan te bieden aan ontwikkelaars waarin programmeurs gemakkelijk kunnen ontwikkelen en testen. Ook tracht Digital Ocean om deze omgeving open te stellen voor productie door het zeer gemakkelijk te maken om services en applicatie schaalbaar te maken op hun platform. Digital Ocean heeft jammer genoeg geen specifiek CI/CD mogelijkheid. Het zou wel mogelijk zijn met het Tekton framework aangezien Digital Ocean Kubernetes ondersteund. Voor deze redenen valt Digital Ocean ook buiten de boot.

4.6 Gebruiksvriendelijkheid

Naast de kandidaat selectie op productaanbod en prijs, wil dit onderzoek ook de gebruiksvriendelijkheid in acht nemen. Hiervoor is er een simpele Java applicatie gebruikt. Alle Cloud platformen bieden startershandleidingen aan voor het opzetten van een CI-pipeline voor Java op hun platform. Het doel van deze snelle test is om als eindresultaat een werkende Java JAR te hebben. Ook werd er bijgehouden hoelang het duurde om een pijpleiding te configureren om beter een idee te krijgen van hoe gebruiksvriendelijk het Cloud platform werkelijk is.

De Java Applicatie bestaat uit een hoofdklasse en een testklasse. De hoofdklasse, ‘MessageUtil’ bevat een variabele voor een bericht te bewaren en vier methodes. De eerste methode is een constructor voor het aanmaken van een ‘MessageUtil’ object met een meegegeven bericht. De tweede methode print dit bericht. In de derde methode wordt er een toevoegsel aan dit bericht gevoegd. Ook wordt het geheel geprint. De laatste methode is een uitvoerbare methode die een ‘MessageUtil’ object maakt en de twee print methodes uitvoert. De code voor ‘MessageUtil.java’ wordt afgebeeld in *figuur 4.1*.

De testklasse ‘TestMessageUtil’ bevat twee testen voor de print methodes. *Figuur 4.2* toont de code voor dit Javabestand.

```
public class MessageUtil {
    private String message;

    public MessageUtil(String message) {
        this.message = message;
    }

    public String printMessage() {
        System.out.println(message);
        return message;
    }

    public String salutationMessage() {
        message = "Hi!" + message;
        System.out.println(message);
        return message;
    }

    public static void main(String[] args) {
        MessageUtil mu = new MessageUtil("Aucxis\u2026");
        mu.printMessage();
        mu.salutationMessage();
    }
}
```

Listing 4.1: Java bestand MessageUtil.java. Hoofd klasse van de applicatie *MessageUtil*

```
import org.junit.Test;
import org.junit.Ignore;
import static org.junit.Assert.assertEquals;

public class TestMessageUtil {

    String message = "Robert";
    MessageUtil messageUtil = new MessageUtil(message);

    @Test
    public void testPrintMessage() {
        System.out.println("Inside testPrintMessage()");
        assertEquals(message, messageUtil.printMessage());
    }

    @Test
    public void testSalutationMessage() {
        System.out.println("Inside testSalutationMessage()");
        message = "Hi!" + "Robert";
        assertEquals(message, messageUtil.salutationMessage());
    }
}
```

Listing 4.2: Java Test klasse MessageUtilTest.java, voor het testen van *MessageUtil 4.1*.

Deze Javabestanden worden op alle Cloud platformen gecompileerd aan de hand van de Maven compiler. Om de compiler te vertellen wat er moet gebeuren tijdens de compilatie van de Javabestanden, moet er een Xml-bestand worden aangemaakt waarin deze opties gedefinieerd staan. Deze ‘pom.xml’ wordt afgebeeld door *figuur 4.3*. Hierin staat dat de meegeleverde Junit testen moeten worden uitgevoerd. Ook wordt er gedefinieerd dat de Java applicatie moet worden gecompileerd tot een JAR-bestand.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
    maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.example</groupId>
  <artifactId>messageUtil</artifactId>
  <version>1.0</version>
  <packaging>jar</packaging>
  <name>Message Utility Java Sample App</name>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
```

```

<scope>test</ scope>
</ dependency>
</ dependencies>
<build>
<plugins>
<plugin>
<groupId>org . apache . maven . plugins</ groupId>
<artifactId>maven-jar-plugin</ artifactId>
<version>3.1.0</ version>
<configuration>
<archive>
<manifest>
<addClasspath>true</ addClasspath>
<classpathPrefix>lib /</ classpathPrefix>
<mainClass>MessageUtil</ mainClass>
</ manifest>
</ archive>
</ configuration>
</ plugin>
</ plugins>
</ build>
</ project>

```

Listing 4.3: XML-bestand voor de Maven compiler. Definieert stappen voor het compileren van een java applicatie.

Deze bestanden zijn toegevoegd aan een folder die de structuur hanteert uit *figuur 4.4*. Deze hoofdmap is dan geïnitialiseerd als een Git repository. Ook is er een gitignore aangemaakt. Deze Java applicatie is het startpunt voor alle testen voor gebruiksvriendelijkheid op de Cloud platformen. Op deze manier is er geprobeerd om op basis van eventuele verschillen een geschikte kandidaat te selecteren voor een Proof Of Concept. In de volgende secties 4.6.1, 4.6.2 & 4.6.3 wordt kort de werkwijze beschreven. Hier wordt de ‘hoe’ minder aangehaald omdat dit in de handleiding beschreven staat.

```

Demo_java_aws /
‘--- .gitignore
‘--- pom.xml
‘--- src
    ‘--- main
        ‘--- java
            ‘--- MessageUtil.java
    ‘--- test
        ‘--- java
            ‘--- TestMessageUtil.java

```

Listing 4.4: Output van het tree commando in een java applicatie bestanden structuur.

4.6.1 Java Amazon AWS

Voor de test op Amazon AWS is deze *handleiding* gevolgd. Er zijn wel een aantal zaken aangepast geweest om een realistischere opstelling te benaderen. De handleiding begint met het aanmaken van twee S3 Storage Buckets. In dit geval is er gekozen om maar een S3 Storage Bucket aan te maken. Deze zal dienen als een output voor bestanden. Voor de input werd een GitHub repository gebruikt. Vervolgens wordt in de handleiding de applicatie gemaakt. Voor dit experiment is er voor de drie Cloud platformen dezelfde applicatie gebruikt. *Figuur 4.1* toont de hoofdklasse van deze applicatie en *figuur 4.2* toont de testklasse van deze applicatie. Vervolgens wordt er in de handleiding een YAML-bestand aangemaakt waarin de verschillende stappen voor het compileren en testen van de applicatie worden gedefinieerd.

Dit YAML-bestand heeft op Amazon AWS de naam ‘buildspec.yml’. *Figuur 4.5* toont dit bestand. Over het aanpassen van dit YAML-bestand is er maar weinig informatie te vinden. Ook is het niet helemaal duidelijk hoe er precies een andere virtuele machine moet worden gekozen. Tijdens het uitvoeren van de configuratie is er gebleken dat de documentatie tot verwarring kan lijden. Dit YAML-bestand bevat altijd vier fasen waarin een reeks commando’s kunnen worden uitgevoerd. De eerste fase is de ‘install’ fase. Hier wordt er gedefinieerd wat voor omgeving moet worden gebruikt. In dit geval gebruikt de handleiding ‘corretto11’. Dit is een Linux gebaseerde machine. De tweede fase is de ‘pre-build’ fase. Hierin kunnen er commando’s worden uitgevoerd om de machine voor te bereiden, repositories te kopiëren, enz. In dit geval staat er een echo-commando als plaatsvulling. De derde stap is de ‘build’ stap. Hierin worden de commando’s geschreven die ervoor moeten zorgen dat de applicatie wordt gecompileerd. In een vierde fase, de ‘post-build’ fase, is er de mogelijkheid om nog commando’s uit te voeren na het compileren van de applicatie. Dit is in dit geval ook ongebruikt. Voor een Java applicatie wordt alles door middel van een pom.xml uitgevoerd. *Figuur 4.3* toont dit bestand. Het enige dat dus moet worden gedefinieerd in de ‘buildspec.yml’ is het uitvoeren van een herstel op de Java-bestanden.

```
version: 0.2

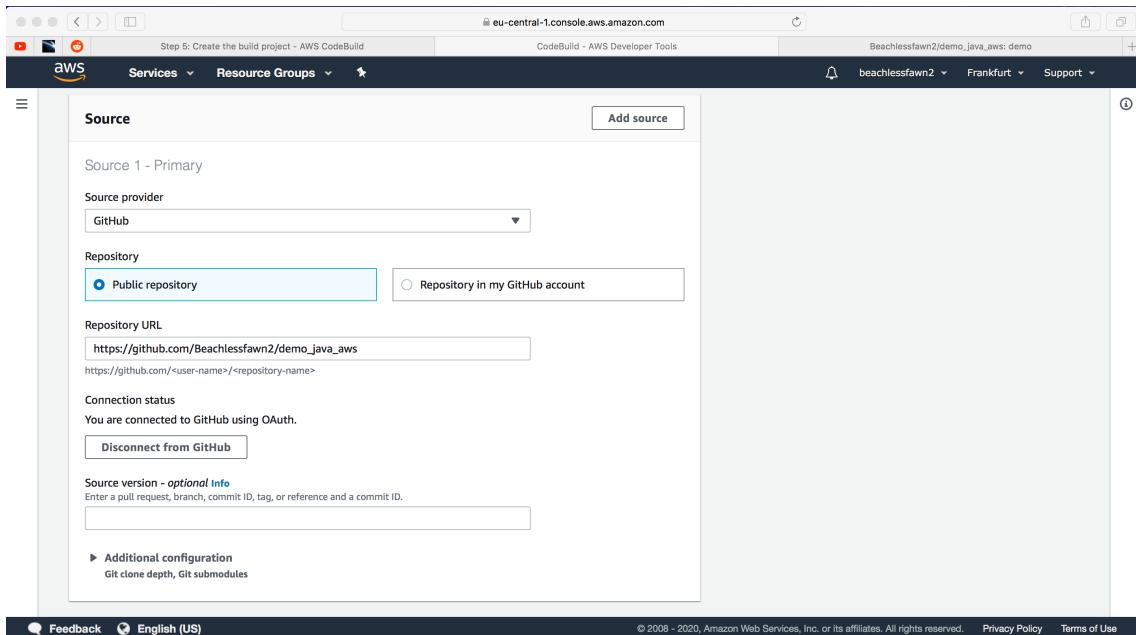
phases:
  install:
    runtime-versions:
      java: corretto11
  pre_build:
    commands:
      - echo Nothing to do in the pre_build phase...
  build:
    commands:
      - echo Build started on `date`
      - mvn install
  post_build:
    commands:
      - echo Build completed on `date`
```

```

artifacts:
  files:
    - target/messageUtil-1.0.jar
  
```

Listing 4.5: YAML-bestand voor het configureren van een pijpleiding op Amazon AWS.

In een volgende stap wordt in de handleiding de pipeline aangemaakt op Amazon AWS. Dit werd volledig gedaan via de online console van Amazon AWS. Een verschil in deze configuratie is het selecteren van een bron voor de broncode. Deze bevindt zich nu op GitHub. *Figuur 4.6* toont de gewijzigde configuratie op AWS. Voor alle ander opties is de handleiding gevuld. Ook de volgende stappen vanuit de handleiding zijn onveranderd gevuld. Zo werd er als resultaat in de Bucket een Jar-bestand achtergelaten. Deze was zeer gemakkelijk te downloaden om daarna uit te voeren.



Figuur 4.6: BronCode locatie seleceteren op AWS. Deze figuur toont het selecteren van een GitHub repository bij het aanmaken van een pipeline op Amazon AWS.

Het volgen van deze handleiding ging niet zonder slag of stoot. Het was in het begin zeer moeilijk om te begrijpen wat ‘buildspec.yml’ nu allemaal kan. Ook heeft Amazon AWS een hele reeks van producten dat het soms moeilijk maakt om te selecteren wat er nu juist nodig is. De tijd die in beslag genomen is in vergelijking met de andere platformen, om de handleiding van het begin tot het einde te volgen wordt getoond in *tabel 4.1*. Hier is ook het opstellen van de applicatie meegerekend en ook het uitvoeren van het resulterende Jar-bestand.

4.6.2 Java Google Cloud Platform

Voor Google Cloud Code Build is volgende *handleiding* gebruikt geweest. Al snel bleek bij het begin van de handleiding dat er geen manier was om via een online console een

pipeline te configureren. De pipeline en de verschillende logs kunnen wel bekijken worden op het portaal. Er bestaat een mogelijkheid om in de console een terminal te openen waarin Google Cloud CLI geïnstalleerd staat. Hiermee kan de gebruiker alle producten beheren op het platform. Om een pijpleiding aan te maken van op een lokale computer, moet eerst de *Google Cloud CLI* worden geïnstalleerd. Deze CLI is beschikbaar op ieder besturingssysteem en staat de gebruiker toe om alle services en producten van GCP aan te sturen en te beheren. De installatie ervan is eenvoudig. Zeker met behulp van de *documentatie* van Google zelf. Ook moet er een recente versie van Python beschikbaar zijn op de gebruiker zijn computer. Na deze te hebben geïnstalleerd moet het Google Cloud account geconnecteerd worden aan de CLI. Deze *documentatie* legt dit zorgvuldig uit.

Daarna gaat de handleiding verder met het definiëren van een YAML-bestand voor de configuratie van de pipeline. Bij Google Cloud noemt dit bestand ‘cloudbuild.yml’. *Figuur 4.6* toont het gebruikte bestand voor dit geval. In deze configuratie kunnen er zoveel stappen worden gedefinieerd als de gebruiker wenst. In dit geval zijn er slechts twee stappen gedefinieerd. In een eerste stap wordt de code getest door middel van het test commando op een Maven Docker container. In een tweede stap wordt door middel van dezelfde machine de applicatie gecompileerd tot een Jar-bestand. In een aanvullende stap wordt deze Jar geüpload naar een Storage Bucket op Google Cloud. Een Storage Bucket kan worden gecreëerd door middel van de console. De gebruiker moet enkel een naam ingeven. Hierna is er het commando ‘glcoud submit’ uitgevoerd om deze pijpleiding uit te voeren.

```
steps:
  - name: maven:3-jdk-8
    entrypoint: mvn
    args: [ 'test' ]
  - name: maven:3-jdk-8
    entrypoint: mvn
    args: [ 'package', '-Dmaven.test.skip=true' ]
    artifacts:
      objects:
        location: 'gs://output-bucket-codebuild/target/'
        paths: [ '/workspace/target/messageUtil-1.0.jar' ]
```

Listing 4.6: YAML-bestand voor het configureren van een pijpleiding voor java op Google Cloud.

Dit hele proces was vrijwel pijnloos. Ook de extra documentatie voor het aanmaken van Storage Buckets, het uploaden van artefacten en het installeren van Google Cloud Cli, was gemakkelijk te vinden. Het gecompileerde Jar-bestand was gemakkelijk te downloaden van de Stroage op Google Cloud. De tijd die in beslag genomen is in vergelijking met de andere platformen, om de handleiding van het begin tot het einde te volgen wordt getoond in *tabel 4.1*. Hier is ook het opstellen van de applicatie meegerekend en ook het uitvoeren van het resulterende Jar-bestand. Deze configuratie ging vlot door het gemakkelijk vinden van de benodigde informatie en de intuïtieve configuratie bestanden.

4.6.3 Java Azure DevOps

Voor deze opstelling is deze *handleiding* gebruikt geweest. Vooraleer deze handleiding kon worden gevuld, is er eerst een nieuwe organisatie aangemaakt geweest in Azure DevOps. Binnen deze organisatie is er dan een nieuw project aangemaakt geweest om de pipelines in te configureren. Zo een Azure DevOps project bevat een hele serie producten en services om ontwikkelaars en organisaties te helpen in het softwareontwikkelingsproces. Ook is zo een project handig omdat het net zoals de andere Cloud platformen, alle aanrekeningen en producten stopzet bij het verwijderen.

Verder moest er ook een GitHub repository bestaan om deze handleiding te kunnen starten. Deze repository bevat de Java applicatie met de hoofdklasse en de testklasse. *Figuur 4.1* en *figuur 4.2* toont de code van deze Java klassen. Ook moet er een pom.xml aanwezig zijn. *Figuur 4.3* toont dit XML-bestand. Deze bestanden zijn dan geüpload geweest naar GitHub.

Voor het aanmaken van een pipeline op Azure DevOps is de documentatie verder gevold. De combinatie van de handleiding en de online wizards maakten het configureren van een pipeline gemakkelijk. In een eerste stap kiest de gebruiker de bron van de code. Hier is GitHub gekozen, zoals de handleiding suggereerde. Dan werd er automatisch omgeleid naar de GitHub login pagina. Hier kan de gebruiker zijn inlog gegevens ingeven. Bij het inloggen wordt de gebruiker gevraagd om de Azure DevOps applicatie toe te voegen aan GitHub. Deze applicatie staat Azure DevOps toe om rechtstreeks de broncode te kunnen aanspreken. Ook wordt er automatisch een trigger ingesteld. Deze kan later worden aangepast naar de gebruiker zijn noden. Nadat de GitHub applicatie toegevoegd is, werd de broncode uitgelezen door Azure DevOps. Azure DevOps stelt op basis hiervan, de beste optie voor het compileren van de code voor. In dit geval was dat Maven.

Dan gaat Azure DevOps automatisch een YAML-bestand gaan toevoegen aan de gebruiker zijn broncode op GitHub. Dit YAML-Bestand heeft de naam ‘azure-pipelines.yml’ en bevat alle configuratie in verband met het CI gedeelte van de pipeline. *Figuur 4.7* toont de code van het gegenereerde YAML-bestand voor dit geval. In dit bestand staan er een aantal zaken gedefinieerd. Hier kan de gebruiker een trigger gaan instellen voor de pijpleiding. Hier is dit de master tak. Verder kan de gebruiker een virtuele machine definiëren. Tot slot worden de compileren stappen gedefinieerd. Hier zijn er twee stappen gedefinieerd geweest. In de eerste stap wordt de volledige applicatie gecompileerd. Ook worden de meegeleverde Junit testen uitgevoerd. En de applicatie wordt gecompileerd tot een Jar-bestand. In de tweede stap wordt het Jar-bestand geüpload naar Azure DevOps. Deze laatste stap staat niet in de handleiding. Het opzoeken van de nodige informatie voor deze stap te definiëren was niet tijdrovend. De oplossing was snel gevonden in andere documentatie die meer uitleg geeft over ‘Azure DevOps Artifacts’. Ook moet voor deze stap niets extra worden toegevoegd aan het project. Azure DevOps creëert automatisch de gewenste bestanden structuur.

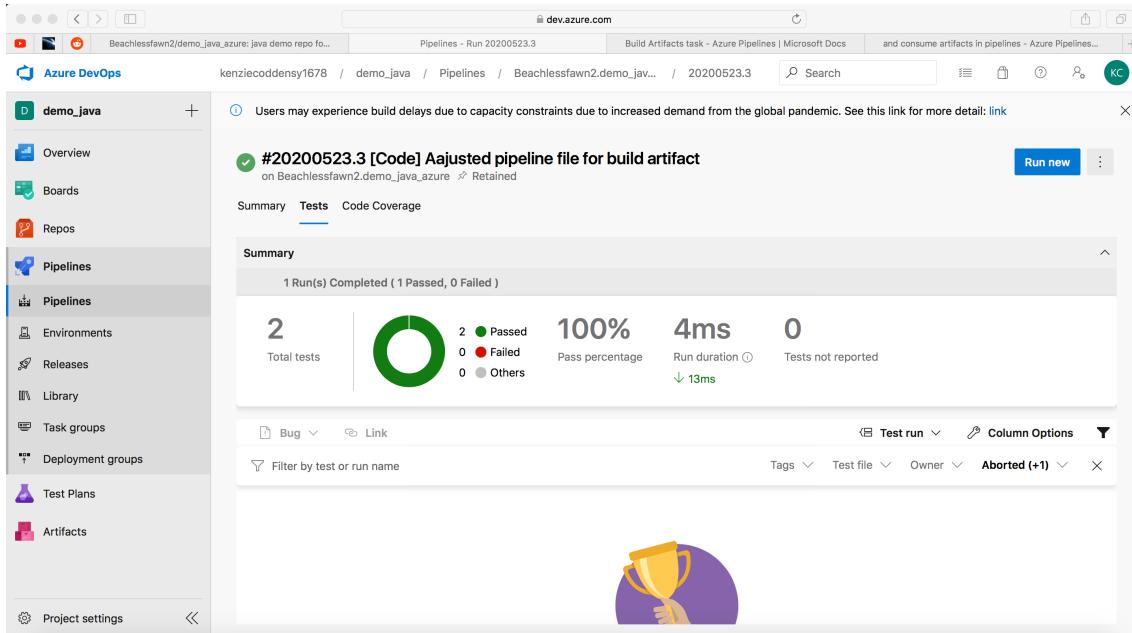
```
# Maven
# Build your Java project and run tests with Apache Maven.
# Add steps that analyze code, save build artifacts, deploy
```

```
, and more:  
# https://docs.microsoft.com/azure/devops/pipelines/  
languages/java  
  
trigger:  
  - master  
  
pool:  
  vmImage: 'ubuntu-latest'  
  
steps:  
- task: Maven@3  
  inputs:  
    mavenPomFile: 'pom.xml'  
    mavenOptions: '-Xmx3072m'  
    javaHomeOption: 'JDKVersion'  
    jdkVersionOption: '1.8'  
    jdkArchitectureOption: 'x64'  
    publishJUnitResults: true  
    testResultsFiles: '**/surefire-reports/TEST-*.xml'  
    goals: 'package'  
- task: PublishPipelineArtifact@1  
  inputs:  
    targetPath: $(System.DefaultWorkingDirectory)/  
      target/messageUtil-1.0.jar  
    artifactName: messageUtil
```

Listing 4.7: YAML-bestand voor het configureren van een pipeline voor java op Azure DevOps.

Nadat dit YAML-bestand is toegevoegd aan de broncode op GitHub, is de pipeline uitvoerbaar. Azure DevOps visualiseert de resultaten van het uitvoeren van deze pipeline zeer goed. *Figuur 4.7* toont het dashboard van de testen. Hierop kan de gebruiker zijn slaag percentages bekijken. Ook kan de gebruiker direct zien welke testen gefaald zijn.

Deze handleiding is duidelijk geschreven. Een gebruiker kan deze perfect volgen zelfs als er geen voorkennis is over het platform. Ook het configureren van de pipeline is veel minder complex dan de andere Cloud platformen. Extra informatie en documentatie is gemakkelijk te vinden en deze is te begrijpen. Het hele proces voelde zeer intuïtief aan. Bovenal, heeft het weinig tijd gekost om deze pipeline te configureren. De bestanden structuur voor de applicatie heeft langer geduurd om te maken dan de pipeline zelf. *Tabel 4.1* toont de gespendeerde tijd van begin tot einde voor alle drie de kandidaten. Dit product van Azure is zeer krachtig en integreert goed met andere tools en bestaande procedures. Het zal dan ook niet gemakkelijk zijn voor het andere Cloud platform om dit te evenaren.



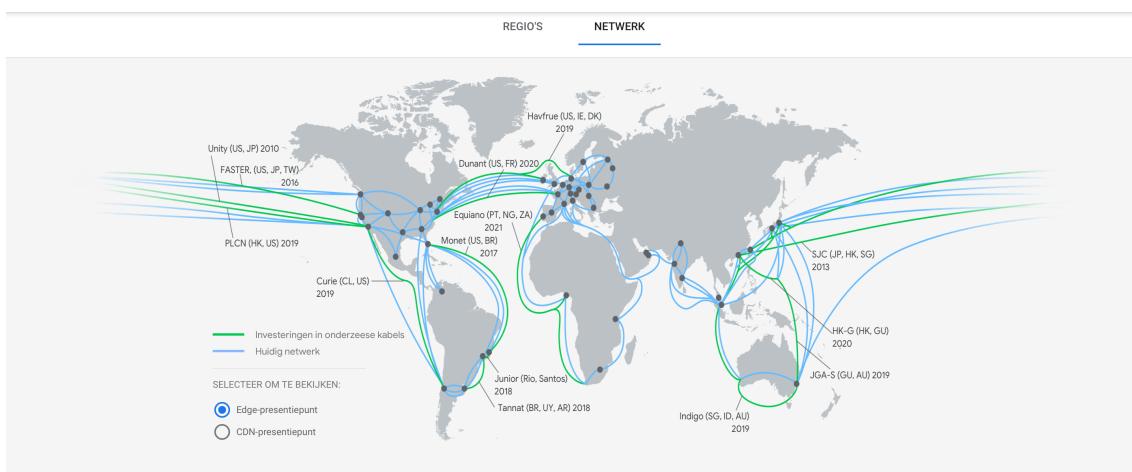
Figuur 4.7: Een ingebouwd dashboard op Azure DeOps. Dit dashboard geeft de resultaten van de uitgevoerde testen tijdens de laatste compilatie.

4.7 Kandidaat

Na al deze Cloud platform aanbieders hun aanbod naast elkaar gelegd te hebben, is er toch een platform dat er wat van tussen uitspringt. Dit is GCP. Daarom is deze Cloud aanbieder ook gekozen in dit onderzoek om een Proof Of Concept op uit te werken. Google is niet alleen een stuk goedkoper, het gebruikt ook eenvoudig en gemakkelijk te gebruiken containers. Ook is het een voordeel dat er gemakkelijk gebruikersrechten kunnen worden aangemaakt op basis van de GitHub deelnemers. Google heeft ook de meest duidelijke informatiebronnen. Ook zijn het hypermoderne datacenters die over heel de wereld verspreid zijn. Dus prestatie zou geen probleem mogen zijn. Zie figuur 4.8. Ook het uitvoeren van de gebruiksvriendelijkheid testen heeft dit bevestigd. Tabel 4.1 toont de gespendeerde tijd. Het moeten opzoeken van extra informatie heeft hier een grote rol gespeeld.

	Amazon AWS	Google Cloud	Azure DevOps
Gespendeerde tijd in minuten	195	125	60

Tabel 4.1: toont de gespendeerde tijd voor de gebruiksvriendelijkheid test per Cloud platform. De tijden staan in minuten. De Cloud platformen vergelijken goed met elkaar. Ook schaalt de complexiteit mee met de gespendeerde tijd.



Figuur 4.8: Overzicht van datacenter locaties Google Cloud. Figuur van Google Cloud website. Figuur toont de connectiviteit van de verschillende datacenters verspreid over de wereld.

5. Proof Of Concept

Omdat Aucxis met een Microsoft georiënteerde werkwijze werkt, moet er aangetoond worden dat het mogelijk is om dit te realiseren op GCP. Voor deze Proof Of Concept (POC) wordt er een CI/CD pipeline geconfigureerd. Hierin wordt er een simpele .NET Core applicatie gecompileerd. Ook de meegeleverde testen worden uitgevoerd tijdens de compilatie. Hierna wordt de applicatie en de bijbehorende componenten gecomprimeerd en naar een lokale fileserver geüpload. De gecompileerde applicatie kan dan uitgebreid worden getest in een lokale test omgeving. Ook is er een stap voorzien in de pijpleiding om de gecompileerde applicatie tijdelijk op het Cloud platform te bewaren. Dit voor het geval er iets verkeerd loopt tijdens het testen of tijdens het uploaden.

De .Net applicatie is een simpele .Net core console applicatie. Het bestaat uit een klasse, *MessageUtil 5.1*. Deze klasse bevat een variabele, twee constructors, een methode om de variabele op te vragen, een methode om de variabele te tonen met een toevoegsel en een hoofdmethode die uitvoerbaar is.

```
using System;
using System.Threading;

namespace MessageUtil {
    public class MessageUtilProgram {

        private String p_message;

        private MessageUtilProgram() { }

        public MessageUtilProgram( String message ) {
            p_message = message;
        }
    }
}
```

```

public String Message {
    get { return p_message; }
}
public String SaluteMessage( String m) {
    Console.WriteLine("Hello\n{0}", m);
    return "hello" + m;
}

static void Main( string[] args ) {
    MessageUtilProgram mup = new MessageUtilProgram("Aucxis");
    Console.WriteLine("{0}", mup.Message);
    //mup.SaluteMessage(mup.Message);
    Thread.Sleep(60000);
}
}
}
}

```

Listing 5.1: C# bestand MessageUtil.cs. Hoofd klasse van de applicatie *MessageUtil*

Daarnaast is er ook een testklasse voorzien, *MessageUtilTest* 5.2. Hierin staan er twee methodes. De eerste methode test de constructor die de variabele moet instellen. De tweede methode test een van de print methodes.

```

using Microsoft.VisualStudio.TestTools.UnitTesting;
using MessageUtil;

namespace MessageUtilTest {

    [TestClass]
    public class MessageUtilTests {

        [TestMethod]
        public void ConstructWorks() {
            string testmessage = "Test";
            MessageUtilProgram mup = new MessageUtilProgram(testmessage);

            string value = mup.Message;
            Assert.AreEqual(testmessage, value, "Message didn't set correctly");
        }

        [TestMethod]
        public void SaluteWorks() {
            string testmessage = "Test";
            string expected = "helloTest";
        }
    }
}

```

```

MessageUtilProgram mup2 = new MessageUtilProgram(
    testmessage);

string value = mup2.SaluteMessage(mup2.Message);
Assert.AreEqual(expected, value, "Message didn't salute_
    correctly");
}
}
}
}

```

Listing 5.2: C# Test klasse MessageUtilTest.cs, voor het testen van MessageUtil 5.1.

De applicatie wordt in beide gevallen onveranderd gebruikt zodanig dat een potentieel verschil zichtbaar wordt. Dit is de basis waaruit vertrokken is voor de POC. Om de prijs en het aantal benodigde producten zo laag mogelijk te houden, is er in dit onderzoek gekozen om Git en GitHub te gebruiken als versiebeheersysteem. GitHub is volledig ondersteund door beide platformen. Ook voorzien beide Cloud platformen applicaties op GitHub om automatisch de gewenste repositories te verbinden aan de CI/CD pipeline.

Het uploaden van de gecomprimeerde applicatie wordt door middel van Secure File Transfer Protocol (SFTP) op een Linux Ubuntu machine gedaan. Deze maakt verbinding met een Docker container die lokaal op een Windows Server 2019 draait. Deze *Docker container* deelt een directory met Windows zodanig dat dit volledig modulair is met andere besturingssystemen of situaties. SFTP werkt onderliggend op basis van Secure Shell (SSH). Om verbinding te maken is dus een wachtwoord nodig. Dit is een probleem aangezien de virtuele systemen in de Cloud niet interactief zijn. Dit is opgelost door het gebruik van *SSHPASS*. Dit Linux pakket heeft de mogelijkheid om aan SSH-toepassingen het wachtwoord mede te geven. Weliswaar zonder encryptie van het wachtwoord. Hierdoor is het wachtwoord leesbaar. Er wordt enkel op deze wijze gewerkt om het geheel relatief simpel te houden. Voor productie omgevingen zou er met SSH-sleutels moeten worden gewerkt. De SFTP operatie op de Linux Ubuntu in de Cloud, wordt uitgevoerd door middel van het *script 5.3*. Dit Bash script zorgt ervoor dat het *SSHPASS* pakket beschikbaar is en voert de SFTP transactie uit.

```

#!/bin/bash
apt-get update
apt-get -y upgrade
apt-get -y install sshpass
sshpass -p `pwd` sftp -o StrictHostKeyChecking=accept-new -
    P 5151 -oBatchMode=no -b - TestU@server << !
cd documents
put /workspace/MessageUtil/bin/Release/netcoreapp3.1/win10-
    x64/messageutil-win10-x64.tar.gz
bye
!

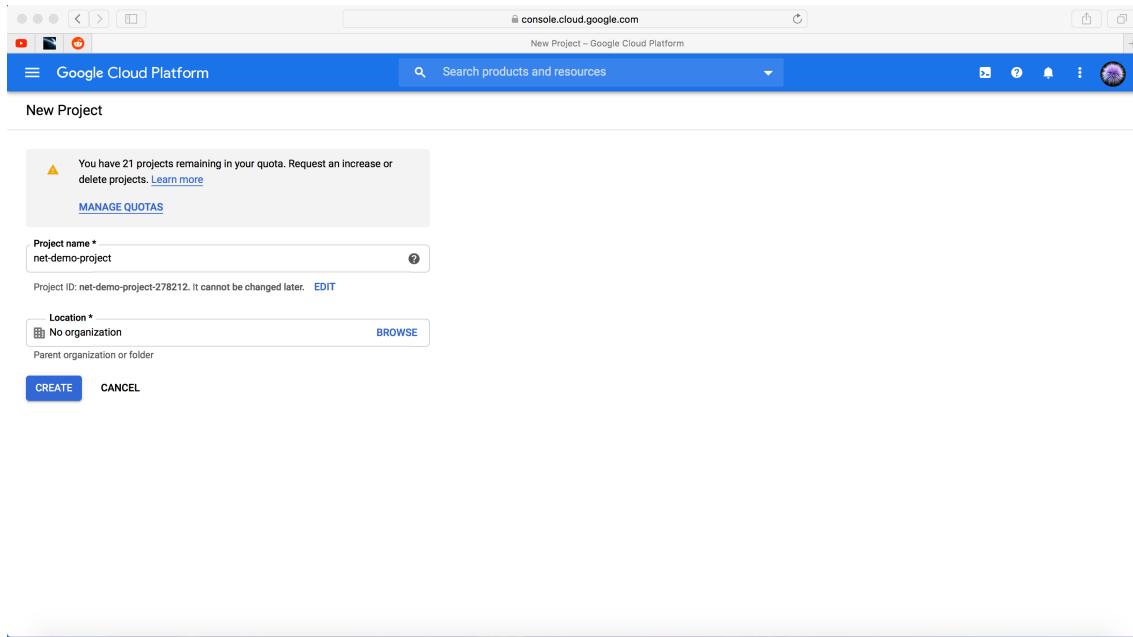
```

Listing 5.3: Bash script filetrans.sh. Script installeert de juiste Linux packages en kopieert de bestanden met SFTP.

De Cloud specifieke configuratie bestanden worden in de volgende secties verder verduidelijkt. Voor GCP in *sectie 5.0.1* en voor het Azure DevOps platform *sectie 5.0.2*.

5.0.1 Google Cloud Platform

Voor deze POC is er een nieuw project aangemaakt op Google Cloud. Als op Google Cloud een project verwijderd wordt, worden ook alle resources en aanrekeningen stopgezet. Ook omdat er dan per project producten en services toegewezen kunnen worden. Zo kan er optimaal voldaan worden aan de gebruiker zijn noden. Het nieuw gecreëerde project, heeft de naam ‘net-demo-project’ gekregen zoals in *figuur 5.1*.



Figuur 5.1: New Project scherm op Google Cloud. Figuur toont het scherm om een nieuw project te creëren op Google Cloud.

Om Code Build op Google Cloud te gebruiken moest eerst de API via de online console worden geactiveerd. Ook moest er nog een Storage Bucket worden aangemaakt voor het opslaan van de gecompileerde applicatie. De Storage service is standaard geactiveerd en hoefde dus niet te worden aangezet. Deze Storage Bucket heeft de naam ‘net-demo-output-bucket’ gekregen. Ook belangrijk was de selectie voor de locatie van deze Storage Bucket. Hier is er gekozen voor een geografisch zo dicht mogelijke locatie. Dit om de overdracht tijden van bestanden zo minimaal mogelijk te houden. Alle andere opties zijn onveranderd gebleven. Ook is er de mogelijkheid om toegangsrechten toe te kennen. Dit kan interessant zijn voor een productie omgeving. *Figuur 5.2* toont de gecreëerde Storage Bucket en de geselecteerde opties.

Hierna is er een nieuwe map aangemaakt op de gebruiker zijn computer voor de applicatie. Deze is geïnitialiseerd als een Git repository. Hierin zijn dan de bestanden, *MessageUtil 5.1* & *MessageUtilTest 5.2* voor de .Net applicatie in geplaatst. Ook het *script 5.3* voor de bestanden overdracht met SFTP is hierbij toegevoegd. Hiernaast is er ook een gitignore

The screenshot shows the Google Cloud Platform Storage browser interface. On the left, there's a sidebar with options like Storage, Browser, Transfer, Transfer for on-premises, Transfer Appliance, and Settings. The main area is titled 'Storage browser' and shows a table of buckets. A single row is visible for 'net-demo-output-bucket'. The table columns include Name, Created, Location type, Location, Default storage class, Updated, and Public access. The 'Created' column shows 'May 21, 2020, 2:21:29 PM'. The 'Location' column shows 'eu (multiple regions)'. The 'Default storage class' is 'Standard'. The 'Updated' column shows 'May 21, 2020, 2:21:29 PM'. The 'Public access' column shows 'Subject to object-level control'. There are also 'DISMISS' and 'ACTIVATE' buttons at the top right of the browser area.

Figuur 5.2: Storage Bucket scherm op Google Cloud. Figuur toont het scherm met een gecreëerde Storage Bucket en de geconfigureerde opties op Google Cloud.

aangemaakt. *Figuur 5.4* toont een tree van de bestanden en mappen structuur.

```
poc_gcp_dotnet/
'--- .git
'--- .gitignore
'--- MessageUtil
'--- MessageUtil.csproj
'--- MessageUtilProgram.cs
'--- MessageUtil.sln
'--- MessageUtilTest
'--- MessageUtilTest.csproj
'--- MessageUtilTests.cs
'--- filetrans.sh
```

Listing 5.4: Tree van de bestanden structuur voor de Proof Of Concept op Google Cloud.

Vervolgens moest er connectie worden gemaakt met het gecreëerde project op Google Cloud met Google Cloud CLI. Hiervoor is op de CLI van de gebruiker zijn computer *gcloud init -console only* uitgevoerd. Vervolgens is de wizard gevuld en is het correcte project geselecteerd. Als volgende moest de configuratie van de CI/CD pipeline worden gemaakt. Zoals voorheen aangehaald werkt Google Cloud Code Build met Docker containers om al de gewenste taken uit te voeren. Voor deze POC is er gekozen om een Docker container van Microsoft zelf te gebruiken. Er is gekozen voor een .Net Core SDK container op *DockerHub*. Deze container wordt onderhouden door Microsoft zelf en heeft daarnaast ook uitgebreide documentatie ter beschikking op *GitHub*. Zo moet er geen aangepaste container worden gemaakt die in de toekomst problemen zou kunnen krijgen naarmate de software update. Deze container is gebaseerd op een licht gewicht Linux machine. Hierop

staat dan de SDK van Microsoft om .Net applicaties te testen, compileren, publiceren, enz.

Pipelines voor CI op Google Cloud Code Build worden geconfigureerd met behulp van een YAML-bestand. Voor deze POC zijn er vier verschillende stappen gedefinieerd met elk hun eigen doel. De eerste drie stappen maken allemaal gebruik van de door Microsoft gemaakte Docker container, ‘mcr.microsoft.com/dotnet/ore/sdk:3.1’. De laatste Docker container is een Ubuntu container. In de eerste stap worden de testen in *MessageUtilTest* 5.2 uitgevoerd. De tweede stap compileert de code van de applicatie specifiek voor een ’64 Bit Windows 10’ platform en maakt een folder publish aan. In een derde stap wordt het Bash script ‘tarring.sh’ uitgevoerd dat deze publish map comprimeert. *Figuur 5.5* toont dit Bash script. In een laatste stap wordt het ’filetrans’ *script* 5.3 uitgevoerd. Ook is er een stukje code voorzien dat het gemaakte ‘artifact’ (de gecomprimeerde map) gaat uploaden naar een Storage Bucket op Google Cloud. *Figuur ??* toont deze cloudbuild.yaml. Google Cloud Code Build gaat automatisch bij de overgang van iedere stap naar een andere virtuele machine, het volume waarin wordt gewerkt monteren. Hierdoor zijn er geen speciale stappen of acties nodig om bestanden tussen de verschillende machines uit te wisselen.

```
#!/bin/bash
cd /workspace/MessageUtil/bin/Release/netcoreapp3.1/win10-
x64/ && tar -zcvf messageutil-win10-x64.tar.gz ./publish
/
```

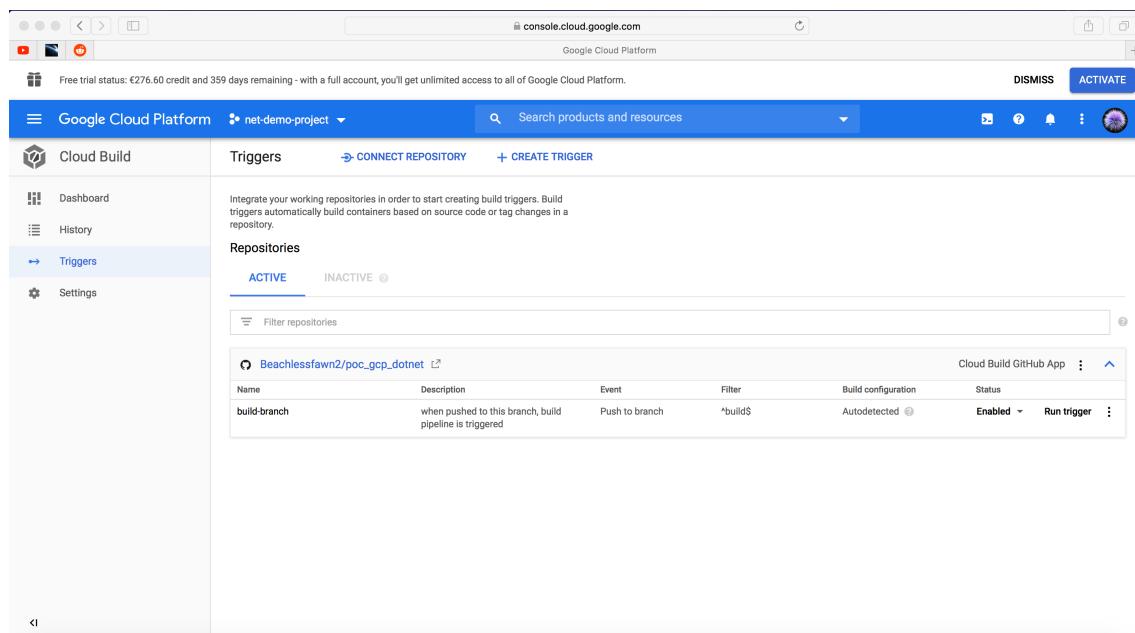
Listing 5.5: Bash script om een bepaalde folder te comprimeren tot een .tar.gz.

```
steps:
- name: 'mcr.microsoft.com/dotnet/core/sdk:3.1'
  entrypoint: 'dotnet'
  args: [ 'test' ]
- name: 'mcr.microsoft.com/dotnet/core/sdk:3.1'
  entrypoint: 'dotnet'
  args: [ 'publish', '-c', 'Release', '-r', 'win10-x64' ]
- name: 'mcr.microsoft.com/dotnet/core/sdk:3.1'
  entrypoint: 'bash'
  args: [ './tarring.sh' ]
- name: 'ubuntu'
  entrypoint: 'bash'
  args: [ './filetrans.sh' ]
  artifacts:
    objects:
      location: 'gs://net-demo-output-bucket/'
      paths: [ '/workspace/MessageUtil/bin/Release/netcoreapp3.1/
              win10-x64/messageutil-win10-x64.tar.gz' ]
```

Listing 5.6: YAML-bestand voor de configuratie van de .Net pijpleiding op Google Cloud.

Nadat al deze bestanden zijn aangemaakt, is deze pijpleiding voor het eerst uitgevoerd. Hiervoor is de Google Cloud CLI gebruikt op de gebruiker zijn computer. Het commando *gcloud builds submit* gaat de volledige Git repository kopiëren naar een tijdelijke Storage Bucket

en start vervolgens de pijpleiding op volgens de configuratie vanuit 'cloudbuild.yaml'. Eenmaal dat deze pijpleiding volledig zonder fouten werkte, is de Git repository op GitHub geplaatst. Vervolgens is er via de online console op Google Cloud een Trigger aangemaakt om deze pijpleiding automatisch te laten uitvoeren bij het updaten van een bepaalde tak op GitHub. Hiervoor is GitHub gelinkt aan Google Cloud. *Figuur 5.3* toont de gemaakte Trigger en ook de specifieke tag die gebruikt is voor de tak op GitHub. Er is gekozen om deze Trigger op een specifieke tak van de GitHub repository te configureren om het aantal keren dat deze pijpleiding wordt uitgevoerd te kunnen minimaliseren. Zo kunnen de ontwikkelaars zelf kiezen wanneer ze de code willen compileren, door naar deze specifieke tak te uploaden. Dit vermindert ook dat er in het wilde weg compilatie commando's worden uitgevoerd door de ontwikkelaars zelf.

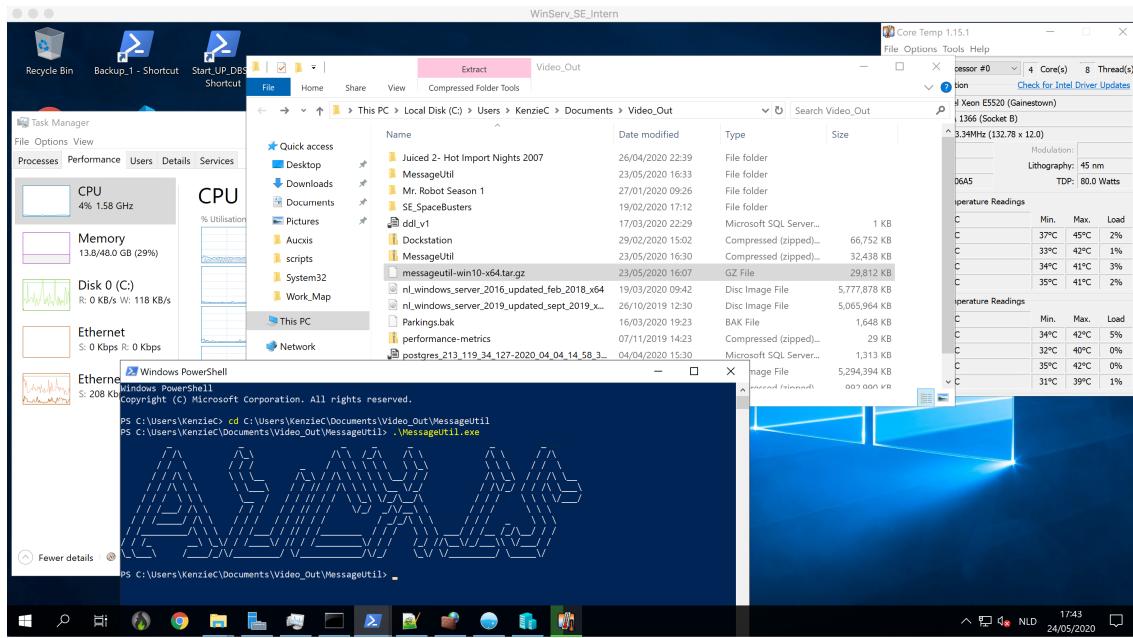


Figuur 5.3: Scherm met Trigger voor een bepaalde pipeline op Google Cloud. Figuur toont het scherm van Google Cloud met een gecreëerde Trigger op de tak 'Build' op GitHub.

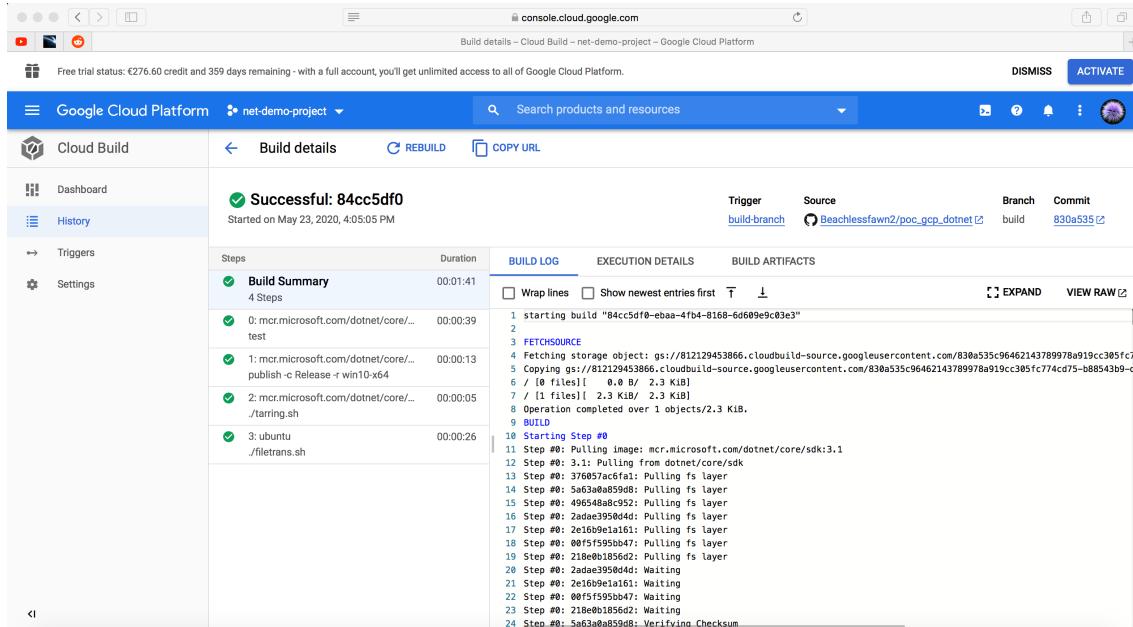
Eenmaal dit gebeurd is kon de trigger worden getest door een commit uit te voeren op GitHub. Vervolgens was het resultaat te testen op de lokale Windows Server 2019. Ook was er een gedetailleerd rapport te zien op de console van Google Cloud zelf. Om de applicatie uit te voeren moest simpelweg het gecomprimeerde bestand worden uitgepakt en uitgevoerd. *Figuur 5.4* toont het resultaat van de uitgevoerde applicatie op de lokale machine. *Figuur 5.5* toont de uitgevoerde compilatie op Google Cloud.

Tot slot is er op de online console van Google Cloud een simpel dashboard te zien. Dit dashboard toont de laatst uitgevoerde compilatie trigger van het project. Ook toont het eventuele fouten, gemiddelde tijden, wat de compilatie heeft doen uitvoeren, commit ID van GitHub, enz. *Figuur 5.6* toont een voorbeeld van zo een dashboard.

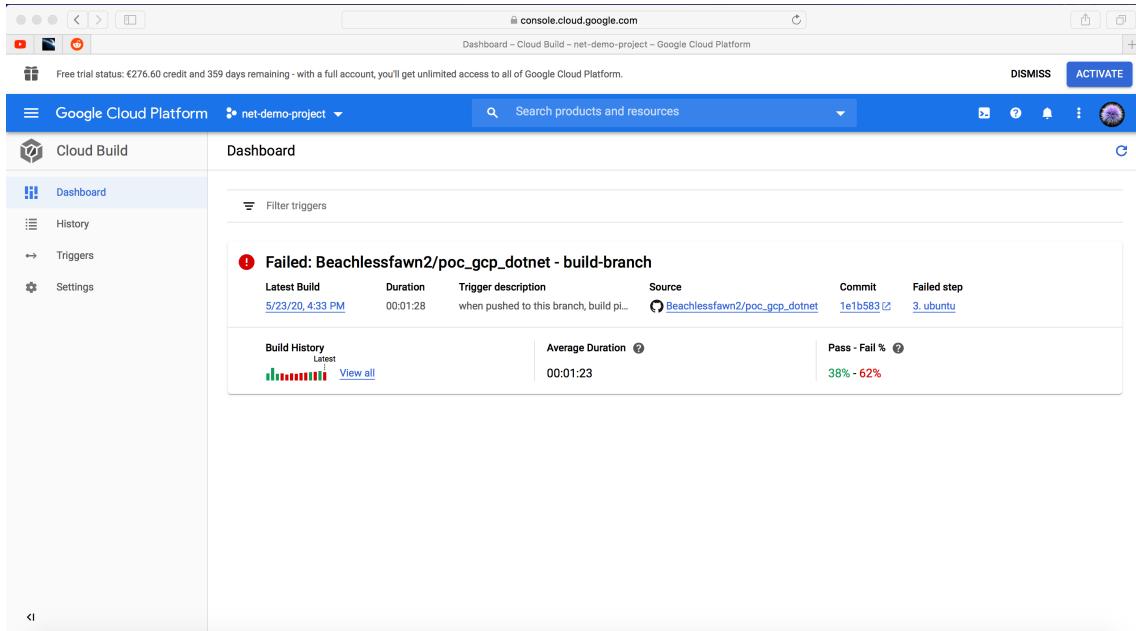
Door deze POC op te stellen is er gebleken dat Google Cloud Build Code gemakkelijk te gebruiken is. Dat is, eenmaal de gebruiker er thuis in is. Er bestaat redelijk wat documentatie over wat allemaal mogelijk is in een 'cloudbuild.yaml'. Helaas is het voor



Figuur 5.4: Resultaat van de applicatie. Figuur toont de uitvoer van de gecompileerde applicatie op een lokale Windows Server 2019. Applicatie is gekopieerd door middel van script 5.3.



Figuur 5.5: Console uitvoer van een pipeline op Google Cloud. Figuur toont het resultaat van de compilatie stappen op Google Cloud. Ook zijn er een aantal andere logs te bekijken op dit scherm.



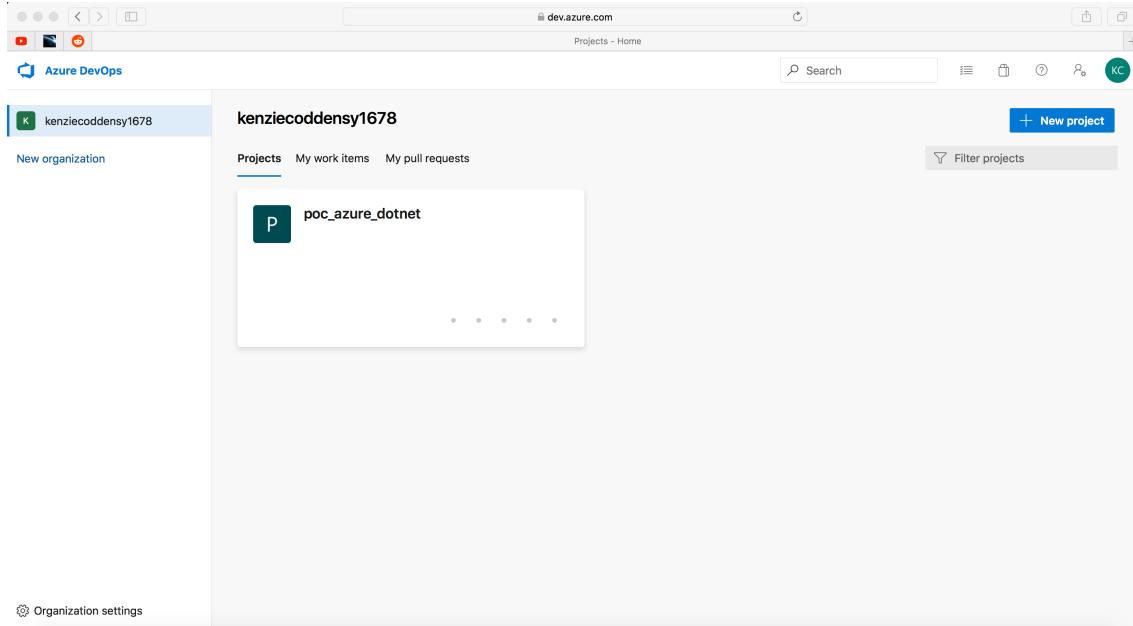
Figuur 5.6: Compileer resultaten dashboard op Google Cloud. Figuur toont een simpel dashboard met gegevens over de compilatie pipeline.

een beginner niet gemakkelijk om te verstaan hoe de verschillende containers met elkaar samenwerken en waar nu alle bestanden staan. Ook is een goede kennis van de containers nodig om te kunnen verstaan wat er allemaal mogelijk is. Dit zorgt ervoor dat het lang kan duren vooraleer er een pipeline operationeel is. Ook is het spijtig dat er niet meer analytics te verkrijgen zijn op Google Cloud Code Build. De gebruiker heeft enkel de analytics van wat er door Google zelf wordt aangeboden. Na het opstellen van deze POC is er ook gebleken dat Google Cloud niet zo duur is. Al het testen en uitproberen heeft slechts 0.40 euro gekost. Het kan interessant zijn om in een later onderzoek te kijken of het niet mogelijk is om de gegenereerde log bestanden van Google Cloud te downloaden en te gebruiken voor eigen visualisatie systemen zoals: Grafana.

5.0.2 Azure DevOps

Naast de POC op Google Cloud is het ook nog interessant om dezelfde opstelling eens te maken in Azure DevOps. Dit om de ervaring en de functionaliteit naast elkaar te kunnen leggen. Voor de POC op Azure DevOps is er een nieuw DevOps project aangemaakt. Dit project heeft de naam ‘poc_azure_dotnet’ gekregen. *Figuur 5.7* toont het aangemaakte project. Net zoals GCP bevatten projecten alle toegewezen producten en services die zijn geconfigureerd. Ook is het gemakkelijk opnieuw te verwijderen en stopt dan ook de aanrekening.

Deze POC vertrekt vanuit dezelfde applicatie als bij GCP. Hiervoor is er een nieuwe map aangemaakt op de gebruiker zijn computer voor de applicatie. Deze is geïnitialiseerd als een Git repository. Hierin zijn dan de bestanden *MessageUtil 5.1 & MessageUtilTest 5.2* voor de .Net applicatie in geplaatst. Ook is er een gitignore aangemaakt. *Figuur 5.7* toont



Figuur 5.7: Hoofd scherm Azure DeVops. Figuur toont een nieuw aangemaakt project op Azure DeVops.

een tree van de bestanden en mappen structuur. Deze repository is dan geüpload naar GitHub.

```
poc_azure_dotnet/
‘--- .gitignore
‘--- MessageUtil
‘--- MessageUtil.csproj
‘--- MessageUtilProgram.cs
‘--- MessageUtil.sln
‘--- MessageUtilTest
‘--- MessageUtilTest.csproj
‘--- MessageUtilTests.cs
```

Listing 5.7: Tree van de bestanden structuur voor de Proof Of Concept op Azure DeVops.

Hierna is er op Azure DeVops een nieuwe pijpleiding aangemaakt. Dit was zeer gemakkelijk te volgen door middel van de online wizard. Eerst moest er gekozen worden wat versiebeheersysteem de gebruiker gebruikt. Hier is er gekozen voor GitHub. Vervolgens vraagt Azure DeVops de gebruiker toestemming voor het uitlezen van de repositories. Tegelijk wordt ook de Azure DeVops applicatie toegevoegd aan de gebruiker zijn GitHub. Na de initialisatie van de Azure DeVops applicatie voor GitHub moest er een repository worden geselecteerd. Hier is er gekozen voor de zojuist gemaakte GitHub repository. Azure DeVops leest deze repository uit en stelt dan op basis van de bestanden de juiste compilatie oplossing voor. Voor deze POC stelde Azure DeVops .Net Core voor als compilatie programma. Er is voor dezen oplossing gekozen. Azure DeVops toont hierna een YAML-bestand, ‘azure-pipelines.yml’, met voor gemaakte stappen. Dit YAML-bestand bevat drie stappen. Een eerste stap die de juiste pakketten installeert op de virtuele machine. In een

tweede stap wordt de applicatie gecompileerd. In de derde stap worden de meegeleverde testen uit *MessagUtilTest* 5.2 uitgevoerd. *Figuur 5.8* toont dit YAML-bestand.

```
# .NET Desktop
# Build and run tests for .NET Desktop or Windows classic
# desktop solutions.
# Add steps that publish symbols, save build artifacts, and
# more:
# https://docs.microsoft.com/azure/devops/pipelines/apps/windows/dot-net

trigger:
- master

pool:
vmImage: 'windows-latest'

variables:
solution: '**/*.sln'
buildPlatform: 'Any CPU'
buildConfiguration: 'Release'

steps:
- task: NuGetToolInstaller@1

- task: NuGetCommand@2
inputs:
restoreSolution: '$(solution)'

- task: VSBUILD@1
inputs:
solution: '$(solution)'
platform: '$(buildPlatform)'
configuration: '$(buildConfiguration)'

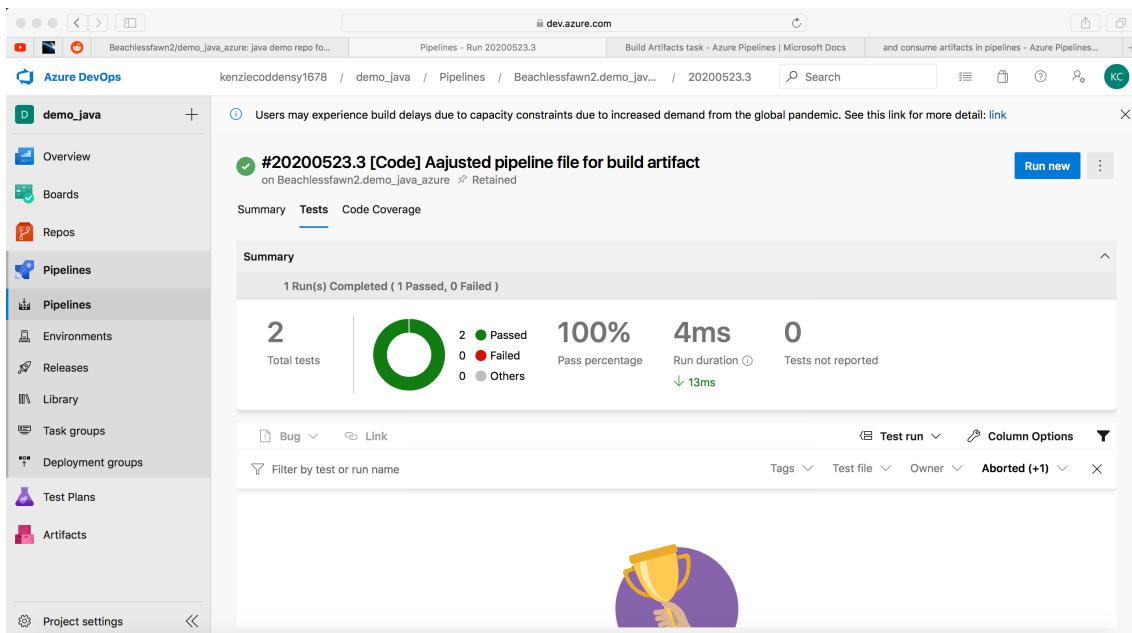
- task: VSTest@2
inputs:
platform: '$(buildPlatform)'
configuration: '$(buildConfiguration)'
```

Listing 5.8: YAML-bestand voor de configuratie van de .Net pipeline op Azure DevOps.

Verwacht was dat dit een werkend vertrekpunt is voor de pipeline. Zeker omdat deze applicatie met een Microsoft specifieke taal is gemaakt, met behulp van het .Net framework dat ook Microsoft specifiek is. In eerste instantie leek deze pipeline te werken. Maar na het bekijken van de logbestanden is er vastgesteld dat de testen niet werden uitgevoerd. Er is gezocht naar een oplossing voor dit probleem. Er is geen duidelijke oplossing gevonden

voor de huidige configuratie. De documentatie over deze modules was zeer onduidelijk en moeilijk te begrijpen. Zeker vanuit het standpunt van een beginner. In dit opzicht is er gekozen om opnieuw te beginnen aan de hand van een andere *handleiding* op Azure. Deze gebruikt een voor gemaakte Windows virtuele machine met .Net core geïnstalleerd. Over deze module is uitgebreide documentatie te vinden. Daarnaast zijn er door Microsoft ook voorbeelden ter beschikking. Dit was dan ook het perfecte vertrekpunt.

Dit nieuw YAML-bestand bestaat uit zes stappen. In een eerste stap wordt de ‘MessageUtil’ applicatie volledige herbouwt. Deze staat dan klaar voor verdere stappen. In de tweede stap wordt de applicatie gecompileerd. De derde stap voert de meegeleverde testen uit. Ook publiceert het de resultaten op Azure DevOps. *Figuur 5.8* toont een voorbeeld van uitgevoerde testen. De vierde stap compileert een uitrol versie voor het ‘64 bit Windows 10’ platform in een map, publish. In de vijfde stap wordt er een PowerShell script uitgevoerd. Dit Script comprimeert de map publish. *Figuur 5.9* toont het ‘zipping.ps1’ script. In een laatste stap wordt deze gecomprimeerde map gepubliceerd als een artifact zodanig dat de applicatie klaarstaat om in een latere stap te worden uitgerold. *Figuur 5.10* toont het verbeterde ‘azure-pipelines.yml’. Vervolgens is deze code toegevoegd op GitHub.



Figuur 5.8: Test-pass dashboard Azure DevOps. Figuur toont een simpel dashboard met gegevens over de uitgevoerde testen.

```
$compress = @{
Path= "d:\a\1\s\MessageUtil\bin\Release\netcoreapp3.1\win10
-x64\publish\*"
CompressionLevel = "Fastest"
DestinationPath = "d:\a\1\s\MessageUtil.zip"
}
Compress-Archive @compress
```

Listing 5.9: PowerShell script dat een map compresseert. Wordt gebruikt om alle .Net Framework afhankelijkheden mee te verpakken.

```
trigger:
- master

pool:
vmImage: 'windows-latest'

variables:
buildConfiguration: 'Release'
runtimeIdentifier: 'win10-x64'

steps:
- task: DotNetCoreCLI@2
inputs:
command: 'restore'
restoreDirectory: '$(System.DefaultWorkingDirectory)'
#packDirectory: '$(System.DefaultWorkingDirectory)'
displayName: 'DotNet Restore Project'

- task: DotNetCoreCLI@2
inputs:
command: 'build'
arguments: '--configuration $(buildConfiguration)'
displayName: 'DotNet Build $(buildConfiguration)'

- task: DotNetCoreCLI@2
inputs:
command: 'test'
projects: '**/*Test/*.csproj'
arguments: '--configuration $(buildConfiguration) --collect "CodeCoverage"'
displayName: 'DotNet Test Build'

- task: DotNetCoreCLI@2
inputs:
command: 'publish'
publishWebProjects: false
arguments: '--configuration $(buildConfiguration) -r $(runtimeIdentifier)'
zipAfterPublish: false
displayName: 'DotNet Publish Build'

- task: PowerShell@2
inputs:
targetType: 'filePath'
filePath: $(System.DefaultWorkingDirectory)\zipping.ps1
```

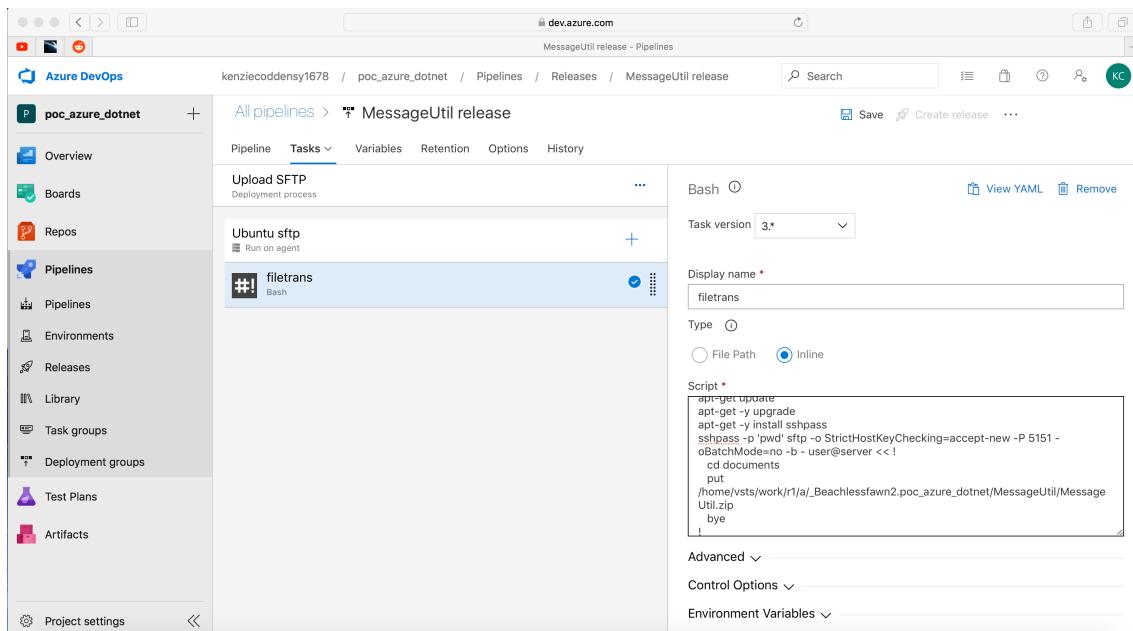
```

errorActionPreference: 'stop'
displayName: 'DotNet Zip Publish'

- task: PublishPipelineArtifact@1
  inputs:
    targetPath: $(System.DefaultWorkingDirectory)\MessageUtil.zip
    artifactName: MessageUtil
    displayName: 'Upload Zip'
  
```

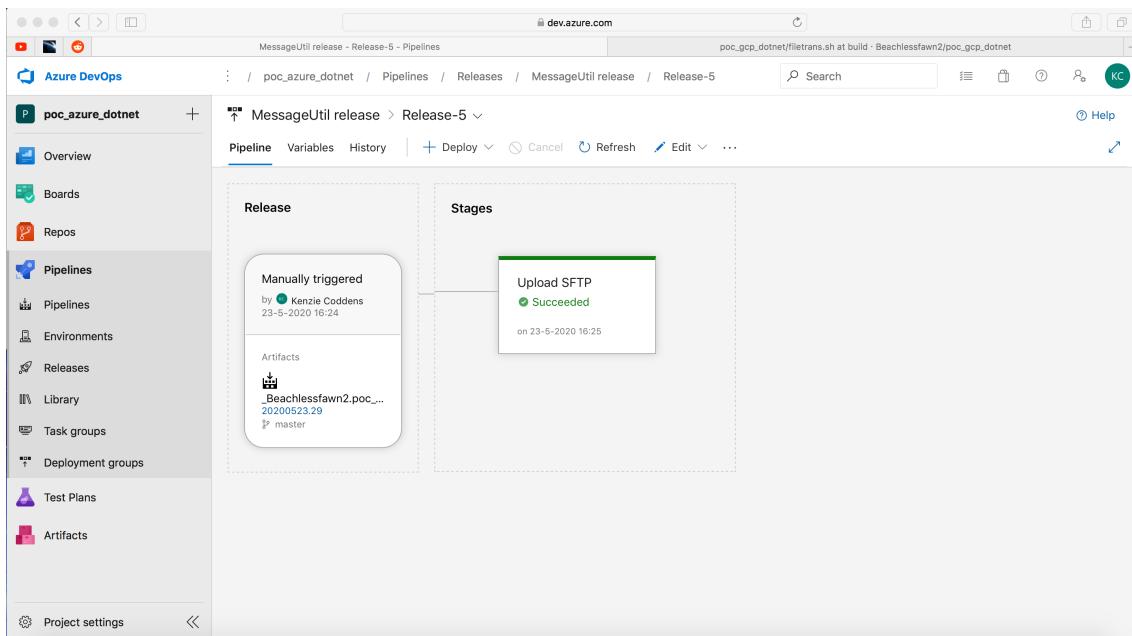
Listing 5.10: Verbeterd YAML-bestand voor de configuratie van de .Net pipeline op Azure DevOps.

Het CI-gedeelte van de pipeline werkt met deze code. Voor het CD-gedeelte moet er een nieuwe release pipeline worden gemaakt. Op Azure DevOps is er een nieuwe release gemaakt bestaande uit een Linux Ubuntu machine. Deze virtuele machine download het gemaakte artifact en verzend dit bestand naar de lokale Windows Server. Deze virtuele machine voert het script uit *figuur 5.3* uit. Het instellen van deze release pipeline is zeer gemakkelijk door de online wizard. *Figuur 5.9* toont het instellen van de virtuele machine. Ook zijn er talloze opties voor het instellen van toestemmingen, rechten, controles, enz. Dit kan interessant zijn in een productie omgeving. Deze POC laat dit buiten beschouwing. *Figuur 5.10* toont de volledige pipeline.

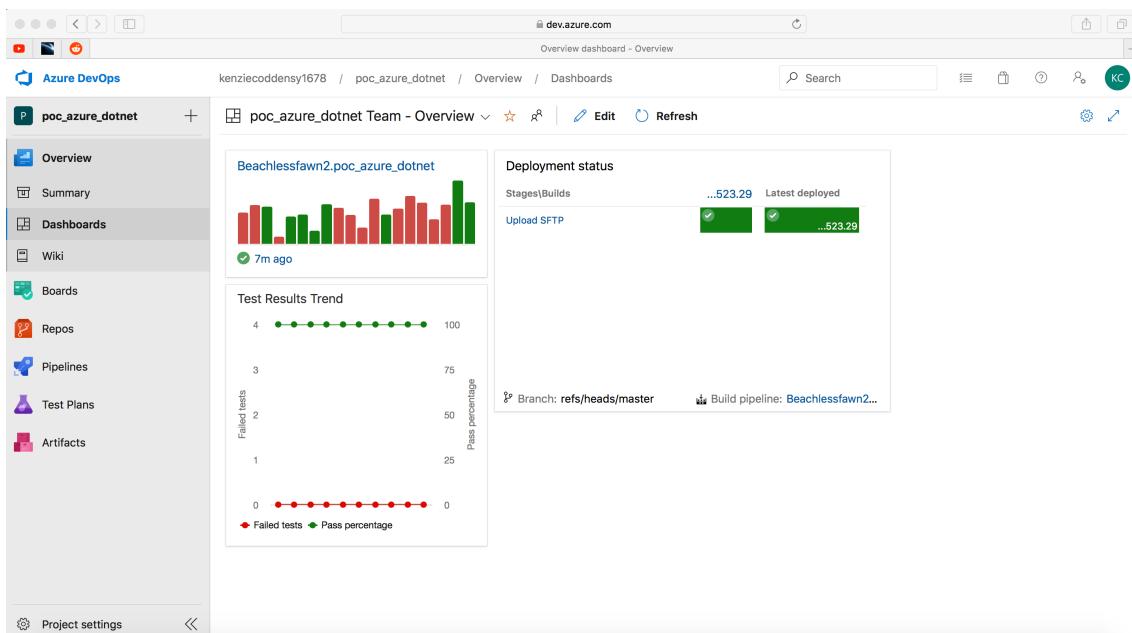


Figuur 5.9: Release pipeline configuratie. Figuur toont hoe een release pipeline op Azure DevOps wordt geconfigureerd. In dit geval wordt er een Ubuntu machine geconfigureerd met een script.

Azure DevOps heeft ook zeer goede analytics. Voor deze POC zijn er een aantal voorbeelden gemaakt. Net zoal Google Cloud heeft Azure ook Dashboards. *Figuur 5.11* toont een overzicht van het project.



Figuur 5.10: Grafische weergave Azure DeVops pipeline. De figuur toont een kort overzicht van de totale pipeline op Azure DeVops.



Figuur 5.11: Project overzicht dashboard Azure DeVops. Figuur toont een simpel dashboard met gegevens over de compilatie pipeline.

Uit deze POC is gebleken dat Azure DevOps veel minder complex is om te configureren dan Google Cloud Code Build. Dit komt door de gemakkelijk te gebruiken wizards die de gebruiker door de configuratie heen gidsen. Zo voelt de complexiteit van het configureren van een pipeline minder zwaar aan. Uit deze POC is gebleken dat de standaardoplossing die Azure DevOps voorstelt niet altijd even gemakkelijk is om aan te passen naar de gebruiker zijn noden. Deze POC heeft ook aangetoond dat de documentatie van Azure DevOps complex kan zijn. Zelfs na meerdere projecten te hebben getest, zijn sommige zaken nog niet helemaal duidelijk. Azure DevOps biedt zeer uitgebreide hulpmiddelen aan om allerlei data te visualiseren. Deze dashboards zijn gemakkelijk te creëren. Ook zijn deze duidelijk en overzichtelijk. Dit in tegenstelling tot Google Cloud. In vergelijking met Azure DevOps is Google Cloud gemakkelijker om aangepaste pipelines te maken. Dit omdat het intuïtiever is om de configuratie YAML-bestanden te maken. Ook omdat Google Cloud Code Build met Docker containers werkt. Dit zorgt ervoor dat compiler machines veel uitgebreider kunnen worden aangepast naar de noden van de gebruiker.

Benaderd vanuit het Microsoft ecosysteem is Azure DevOps een zeer goede en logische keuze. Zeker omdat een hele hoop producten en services, die anders aanvullende kosten hebben, vrij te gebruiken zijn. Ook zijn alle hulpmiddelen om software te ontwikkelen volledig geïntegreerd in het platform. Gebruikers die buiten dit ecosysteem werken kijken beter naar Google Cloud in samenwerking met andere hulpmiddelen en oplossingen.

6. Conclusie

Er bestaan veel Cloud platformen. Dit onderzoek heeft duidelijk aangetoond dat niet ieder platform even geschikt is om CI/CD op te implementeren. Al vroeg in het onderzoek zijn IBM Cloud, Oracle Cloud en Digital Ocean buiten beschouwing gevallen. Deze Cloud platformen bevatten misschien wel de mogelijkheid om een pipeline te configureren. Maar dit werd meestal gedaan door middel van het Tekton framework. Dit framework maakt het mogelijk dat ieder Cloud platform dat Kubernetes ondersteund, kan worden gebruikt om CI/CD te implementeren. Alleen zijn er dan geen dashboards beschikbaar met specifieke informatie over de resultaten van de pipeline. Het zou interessant zijn om in een later onderzoek dit framework eens te bekijken.

Van de twee alternatieve kandidaten die overbleven was Google Cloud veruit de meest geschikte kandidaat. Dit platform is in het begin moeilijk in gebruik. Eenmaal de eerste pipeline is geconfigureerd, wordt het platform makkelijker in gebruik. Door het gebruik van Docker containers kan Google Cloud gemakkelijk worden aangepast naar de gebruiker zijn noden. Ook zorgen de containers ervoor dat het configureren van een pipeline intuïtief is. De documentatie die ter beschikking is van Google, is duidelijk en gemakkelijk te volgen. Er bestaan genoeg voorbeelden om als basis te gebruiken voor een opstelling. Ook is gebleken uit de gebruiksvriendelijkheid test dat Google Cloud in vergelijking met Amazon AWS gemakkelijker te configureren is. Tabel (verwijzing) toont de gespendeerde tijd voor de Cloud platformen. Google Cloud is een goed alternatief voor Azure DevOps. De analytics zijn minder goed in vergelijking met Azure DevOps. Deze zijn ook minder visueel. Een toekomstig onderzoek kan onderzoeken of er systemen bestaan om betere visuele dashboards te maken. Er kan bijvoorbeeld worden onderzocht of Grafana een oplossing kan zijn.

Ook is gebleken uit de gebruiksvriendelijkheid test dat Amazon AWS slachtoffer is van zijn

eigen succes. Er bestaan te veel producten en services waardoor het moeilijk is om de juiste producten te selecteren. Ook de documentatie kan worden verbeterd. Een beginner zal echt problemen hebben bij het opstellen van zijn eerste pipeline door deze documentatie. Een plus punt is de online console. Deze console is even gemakkelijk in gebruik als de console van Azure DeOps. Dit is iets wat Google Cloud in de toekomst zou kunnen verbeteren.

De ‘Proof Of Concept’ op Google Cloud is geslaagd. Er is succesvol een pipeline opgesteld die automatisch wordt uitgevoerd bij het updaten van de broncode. Ook wordt de applicatie geüpload naar een lokale omgeving. Deze is uitvoerbaar en kan getest worden in een lokale omgeving.

Dit onderzoek heeft aangetoond dat binnen het ecosysteem van Microsoft, Azure DeOps de beste keuze is. Dit platform heeft meer mogelijkheden voor CI/CD dan Google Cloud. Ook heeft het meer hulpmiddelen om projecten te begeleiden. Er is geen nood meer om nog andere hulpmiddelen van derden te gebruiken. Het nadeel van Azure DeOps is dat niet al deze hulpmiddelen beschikbaar zijn bij een basis Azure DeOps licentie. Het kost meer om al deze hulpmiddelen ter beschikking te hebben. Daarom is Google Cloud het beste alternatief voor gebruikers die niet binnen dit ecosysteem zitten. Google Cloud evenaart goed de functionaliteit van Azure DeOps. Het platform is zelfs beter aan te passen naar specifieke noden. Beide platformen hebben ook goede ondersteuning voor andere hulpmiddelen voor bijvoorbeeld het uitrollen van de applicatie naar mobiele systemen.

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

Software release management kan op verschillende manieren geïmplementeerd worden. Dit zijn vaak complexe en zeer use case specifieke omgevingen. Ook testen en debuggen van software en infrastructuur zijn belangrijk voor het afleveren van kwalitatieve producten. Een vraag die hierbij opduikt is of dit op lokale infrastructuur moet gebeuren of op cloud platformen die IaaS (Infrastructure as a Service), TaaS (Testing as a Service) of SaaS (Software as a Service) aanbieden.

Der bestaan al tientallen papers over de performantie, flexibiliteit, enz. in een cloud omgeving. Ook over software release management zijn er al talloze papers geschreven. Des ondanks is het toch nog interessant om dit voor een specifieke use case te bekijken. Aucxis heeft al een software release omgeving op Azure. Ze hebben echter geen onderzoek gedaan naar andere cloud platformen of oplossingen. Deze paper is in de eerste plaats bedoelt om voor Aucxis een duidelijk beeld te scheppen over de mogelijkheden.

Deze paper zal proberen in detail duidelijkheid te scheppen over wanneer een cloud platform een goede keuze is en wanneer niet, wat de beste tools zijn, hoe het zit met de gebruiksvriendelijkheid en de prijs. Ook de performantie is niet onbelangrijk. Deze paper zal zich ook afvragen hoe data privacy kan gecontroleerd en geïmplementeerd worden. Op het einde van deze paper zal er een conclusie gemaakt worden over welke optie het best

past binnen de use case van Aucxis.

A.2 Literatuurstudie

A.2.1 Conventional Software Testing Vs. Cloud Testing

Dit artikel (Katherine & Alagarsamy, 2012) snijdt oppervlakkig aan wat pijnpunten kunnen zijn voor testen van software in een cloud platform. Hier gebruiken ze een web applicatie als voorbeeld. Het artikel stelt een aantal punten voor, waarop getest kan worden. Dit zijn de traditionele test cases. Functionaliteit testen, gebruiksvriendelijkheid testen, interface testen, compatibiliteitstesten, performantie testen en tot slot veiligheid en privacy testen. Het artikel stelt een aantal uitdagingen voor bij lokale omgevingen. Dit gaat over de kost, over het onderhoud ervan, hoe eenzijdig een lokale omgeving is en voor ieder project een nieuwe omgeving gebouwd moet worden en het feit dat het geen accurate weergave is van de werkelijke omgevingen waarin de software zal draaien.

Verder legt het artikel kort uit wat voor mogelijke cloud oplossingen er op dat moment zijn. Eerst moet er onderscheid gemaakt worden in hoe een cloud platform uitgerold kan worden. Er bestaat enerzijds de publieke cloud (Google Cloud Platform, AWS, Azure, DigitalOcean) en anderzijds een privé cloud. De privé cloud is een lokale opstelling die beschikbaar is over het internet naar andere gebruikers. Ook bestaat er iets zoals een hybride cloud. Hierna wordt er dan nog onderscheid gemaakt tussen welke services deze platformen kunnen aanbieden. Dit artikel beschrijft er drie. SaaS (service as a Service), PaaS (Platform as a Service), IaaS (Infrastructure as a Service). Het artikel maakt toch een onderscheid van het traditionele testen. Aangezien het in de cloud mogelijk is om de applicatie te testen op load, stress en capaciteit.

Tot slot stelt dit artikel belangrijke uitdagingen aan het licht. Zo is de beveiliging van de data die ontvangen, verstuurd of bewaard worden op een cloud platform belangrijk. Ook zijn er over alle platformen heen weinig tot geen standaarden vastgelegd over zowel de performantie van de systemen als de beschikbaarheid op vlak van aanbod. Het zijn juist deze uitdagingen die belangrijk kunnen zijn voor de onderzoeks vragen.

A.2.2 Software Testing Based on Cloud Computing

In dit artikel (Jun & Meng, 2011) wordt er opnieuw in detail beschreven wat de verschillende platform mogelijkheden zijn zoals IaaS, PaaS, SaaS. Het artikel probeert ook een definitie te geven aan testen op cloud platformen. Tevens geeft het artikel ook een aantal redenen waarom cloud testen een stuk beter zou zijn dan het testen in lokale omgevingen. Deze komen grotendeels overeen met het vorige artikel. In dit artikel wordt er ook besproken dat beveiliging een groot probleem kan zijn. Juist zoals het vorige artikel wordt er gesteld dat het een echte uitdaging is om test datasets in de cloud te gebruiken aangezien deze meestal afkomstig zijn van een klant. Het artikel bespreekt ook een aantal mogelijkheden om met de cloud te verbinden en testomgevingen te configureren. Het

artikel bespreekt vooral virtualisatie.

Dit artikel bevestigt deels het vorige artikel. Het geeft wat detail en inzicht in cloud testen. Dit artikel sluit aan met de onderzoeks vragen en geeft richting in probleem gebieden.

A.2.3 Benchmarking in the Cloud: What It Should, Can, and Cannot Be

Zomaar willekeurig testen of experimenteren uitvoeren is meestal geen goed idee. Er is nood aan een goed gedefinieerde methode om deze testen uniform uit te voeren. Dit artikel (Folkerts e.a., 2013) beschrijft in extreem detail hoe een cloud platform het best getest kan worden. Er wordt beschreven wat de valkuilen zijn bij performantietesten van een cloud platform. Zo wordt het testen van een lokale omgeving vergeleken met het testen van een cloud omgeving. Dit is een hele uitdaging aangezien de hardware van een cloud platform meestal verschilt en niet hetzelfde is. Het artikel beschrijft het testen van een cloud platform aan de hand van een aantal use cases. Het artikel gebruikt hiervoor use cases die schaalbaar zijn en in pieken benaderd worden. Ook beschrijft het artikel dat het belangrijk is om goed te definiëren wat er allemaal getest moet worden over de verschillende platformen heen.

Dit artikel biedt een gedetailleerd inzicht in het opstellen van benchmarks voor cloud omgevingen en zal een belangrijke leidraad vormen voor het opstellen van de experimenten.

A.2.4 When to Migrate Software Testing to the Cloud?

Wanneer moet er gedacht worden om naar een cloud omgeving te migreren? Dit artikel (Parveen & Tilley, 2010) beschrijft vanaf wanneer het nuttig is om naar de cloud te migreren. Ook beschrijft het artikel kort wat de ervaring was bij een migratie. Dit artikel is interessant omdat het kort een inzicht geeft in wanneer het nuttig en efficiënt is om naar een cloud te migreren. Dit is interessant omdat dit aansluit bij de probleemstelling of een cloud omgeving voor testen nu zoveel beter kan zijn dan een lokale omgeving.

A.3 Methodologie

Een groot deel van de onderzoeks vragen zullen beantwoord worden door onderzoekswerk en vergelijkingen. Zo zal deze paper in detail bespreken welke cloud platformen er bestaan en wat de mogelijk plannen (tarieven en voor gedefinieerde configuraties) zijn. De verschillende platformen zullen op een duidelijke manier naast elkaar gelegd worden en vergeleken worden. Ook zal er gekeken worden naar software release tools. Op basis hiervan zal er dan een keuze gemaakt worden welke platformen er in aanmerking komen voor een proof of concept.

Ook zal deze paper methodes beschrijven en testen door middel van experimenten wat betreft het behouden van data privacy. Deze paper zal bijvoorbeeld een experiment uitvoeren met een proxy (een tunnel met encryptie naar het datacenter) om de gebruiksvriendelijkheid hiervan te testen.

Na het onderzoek zal er een proof of concept opgezet worden met beste alternatief. Er zal getracht worden om de huidige omgeving van Aucxis zo goed mogelijk te benaderen in functionaliteit. De bedoeling is om dezelfde tools te gebruiken om de omgevingen te monitoren (bijvoorbeeld: een Docker image voor dezelfde configuratie en Telegraf en Grafana voor monitoring). Ook zal er getracht worden om dezelfde testen te gebruiken. Dit alles zal over een bepaalde periode draaiende gehouden worden waarna alle resultaten gebundeld zullen worden. Hierbij zitten ook een aantal subjectieve waarnemingen aangezien er ook onderzoek zal gedaan worden naar gebruiksvriendelijkheid. In deze proof of concept zal er een alternatief platform vergeleken worden met het huidige systeem.

A.4 Verwachte resultaten

A.4.1 Vergelijking van platformen

Er wordt verwacht dat de huidige cloud omgeving vanuit de use case van Aucxis de beste oplossing is, zeker op vlak van gebruiksvriendelijkheid en efficiëntie. Ondanks dit wordt er verwacht dat de alternatieven een even goede oplossing zullen aanbieden. Deze zullen waarschijnlijk niet de meest gebruiksvriendelijkste of goedkoopste oplossingen zijn. Voor de tools voor software release management wordt er verwacht dat er gelijkaardige alternatieven aan de huidige tools vanuit de use case gevonden worden.

A.4.2 Proof of concept

Er wordt verwacht dat er een werkende alternatieve omgeving opgezet zal worden die de huidige functionaliteit benaderd. Er wordt verwacht dat de performantie van dit platform op zen minst gelijkaardig is aan de huidige use case. Er wordt verwacht dat de gebruiksvriendelijkheid en de kost verbeteren.

A.4.3 beveiliging experiment

Voor dit experiment zijn er gemengde verwachtingen. Vooral op het vlak van tijdrovende configuraties. Er wordt verwacht dat een proxy het meest flexibel is en het meest gebruiksvriendelijk, zeker als het vergeleken wordt met encryptie of andere tools.

A.5 Verwachte conclusies

A.5.1 Vergelijking van platformen

Het is moeilijk om een conclusie te voorspellen over welk cloud platform het beste uit de vergelijkingen zal komen. Ook is het moeilijk te voorspellen wat de beste tools zullen zijn. Er wordt wel verwacht dat er een degelijk alternatief voor de huidige oplossing gevonden zal worden.

A.5.2 Proof of concept

De conclusie zal voor dit experiment zeer duidelijk zijn. Deze zal afhangen van de werking van het alternatief. Is het alternatief sneller, gebruiksvriendelijker, enz. dan zal deze conclusie positief zijn. Anders niet.

A.5.3 beveiliging experiment

Om de data privacy te garanderen wordt er verwacht dat een proxy de beste en meest doenlijke oplossing zal zijn. Het valt moeilijk te zeggen of andere tools of methodes beter zullen presteren.

Bibliografie

- Barshefsky. (2005). *Methodes and systems for software release management*. 2005/0216486. DUFT SETTER OLLILA & BORNSEN LLC 2060 BROADWAYSUTE300BOULDER, CO 80302(US).
- Copeland, M., Soh, J., Puca, A., Manning, M. & Gollob, D. (2015, oktober 8). *Microsoft Azure*. Springer-Verlag GmbH. Verkregen van https://www.ebook.de/de/product/25127919_marshall_copeland_julian_soh_anthony_puca_mike_manning_david_gollob_microsoft_azure.html
- Debroy, V., Miller, S. & Brimble, L. (2018). Building lean continuous integration and delivery pipelines by applying DevOps principles: a case study at varidesk. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering - ESEC/FSE 2018*. ACM Press. doi:10.1145/3236024.3275528
- Ebert, C., Gallardo, G., Hernantes, J. & Serrano, N. (2016). DevOps. *IEEE Software*, 33(3), 94–100. doi:10.1109/ms.2016.68
- Folkerts, E., Alexandrov, A., Sachs, K., Iosup, A., Markl, V. & Tosun, C. (2013). Benchmarking in the cloud: what it should, can, and cannot be. *Springer-Verlag Berlin Heidelberg*. TPCTC 2012, LNCS 7755.
- Jackson, K. R., Ramakrishnan, L., Muriki, K., Canon, S., Cholia, S., Shalf, J., ... Wright, N. J. (2010). Performance analysis of high performance computing applications on the amazon web services cloud. In *2010 IEEE Second International Conference on Cloud Computing Technology and Science*. IEEE. doi:10.1109/cloudcom.2010.69
- Jun, W. & Meng, F. (2011). Software Testing Based on Cloud Computing. In *2011 International Conference on Internet Computing and Information Services*. IEEE. doi:10.1109/icicis.2011.51
- Katherine, M. & Alagarsamy, D. K. (2012). Conventional software testing vs cloud testing. *International Journal Of Scientific & Engineering Research*, 3(9).

- Lahtela, A. & Jantti, M. (2011). Challenges and problems in release management process: A case study. In *2011 IEEE 2nd International Conference on Software Engineering and Service Science*. IEEE. doi:10.1109/icsess.2011.5982242
- Meyer, M. (2014). Continuous integration and its tools. *IEEE Software*, 31(3), 14–16. doi:10.1109/ms.2014.58
- Parveen, T. & Tilley, S. (2010). When to migrate software testing to the cloud? *Third International Conference on Software Testing, Verification, and Validation Workshops*. DOI 10.1109/ICSTW.2010.77 IEEE. doi:10.1109/ICSTW.2010.77
- van der Hoek, A. & Wolf, A. L. (2002). Software release management for component-based software. *Software: Practice and Experience*, 33(1), 77–98. doi:10.1002/spe.496