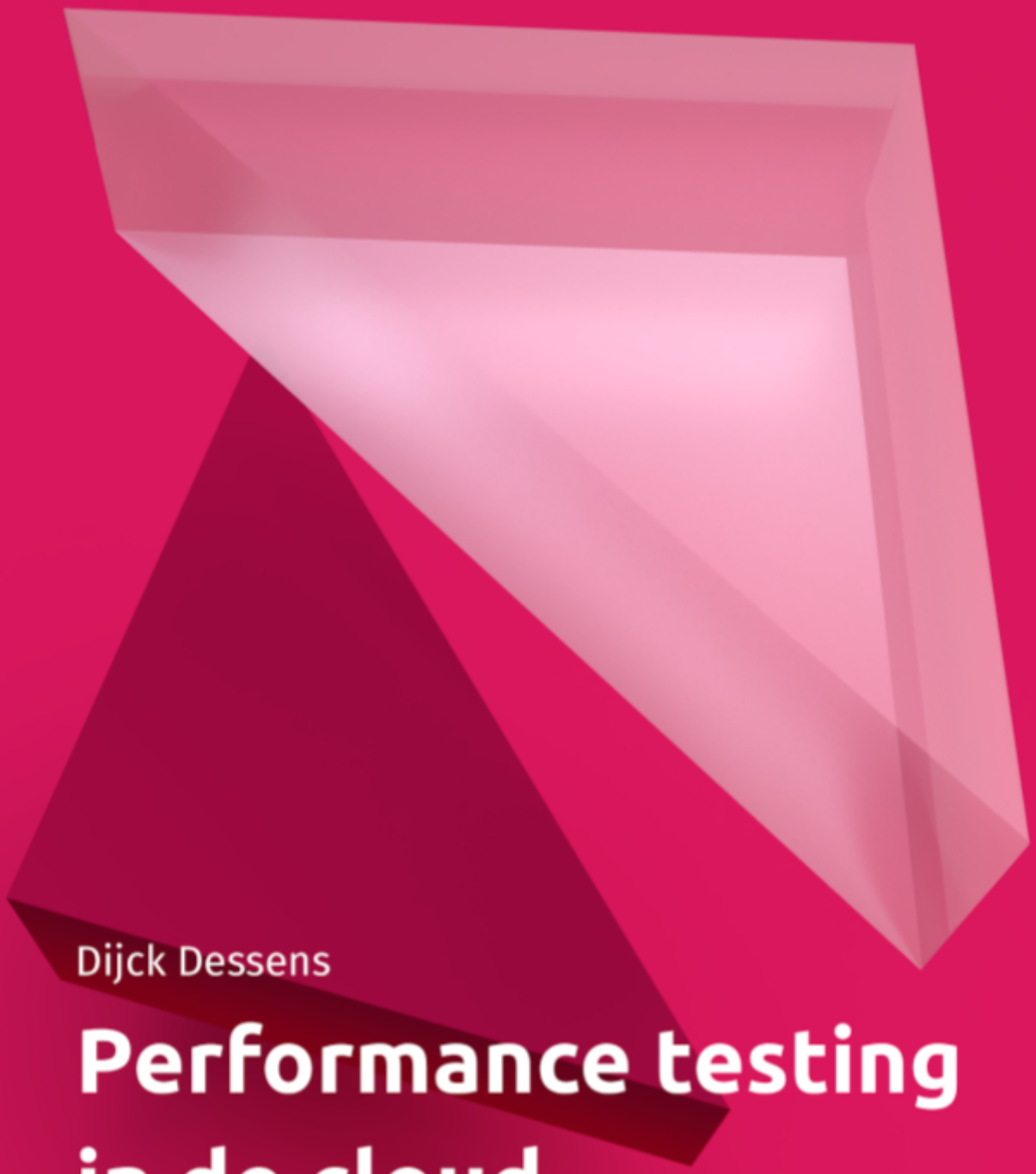
Fonto



Dijck Dessens

Performance testing in de cloud

Scriptie

Uitvoeren van een onderzoek en het realiseren van een prototype omtrent de test runner bij Fonto.

Student
Naam: Dijck Dessens
Email:
Opleiding: HBO-ICT
Jaar: 2020/2021
Onderwijsinstelling: De Haagse Hogeschool

Bedrijfsinformatie
Naam: Dhr. Bert Willems
Email:
Bedrijf: Fonto Group B.V.
Locatie: Rijswijk

Begeleidend examinator
Naam: Rob van der Krog
Email:

Voorwoord

De scriptie die voor u ligt is opgesteld ter afsluiting van mijn HBO-ICT opleiding aan de Haagse Hogeschool.

In de afgelopen vier maanden heb ik in het kader van schaalbaarheid onderzoek gedaan bij Fonto. Hierbij heb ik onderzocht hoe het testsysteem van Fonto op een schaalbare manier kan worden ingericht.

De scriptie die voor u ligt zou niet tot stand zijn gekomen zonder de uitmuntende begeleiding en ondersteuning vanuit Fonto. Naast de behulpzame en meedenkende collega's binnen Fonto, wil ik mijn begeleiders genaamd Bert Willems en Marijn van Butselaar bedanken voor de constante ondersteuning tijdens mijn afstudeerperiode. Mede door hun heb ik mijzelf de afgelopen vier maanden continu kunnen ontwikkelen.

Daarnaast bedank ik ook Rob van der Krogt, voor de begeleiding vanuit de Haagse Hogeschool en bedank ik Ad de Rijke voor de expert begeleiding tijdens het afstudeertraject.

Tot slot wens ik ieder veel leesplezier met mijn scriptie.

Dijck Dessens
Delft, 4 juni 2021

Inhoudsopgave

1.0 Inleiding

In de scriptie die voor u ligt is onderzoek gedaan naar de mogelijkheden op het gebied van schaalbaarheid omtrent het testsysteem van Fonto. Deze scriptie is opgesteld ter afronding van de HBO-ICT opleiding aan de Haagse Hogeschool.

Binnen de eerst volgende hoofdstukken is het bedrijf Fonto tot op meer detail beschreven waarbij de aanleiding, probleemstelling, doelstelling en context van de afstudeeropdracht toegelicht worden. De opvolgende hoofdstukken bevatten deskresearch, waarna het proces van het uitgevoerde onderzoek beschreven is. Na de afronding van het onderzoek zijn de opgestelde software architectuur en het prototype toegelicht. Tot slot is een evaluatie en adviesrapport opgesteld, waarna de vanuit de opleiding opgestelde beroepstaken bewezen worden.

2.0 Over Fonto Group B.V.

Om een beter beeld te krijgen van het bedrijf waar het project is uitgevoerd, worden in hoofdstuk 2 de organisatie, kernactiviteiten, teams, bedrijfsstructuur en probleemdomein behandeld.

2.1 Organisatie

Liones is een bedrijf die zich focust op intelligente content creatie. Hierbij bieden ze consultancy aan op het gebied van XML authoring en geven ze training omtrent Fonto. Het bedrijf is gevestigd in Rijswijk en beschikt momenteel over circa 50 medewerkers. Liones is opgericht in 1999 en beschikt onder andere over back-end ontwikkelaars, front-end ontwikkelaars, UX designers, marketing experts en interaction designers.

Fonto is een product dat in 2013 ontwikkeld is door Liones. Dit product is een client-side editor die het structureren van content eenvoudig en toegankelijk maakt voor gebruikers die niet beschikken over kennis van XML. Fonto richt zich tot meerdere branches van de samenleving. Zo verstrekt Fonto hun applicatie aan onder andere de zorg, luchtvaartmaatschappijen en verschillende wetgevende organisaties. Enkele voorbeelden hiervan zijn Smith+Nephew, SunExpress en Stabilimenti Tipografici Carlo Colombo.

Een belangrijk aspect van Fonto is de hechte bedrijfscultuur. Deze cultuur wordt gevormd door medewerkers die bereid zijn elk ander te helpen. Daarnaast is Fonto gecertificeerd door het bedrijf Great Place To Work. Dit bedrijf heeft Fonto beoordeeld op basis van enquêtes onder de medewerkers, waarbij de enquêtes gericht waren op prestatie, geloofwaardigheid, respect, eerlijkheid, trots en kameraadschap. Dit resulteerde in een gemiddelde score van 100%.

2.2 Kernactiviteiten & teams

De activiteiten van Fonto hebben betrekking tot hun applicatie, genaamd Fonto. Hierbij werken ze aan nieuwe functionaliteiten en testen ze de applicatie. Daarnaast ontwikkelen ze nieuwe tools voor de Fonto applicatie, zoals een tool om de geschiedenis bij te houden of een tool waarmee gebruikers elkaars werk kunnen controleren.

Fonto houdt zich tevens ook bezig met de marketing en sales omtrent hun product. Fonto beschikt over verschillende teams. Deze teams zijn op te delen in product development -en ondersteunende teams. De product development teams zijn gericht op het onderhouden en uitbreiden van de Fonto applicatie. De ondersteunende teams zijn verantwoordelijk voor de standaard bedrijfsprocessen zoals marketing en financiën.

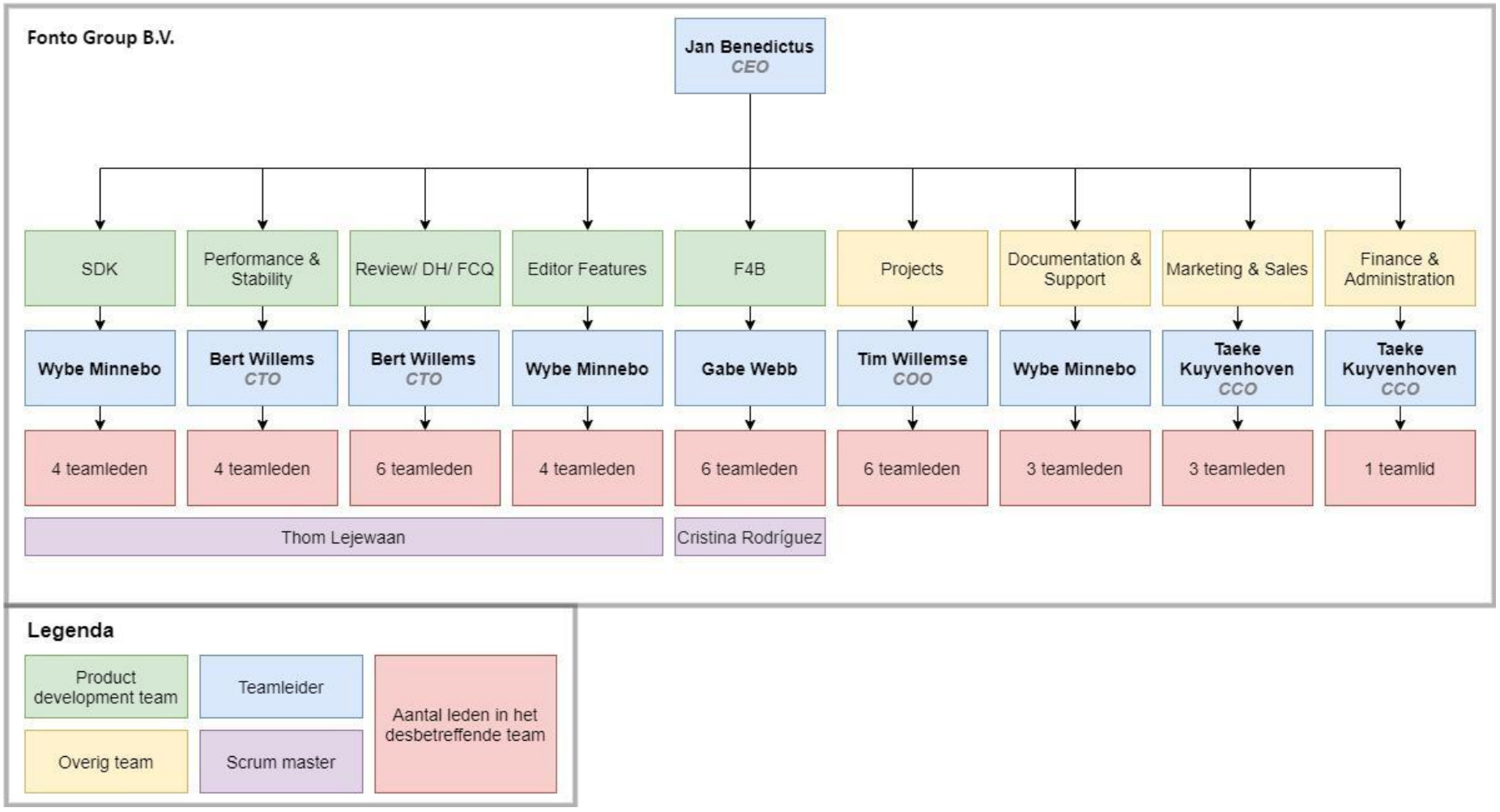
2.3 Bedrijfsstructuur

De teams die betrekking hebben op het project zijn het Performance en Stability team in combinatie met het SDK team. Tijdens het project wordt er contact gelegd met deze teams, omdat deze beschikken over de meeste kennis van het probleemdomein. Het Performance en Stability team richt zich op de kwaliteit van de applicatie. Hierbij voeren ze onder andere prestatietesten uit om de prestatie van verschillende productversies te bewaken. Het SDK team houdt zich vooral bezig met het toevoegen van nieuwe functionaliteiten en het optimaliseren van de code. Daarnaast ondersteunen ze klanten met de implementatie van de Fonto Applicatie.

Naast bovengenoemde teams beschikt Fonto over drie andere product development teams. Het Editor Features team focussed zich op het ontwikkelen en doorvoeren van grote features in de Fonto applicatie. Het Review / DH / FCQ team is verantwoordelijk voor de extra producten die Fonto aanbiedt, zoals de Review tool. Tot slot richt het F4B team zich tot de Fonto 4 Business applicatie.

Elk team beschikt daarnaast over een Product Owner en een Scrum Master. Deze personen voeren onder andere de volgende taken uit; bewaken van het Scrum proces, ondersteuning bieden aan teamgenoten en contact leggen met de stakeholders.

Naast de product development teams beschikt Fonto ook over een aantal ondersteunende teams. Deze teams richten zich op onderwerpen die buiten het domein van de product development teams vallen. Deze teams zijn onder andere verantwoordelijk voor de project planning, budget, documentatie, klantenservice, marketing en financiën.



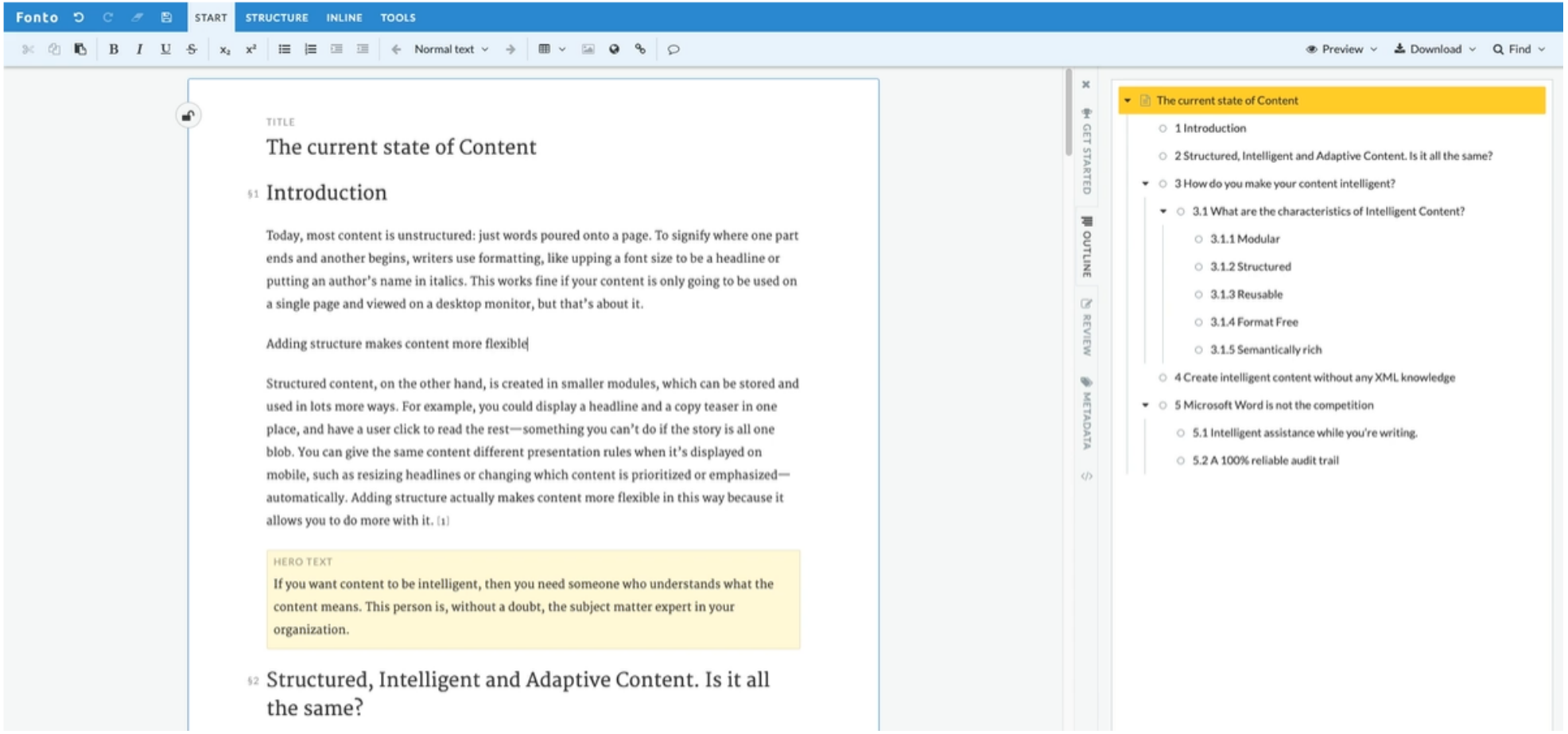
Figuur 1: Organogram bedrijfsstructuur Fonto B.V.

2.4 Probleemdomein

Fonto is een client-side applicatie die een gebruiksvriendelijke manier biedt voor het structureren van content. Gebruikers kunnen met behulp van de editor, genaamd Fonto, onder andere metadata toevoegen, specifieke structuren opslaan om later te hergebruiken, taal en grammatica met behulp van Fonto Content Quality controleren en verandering bijhouden in het document.

Deze applicatie wordt geïntegreerd met een server-side CMS, zodat documenten, structuren etc. van de gebruiker opgeslagen worden. Binnen een bedrijf kunnen verschillende afdelingen gebruik maken van een eigen repository. Hierin kunnen ze vervolgens documenten opzoeken, de verandering inzien en structuren hergebruiken die betrekking hebben op hun eigen afdeling.

Om de kwaliteit van de Fonto applicatie te waarborgen, moet deze getest worden. Om de applicatie te testen maakt Fonto gebruik van een onder andere unit testen, acceptatietesten en integratie testen. Daarnaast voeren ze ook performance testen uit. Deze testen worden uitgevoerd door een programma genaamd test runner. Dit programma wordt meerdere keren per dag uitgevoerd tegen de huidige versie van de Fonto applicatie. Dit resultaat wordt vervolgens vergeleken met de resultaten van oudere versies, om zo regressie in de performance waar te kunnen nemen.



Figuur 2: Fonto editor in gebruik [21]

3.0 Context

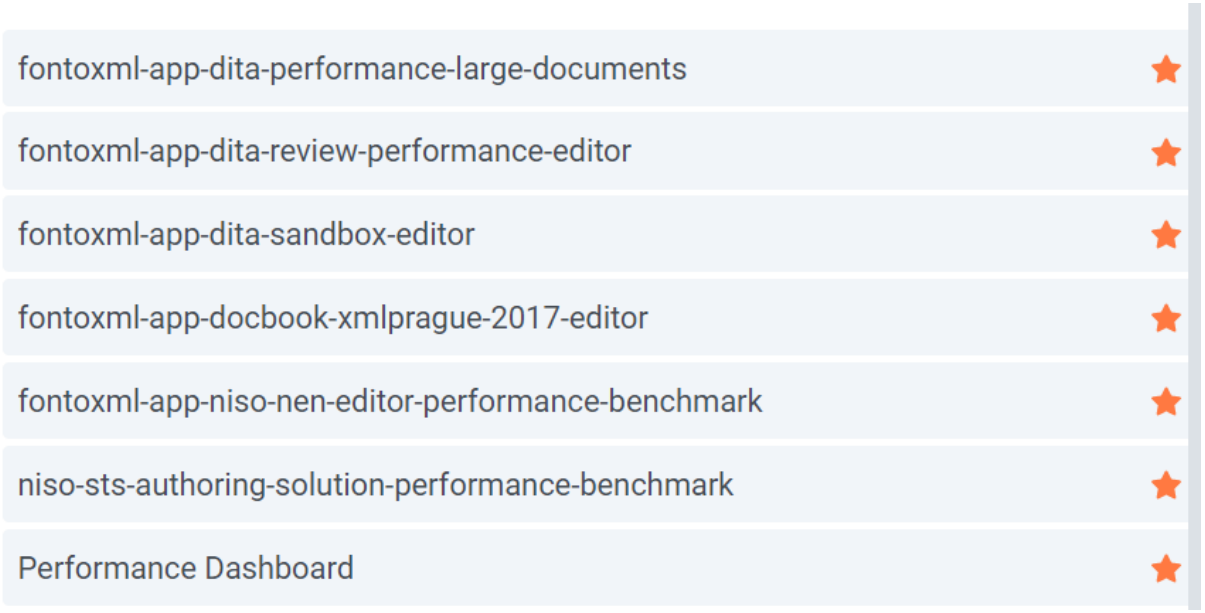
Tijdens de afstudeerperiode staat het programma genaamd als de test runner centraal. In het huidige hoofdstuk worden de test runner en bijbehorende begrippen verder toegelicht om zo een completer beeld te creëren van de context en het probleemdomein.

3.1 Fonto's applicatie

Zoals vermeld in de kop probleemstelling biedt de applicatie van Fonto zijn klanten een gebruiksvriendelijke manier voor het structureren van content. Met behulp van deze applicatie zijn gebruikers in staat gestructureerde content te creëren, zoals weergeven is in figuur 2. De Fonto applicatie wordt aangeboden in de vorm van een software development toolkit (SDK). Een SDK wordt gedefinieerd als een collectie van verschillende software tools, die de gebruiker kan inzetten om de applicatie in te richten naar zijn eigen voorkeur.[44] Hierbij biedt de SDK verschillende functionaliteiten aan waarmee de klant de Fonto applicatie naar eigen wens kan inrichten. De klant is in staat om onder andere de lay-out en naamgeving te veranderen en om widgets toe te voegen. Daarnaast biedt Fonto in deze SDK verschillende editors aan. Een editor biedt een specifieke functionaliteit aan, zoals het bewerken van tabellen of de mogelijkheid om binnen de Fonto applicatie elkaars werk te controleren. De klant maakt zelf de keuze welke van deze editors wel en niet gebruikt worden. Daarnaast is de Fonto applicatie toegankelijk via de browser. Momenteel kan deze ten uitvoer worden gebracht in drie browsers, namelijk Chrome, Edge Legacy en Firefox. Hierbij wordt bij elke browser de laatst uitgebrachte versie ondersteund, evenals voorgaande versies.

Een belangrijk onderwerp binnen Fonto is de performance. Hierbij wenst Fonto dat de performance van de verschillende editors stabiel blijft over verschillende releases. Hiernaast wenst Fonto ook dat code die een negatieve impact heeft op de performance aangewezen kan worden en daarmee eventueel veranderd kan worden. Om dit te realiseren is er een programma ontwikkeld die de performance monitort, genaamd de test runner. Deze wordt in de kop genaamd test runner verder toegelicht.

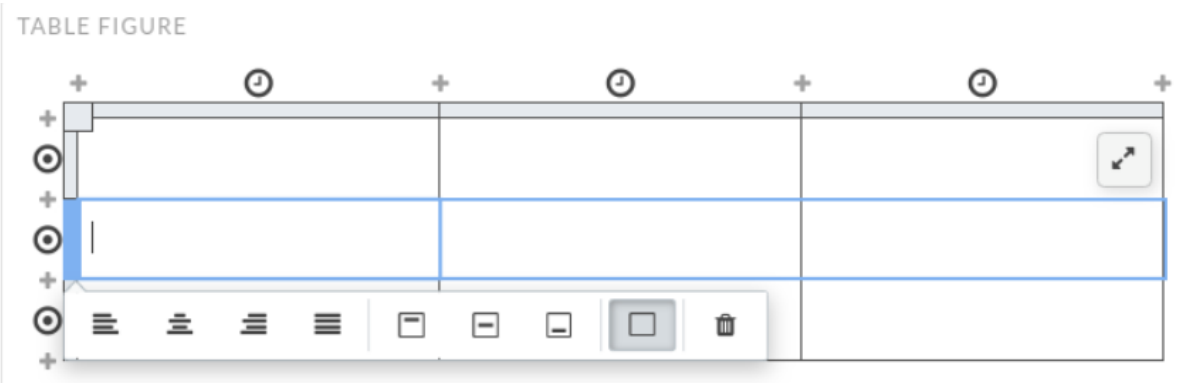
In figuur 3 zijn de verschillende editors weergeven. Elke van deze editors beschikken over verschillende functionaliteiten die de gebruiker kan utiliseren. In figuur 4 is weergeven hoe de gebruiker, met behulp van de Fonto SDK, een functionaliteit toevoegt. Hierbij wordt de functionaliteit aan de applicatie toegevoegd die het mogelijk maakt om tabellen toe te voegen, zoals te zien in figuur 5. Deze specifieke functionaliteit komt voor uit de fontoxml-app-dita-sandbox-editor.



Figuur 3: De verschillende editors binnen Fonto

```
1  configureAsCalsTableElements(sxModule, {
2      table: {
3          localName: 'table'
4      }
5  });
```

Figuur 4: Door middel van de Fonto SDK de tabellen functionaliteit toevoegen



Figuur 5: De functionaliteit toegevoegd in figuur 4

3.2 De test runner

De test runner is een programma dat ontworpen en gerealiseerd is door Fonto. Dit programma is ontwikkeld in de programmeertaal Javascript en heeft als doel om de performance van de beschikbare editors binnen Fonto te meten. Dit doet de test runner door verschillende performance testen uit te voeren op Chrome, Firefox en Edge legacy, waarbij de uitvoeringstijd van de performance test gemeten wordt. Door de performance van de editors te meten kan performance regressie in de editors worden aangetoond. Zodra er zich regressie voordoet, kan dit bijvoorbeeld terug worden herleid naar code die recent is toegevoegd aan de repository. Deze regressie wordt handmatig waargenomen door een van de aangewezen developers binnen Fonto, namelijk Jos Verburg. Hierbij analyseert hij aan het eind van elke dag de grafieken in het Grafana dashboard, zie kop Grafana. Wanneer er regressies worden waargenomen vanaf de tien tot twintig procent wordt dit genoteerd. Zodra deze regressie aanhoudt voor drie of meer uitvoeringen, wordt er actie ondernomen waarbij recent toegevoegde code wordt onderzocht. Daarnaast geeft Jos Verburg aan dat het in de praktijk regelmatig voorkomt dat de uitvoeringstijd van een test verdubbeld. Hierbij geeft hij aan dat de test runner is opgericht om significante regressies waar te nemen en dat er bij regressies rond de tien procent procent zelden actie wordt ondernomen.

De performance heeft in de huidige context betrekking op onder andere hoe snel een editor opstart of hoe snel de editor bepaalde functies kan uitvoeren. Hierbij wordt er niet gemeten hoelang het duurt om data vanuit de database op te halen, omdat dit volgens Fonto buiten de scope valt van de test runner.

Zoals eerder vermeld test de test runner de performance van de verschillende editors uit de SDK. Elke editor wordt hierbij apart getest met een selectie van performance testen, waarbij de uitvoeringstijd van de testen gemeten wordt. Tijdens de uitvoering van een performance test kunnen onder andere de volgende acties ondernomen worden; het toevoegen en verwijderen van een tabel, scrollen door de applicatie en gebruik maken van de zoekfunctie. Elk van deze activiteiten is gedefinieerd in een aparte test. In figuur 6 is het proces van de test runner weergeven in een sequentiediagram. De werking wordt hieronder verder toegelicht.

Reporting service

Binnen de test runner wordt allereerst de reporting service opgestart. Deze service is in staat om de uitvoeringstijd van de verschillende performance testen in een browser te meten.

Testconfig.json

Vervolgens wordt de testopzet bepaald door het testConfig.json bestand. Dit bestand bevat voor elke editor de lijst met performance testen die uitgevoerd moeten worden. In figuur 7

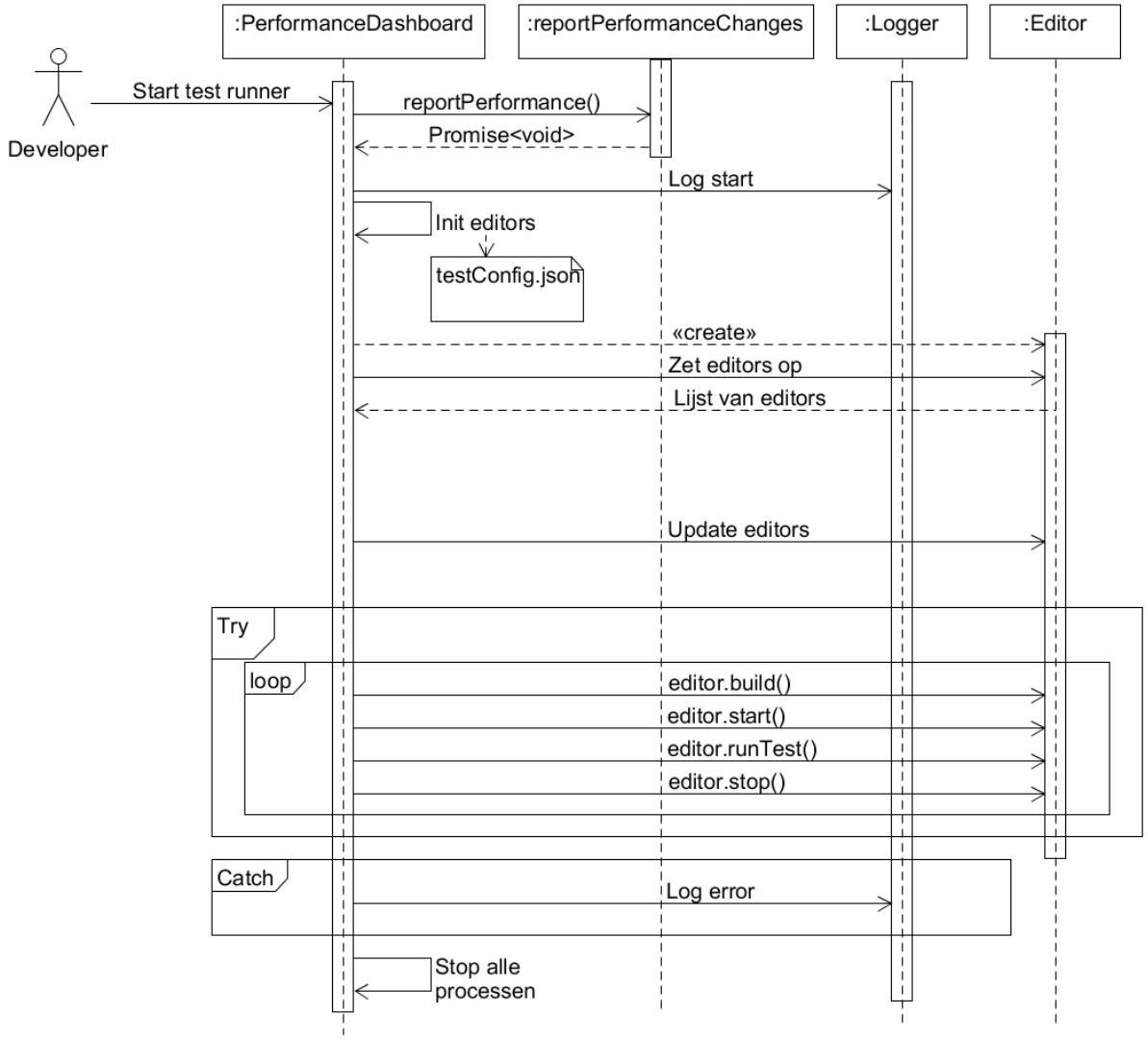
is een voorbeeld weergegeven waarbij de review editor getest wordt. Hierbij staat aangegeven dat de test “ReviewLoadTime” uitgevoerd moet worden op Chrome, Firefox en Edge Legacy. Daarnaast moet de editor versie aanwezig in de productieomgeving van Fonto getest worden.

Upgraden van de editor

De code van de editors staat lokaal opgeslagen in de test runner. Wanneer de code van de editors aangepast wordt in de productieomgeving moeten de editors die opgeslagen staan in de test runner geüpdatet worden. Hierdoor worden de nieuwste versies van de editors getest, waardoor de impact van de recente veranderingen in de code op de performance direct gemeten kan worden.

Testproces

Tijdens het testproces worden de editors stuksgewijs getest. Allereerst wordt een editor gebouwd. Hierbij wordt de editor opgestart in de beschikbare browsers, namelijk Chrome, Firefox en Edge Legacy. Elke test gedefinieerd in het testConfig.json bestand wordt vervolgens uitgevoerd waarna de gemeten uitvoertijd gelogd wordt naar de test runner. De resultaten worden tot slot opgeslagen in een PostgreSQL database, die gehost wordt op Microsoft Azure.



Figuur 6: Sequentiediagram test runner

```
{
    "branch": "master",
    "browsers": ["chrome", "edge", "firefox"],
    "editor": "fontoxml-app-dita-review-performance-editor",
    "environment": "production",
    "path": "fontoxml-app-dita-review-performance-editor-production",
    "tests": [
        {
            "name": "reviewLoadTime",
            "documentIds": ["map-main.xml"]
        }
    ]
}
```

Figuur 7: Voorbeeld uit het testConfig.json bestand

Database

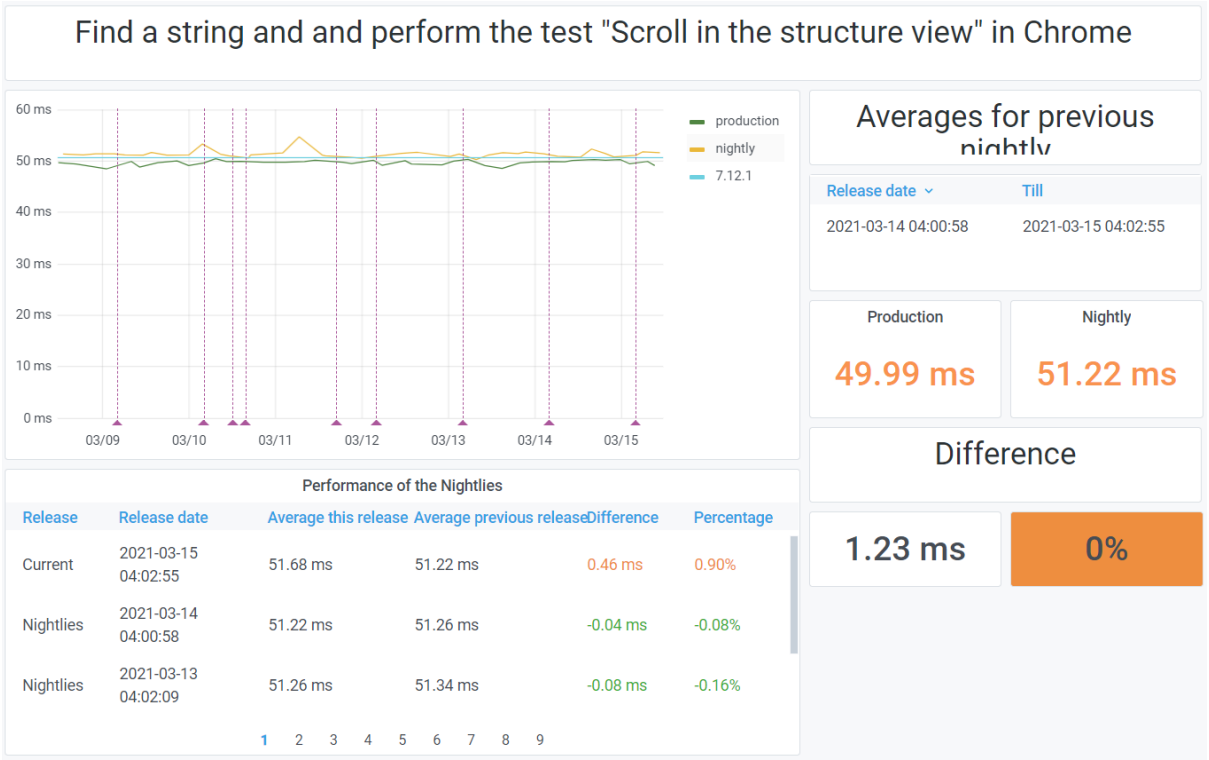
Na afloop van de test runner worden de testresultaten opgeslagen in een PostgreSQL database, die wordt aangeboden door Microsoft Azure. In deze database zijn per uitgevoerde performance test onder andere de gemiddelde uitvoertijd, gebruikte browser en gebruikte editor te vinden.

Omgeving

De test runner wordt ten uitvoer gebracht op een server die fysiek aanwezig is in de vestiging van Fonto. Deze server staat constant op actief waarbij de test runner om de vier uur ten uitvoer gebracht. Meer informatie hierover is te vinden in hoofdstuk 4.

Grafana

De testinformatie afkomstig van de test runner wordt visueel weergegeven in een dashboard, genaamd Grafana. Dit dashboard plot de testinformatie, afkomstig uit de database, met als doel om de data eenvoudig interpreteerbaar te maken. Tijdens het onderzoek zal dit dashboard meerdere malen geraadpleegd worden. Op figuur 8 is een grafiek afkomstig van Grafana uitgebeeld. Op deze grafiek worden de testresultaten afkomstig van de test runner, specifiek de uitvoeringstijden, van drie omgevingen uitgezet tegenover de tijd. Hierbij worden de testresultaten van de productieomgeving, onstabiele omgeving (nightly) en de laatst uitgegeven versie opgenomen in de grafiek. In de grafiek hieronder zijn de resultaten van de test genaamd *"Find a string and perform the test "Scroll in the structure view" in Chrome"* weergegeven. De y-axis geeft de uitvoeringstijd weer in milliseconden, de x-axis geeft de datum weer.



Figuur 8: Voorbeeld grafiek afkomstig van Grafana

4.0 Opdrachtomschrijving

In hoofdstuk 4 wordt er ingegaan op de opdrachtomschrijving. Hierbij worden de aanleiding, probleemstelling, hoofdvraag en deelvragen behandeld.

4.1 Aanleiding

Het programma dat de performance testen uitvoert, genaamd de test runner, heeft zijn grenzen bereikt en kan niet meer worden uitgebreid. Momenteel kosten de performance testen vier uur om te voltooien. Fonto heeft vastgesteld dat er minimaal drie testen opeenvolgend uitgevoerd moeten, om regressie aan te tonen. Dit leidt ertoe dat Fonto binnen een halve dag kan aantonen of er regressies zijn opgetreden binnen de applicatie. Zodra de selectie van performance testen langer duurt, kan deze doelstelling niet meer behaald worden. Hierdoor is het dus niet mogelijk om nieuwe testen toe te voegen, omdat er anders teveel tijd zit tussen de regressies en de constatering daarvan. Daarnaast is het hierdoor niet mogelijk om in de test runner stresstesten uit te voeren, die de bovengrens bewaken van de Fonto applicatie.

Verder voldoet de huidige test runner niet aan de ISO 27001, wat tevens een doelstelling is binnen Fonto. De test runner beschikt niet over disaster recovery waardoor er bij hardware fouten, inbraken, natuurrampen etc. een hoog risico ontstaat, doordat er geen back-up beschikbaarbaar is. Daarbij kunnen deze incidenten zorgen voor een lange hersteltijd, wat resulteert in een periode waarin de applicatie van Fonto niet getest kan worden op het gebied van performance. Hiernaast is Fonto zelf verantwoordelijk voor het serverbeheer. Dit brengt hoge risico's met zich mee op het gebied van security.

4.2 Probleemstelling

De test runner, waar uitsluitend performance testen worden uitgevoerd, heeft zijn limiet bereikt en is daarnaast ook niet schaalbaar. De huidige test runner draait op één server en is niet in staat op te schalen. Hierdoor kan Fonto geen nieuwe performance testen toe voegen, kunnen ze geen stress testen uitvoeren en kunnen ze geen effectieve performance testen starten. Dit leidt ertoe dat Fonto de kwaliteit van hun applicatie niet kan waarborgen. Fonto heeft de cloud geselecteerd als mogelijke oplossing, name door de mogelijkheden die de cloud verschaft op het gebied van schaalbaarheid. Daarnaast is Fonto van plan de nieuwe test runner te ontwikkelen met behulp van een orchestrator architectuur.

4.3 Hoofdvraag

Hoe kan de test runner bij Fonto schaalbaar gemaakt worden?

4.4 Deelvragen

1. Bieden performance testresultaten, afkomstige van een cloud instance, dezelfde betrouwbaarheid als performance testresultaten afkomstig van de on-premise omgeving?
2. Hoe ontwerp je de orchestrator zo dat hij constant kan draaien in een cloud / CI omgeving, waarbij hij de verschillende workers continue blijft aansturen?
3. Is het mogelijk om de log gegevens te verplaatsen naar een centrale plek, die toegankelijk is via een web interface?
4. Hoe plot je de testdata in een grafiek en welke tools kan je daarvoor gebruiken?

5.0 Doelstelling en eindproducten

Om een helder beeld te krijgen van het doel, wordt deze behandeld in hoofdstuk 5. Hiernaast wordt dit verder toegelicht door middel van het gewenste resultaat.

5.1 Doelstelling

Na afloop van de opdracht is er onderzoek gedaan naar de mogelijkheden van een nieuwe test runner bij Fonto. Op basis van het uitgevoerde onderzoek is er een ontwerp opgesteld die de nieuwe test runner in kaart brengt. Aan de hand van dit ontwerp is er ook een prototype opgesteld.

5.2 Gewenste resultaat

Aan het eind de afstudeerperiode wordt er een prototype opgeleverd van de nieuwe test runner samen met het onderzoek, ontwerp en adviesrapport. Dit prototype vormt de basis voor de test runner en kan in de toekomst verder doorontwikkeld worden door Fonto. Het uitbreiden van het prototype kan op basis van het uitgevoerde onderzoek in combinatie met het adviesrapport. Met behulp van de opgeleverde producten is Fonto beter in staat om een hogere kwaliteit software te waarborgen, doordat ze onder andere meer performance testen kunnen uitvoeren, eenvoudig hun testomgeving kunnen opschalen en doordat ze beter inzicht hebben gekregen in de teststatistieken.

In het geval dat er niet voldoende tijd is om alle fases te doorlopen is de keuze gemaakt, in samenwerking met Fonto, om kwaliteit te kiezen boven kwantiteit. In andere woorden zullen de eerdere fases, in dit geval de analyse-en ontwerpfase, in meer detail worden uitgewerkt, waarbij er minder aandacht wordt besteed aan de realisatie- en testfase.

6.0 Werkwijze

Tijdens het project wordt er een hybride vorm van Scrum gehanteerd. Hierbij wordt er gebruik gemaakt van de Scrum rituelen en artefacten. Deze zijn in de tabel hieronder weergegeven. Het onderzoek, creëren van het prototype en het opzetten van het adviesrapport volgen een waterval methodiek, wat in combinatie met Scrum leidt tot een hybride werkwijze. Allereerst moet het onderzoek uitgevoerd worden waaruit een conclusie wordt getrokken. Op basis van deze conclusie wordt het prototype gerealiseerd. De schaal van het prototype maakt het overbodig om in iteraties te werken. Het hele ontwerp kan in één keer opgesteld worden, waarna deze gerealiseerd en getest wordt.

Ritueel	Artefact	Frequency
Stand-up		Dagelijks
Sprint planning		Begin van de week
Sprint review/ retrospective		Eind van de week
Backlog refinement		Begin van de week
	Sprint backlog	
	Product backlog	

Tabel 1: Selectie gebruikte Scrum rituelen en artefacten

7.0 Introductie cloud

Om de opgestelde deelvragen te beantwoorden is er onderzoek uitgevoerd naar de cloud. Het uitgevoerde deskresearch is hieronder weergegeven.

7.1 Wat is de cloud?

Om het begrip cloud uit te leggen worden er twee definities gebruikt die afkomstig zijn van cloud leveranciers. Zo licht Microsoft, eigenaar van een van de grootste cloud platformen genaamd Microsoft Azure, het begrip cloud als volgt toe 'The cloud is not a physical entity, but instead is a vast network of remote servers around the globe which are hooked together and meant to operate as a single ecosystem. These servers are designed to either store and manage data, run applications, or deliver content or a service such as streaming videos, web mail, office productivity software, or social media. Instead of accessing files and data from a local or personal computer, you are accessing them online from any Internet-capable device—the information will be available anywhere you go and anytime you need it.'[1]

Bovenstaande definitie omschrijft de cloud als een netwerk van servers die in combinatie één ecosysteem vormen. De servers worden door een aanbieder onderhouden in serverruimtes verdeeld over de wereld. Deze servers worden gehuurd door particulieren en bedrijven om onder andere applicaties te hosten of data op te slaan. Wanneer een entiteit gebruik maakt van de cloud hoeft deze niet meer zelf zijn database, opslag etc. op te zetten, omdat dit geregeld wordt door de cloud provider.

De cloud biedt voordelen voor kleine en grote gebruikers. Amazon Web Services (AWS) legt het begrip cloud als volgt uit ‘cloud computing is the on-demand delivery of IT resources over the Internet with pay-as-you-go pricing. Instead of buying, owning, and maintaining physical data centers and servers, you can access technology services, such as computing power, storage, and databases, on an as-needed basis from a cloud provider like Amazon Web Services (AWS).’[52]

In de definitie van AWS komt een belangrijk begrip aan bod die de cloud aantrekkelijk maakt, namelijk het pay-as-you-go model. Gebruikers van de cloud betalen enkel voor de opslag, performance etc. die gebruikt wordt. Wanneer er meer vereist is, kan dit in eenvoudig worden opgeschaald. Dit biedt nieuwe mogelijkheden in tegenstelling tot de traditionele on-premise aanpak, waarbij de servers met regelmaat maar voor een beperkt gedeelte gebruikt worden en opschalen een tijds- en kapitaalintensief proces is. Daarnaast verschaft de cloud de mogelijkheid om op te schalen zodra er zich piekbelastingen voordoen. Hierbij betaalt de gebruiker uitsluitend voor de periode dat de extra computerkracht vereist is, in tegenstelling tot de traditionele on-premise aanpak, waarbij extra hardware aangeschaft moet worden om piekbelasting te doorstaan.

7.2 Verschillende soorten cloud

Het begrip cloud kan geïmplementeerd worden op verschillende manieren met elk andere functionaliteiten, voor- en nadelen. Hieronder worden deze toegelicht.

Publieke cloud

De publieke cloud is de meest gebruikte vorm van de cloud. Bij deze vorm is de cloud provider beheerder van de cloud resources, waarbij deze via het internet worden verhuurt aan de gebruikers. Daarnaast worden deze middelen gedeeld door meerdere gebruikers. Een server kan hierbij benut worden door tientallen gebruikers die elk een gedeelte van de onderliggende computerkracht van de server gebruiken. Daarnaast biedt de publieke cloud een aantal voordelen. Bij de publieke cloud worden de cloudresrouces beheert door de cloud provider, hierdoor hebben gebruikers geen onderhoud aan de servers en hoeft er geen kapitaal aangeschaft te worden. Verder biedt de publieke cloud een hoge beschikbaarheid doordat de provider beschikt over een significante hoeveelheid servers. Hierdoor zijn gebruikers eenvoudig in staat hun applicatie op te schalen.[2]

Private cloud

Wanneer de resources eigendom zijn van een gebruiker of bedrijf wordt dit gedefinieerd als private cloud. De beheerders zijn eigenhandig verantwoordelijk voor het onderhoud, de performance, de hardware en schaalbaarheid van de servers. In tegenstelling tot de publieke cloud, waar dit afgehandeld wordt door de cloud provider. De servers en infrastructuur zijn regelmatig opgezet in datacenters van het desbetreffende bedrijf, maar kunnen ook aangeboden worden door een externe serviceprovider. Private cloud resulteert in hogere kapitaalkosten vergeleken met de publieke cloud, maar biedt wel een volledige privaat netwerk met volledige aanpasbaarheid.[2]

Hybride cloud

De hybride cloud is een combinatie van de publieke en private cloud waarbij onderdelen van beide vormen samenwerking als één cloud omgeving. Door middel van de hybride cloud kunnen gebruikers profiteren van de voordelen die de publieke en private cloud bieden. Een voorbeeld hierbij is het opschalen van de private cloud. Wanneer deze piekbelasting ervaart kan dat afgehandeld worden in de publieke cloud, terwijl de gevoelige data staat opgeslagen in de private cloud.[3]

7.3 Waarom cloud?

De hoofdvraag van het onderzoek beoogt antwoord te geven op de vraag hoe de test runner, bij Fonto, op een schaalbare manier ingericht kan worden. De cloud brengt veel mogelijkheden op het gebied van schaalbaarheid en is daarom geselecteerd als onderzoeksonderwerp. Hierbij wordt er onderzoek gedaan naar de verschillende mogelijkheden die de cloud biedt en of de cloud een rendabele oplossing is.

Allereerst wordt onderzocht of de publieke cloud een gunstige oplossing is. Vergeleken met de private cloud biedt deze meer voordelen voor Fonto. Zoals vermeld in hoofdstuk 4.1 wenst Fonto te voldoen aan de ISO 27001. Bij de overstap naar de publieke cloud worden de machines onderhouden door de cloud provider, waardoor deze voldoen aan de ISO 27001. In het geval van de private cloud is Fonto zelf verantwoordelijk voor het onderhoud aan de machines. Om hierbij te voldoen aan de ISO 27001 zal Fonto een systeembeheerder in moeten huren. Dit leidt tot extra kosten en is niet de ideale oplossing voor Fonto. Daarnaast biedt de publieke cloud, in vergelijking met de private cloud, een eenvoudige oplossing voor schaalbaarheid. Wanneer een private cloud oplossing geselecteerd wordt, zal Fonto zelf de nieuwe machines aan moeten schaffen waarbij Fonto zelf verantwoordelijk is voor het systeembeheer.

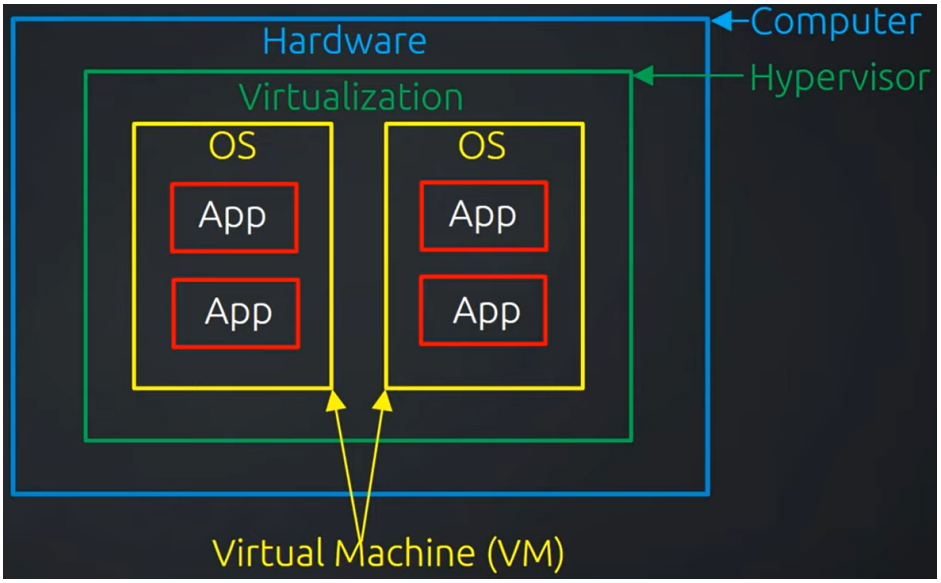
8.0 Literatuuronderzoek

In hoofdstuk 8 wordt er door middel van literatuuronderzoek meer informatie vergaard omtrent performance testing in de cloud. Hierbij worden onderzoeken geanalyseerd en beschreven die in relatie staan tot performance in de publieke cloud.

8.1 Het probleem met de publieke cloud

Het voornaamste probleem van de test runner bedraagt de incompetentie om de applicatie op te schalen, zie hoofdstuk 4.2. Bij problemen op het gebied van schaalbaarheid is de publieke cloud een oplossing die voor de hand ligt. De applicatie wordt gehost in de cloud, zodra er behoefte is aan meer computerkracht kan dit eenvoudig worden verhoogd. Hoewel de cloud in eerste instantie een rendabele oplossing lijkt voor performance testing in de cloud, doen er zich ook een scala van problemen voor. Hieronder worden deze problemen besproken.

Het voornaamste probleem van de publieke cloud, in relatie tot performance testing, is de gelimiteerde controle over de omgeving, wat leidt tot onbetrouwbare en fluctuerende performance.[4] Deze onbetrouwbaarheid ontstaat doordat de onderliggende hardware van de cloud op hetzelfde moment wordt geutiliseerd door meerdere gebruikers. Cloud providers splitsen hardware units op in componenten die bovenop de machine ten uitvoer worden gebracht, een proces dat bekend staat als virtualisatie.[5] Gebruikers van de cloud utiliseren vervolgens de componenten om onder andere applicaties of services op te hosten. Op deze manier kan een cloud provider optimaal gebruik maken van de beschikbare hardware. Dit brengt echter het probleem met zich mee dat zwaarwegende handelingen die uitgevoerd worden door andere gebruikers, invloed kunnen hebben op de performance van de Fonto cloud omgeving.[42] De onderliggende hardware heeft beperkte kracht, wanneer dit gebruikt wordt door een partij kan de performance van de andere partij tijdelijk onvoorspelbaar zijn. Dit fenomeen staat bekend als noisy neighbours.[10] Daarnaast brengt virtualisatie meer laadtijd met zich mee.[6] Dit komt doordat de virtualisatiesoftware de verschillende requests moet afhandelen en doorsturen naar het correcte component. Een artikel (ShatteredSilicon, 2020) concludeerde onder andere dat componenten 33% minder performance hebben, in vergelijking met identieke hardware die geen gebruik maakt van virtualisatie. Daarnaast concludeerde een onderzoek dat virtualisatie instabiliteit en vertraging m.b.t. TCP/UDP veroorzaakt.[43]



Figuur 9: Visuele weergave van virtualisatie [5]

De performance in de cloud kan onbetrouwbaar zijn, mede doordat meerdere gebruikers de onderliggende hardware utiliseren. Wanneer performance testen uitgevoerd worden is het uiterst belangrijk dat er een stabiele performance present is op de onderliggende machine. Als dit niet het geval is, zijn de testresultaten onbetrouwbaar. Er kan immers niet vastgesteld worden of de prestatie regressie een resultaat is van de fluctuerende performance op de onderliggende machine of door aanpassingen in de applicatieprogrammatuur. Hoe significant de onbetrouwbaarheid in de cloud is wordt in de kop hieronder toegelicht.

8.2 Deskresearch significantie onbetrouwbaarheid

Om vast te leggen welke informatie omtrent performance testing in de cloud momenteel bestaat, is er gerelateerde literatuur onderzocht en zijn de conclusies daarvan opgenomen. Hierbij zijn twee onderzoeken gerelateerd aan performance testing in de cloud beschreven. De beschrijvingen van de onderzoeken zijn vastgelegd in bijlage I: Onderzoek A en bijlage J: Onderzoek B. De conclusies uit de geselecteerde onderzoeken zijn hieronder besproken.

Onderzoek A

Het onderzoek uit 2018 (Laaber et al., 2018) is beschreven in bijlage I. Hieronder zijn de conclusies van het onderzoek opgenomen.

a. Testen die beschikken over een korte executietijd van tientallen nanoseconden resulteren in onnauwkeurige metingen.

b. Testen die gebruik maken van I/O intensieve operaties zijn instabiel, ten opzichte van testen die geen gebruik maken van I/O intensieve operaties.

c. De prestaties van verschillende families aangeboden door de cloud providers bieden geen verschil in performance ten opzichte van elkaar.
d. De performance van een bare metal machine is even stabiel, ten opzichte van de geselecteerde cloud instances.
e. De gebruikte methodiek voor het meten van de regressies, genaamd de Wilcoxon rank-sum methode, blijkt niet geschikt.

Onderzoek B

Het onderzoek uit 2016 (Leitner & Cito, 2016) is beschreven in bijlage J. Hieronder zijn de conclusies van het onderzoek opgenomen.

a. De geselecteerde Cloud providers bieden elk een andere mate van performance stabiliteit.
b. De dagen hebben geen specifieke impact op de performance stabiliteit van een cloud instance.
c. De regio waarin een cloud instance gealloceerd is heeft impact op de performance stabiliteit van de cloud instance.
d. De prestaties van verschillende families aangeboden door de cloud providers bieden geen verschil in performance ten opzichte van elkaar.

9.0 Opzet testplan

In de huidige kop wordt het testplan beschreven. Het doel van het testplan is om vast te stellen hoe betrouwbaar de testresultaten van de test runner zijn, wanneer deze uitgevoerd wordt in de publieke cloud. De resultaten van deze test worden vergeleken met de resultaten afkomstig van de huidige test runner, die actief is in de on-premise omgeving bij Fonto. Hieronder worden de opzet en kosten van het testplan toegelicht.

9.1 Cloud providers

Voor de testen worden er verschillende cloud providers geutiliseerd die zijn geselecteerd op basis van meerdere criteria. De selectie van cloud providers bestaat uit:

a. Microsoft Azure
b. AWS
c. GCE

Bovengenoemde cloud providers zijn onder andere geselecteerd, omdat deze gebruikt worden in de onderzoeken beschreven in hoofdstuk 8.2. Daarnaast zijn dit de meest populaire cloud providers, die elk een significant deel van de marktaandelen bezitten. Zo waren AWS en Microsoft Azure gecombineerd goed voor 83.3% van de marktaandelen in 2020.[11] Verder zijn andere cloud providers niet geselecteerd wegens tijdrestricties. cloud

providers zoals IBM cloud, Alibaba cloud en Oracle cloud zouden in een nader onderzoek kunnen worden behandeld.

Naast bovengenoemde criteria zijn deze cloud providers geselecteerd mede doordat de geselecteerde cloud providers diverse server families en instances aanbieden (zie bijlage B: cloud provider families). Verder concludeerde het onderzoek (Leitner & Cito, 2016) dat de server regio invloed kan hebben op de betrouwbaarheid van de performance. Alle geselecteerde cloud providers bieden meerdere server regio's aan, waardoor de regio's eventueel tijdens de test veranderd kunnen worden, om zo aan een verschil in performance aan te duiden.

Daarbij geeft Fonto de voorkeur aan Microsoft Azure. Fonto geeft aan het meest bekend te zijn met deze cloud provider, zo zijn onder andere de databases en web services van Fonto gealloceerd op Microsoft Azure. Daarnaast is Fonto geïnteresseerd in de resultaten van GCE en AWS ten opzichte van Microsoft Azure, zo acht Fonto deze cloud providers leiders in de markt.

9.2 Type server

De geselecteerde cloud providers verschaffen meerdere instances, gecategoriseerd in families (zie bijlage B: Cloud provider families). Voor elke cloud provider worden er één of meerdere instances gehuurd waarop het testprogramma wordt uitgevoerd. Uit onderzoek (zie hoofdstuk 8.2) blijkt dat instances die specifiek ingericht zijn voor performance gerichte handelingen niet betrouwbaardere resultaten opleveren dan standaard instances.

Het verbruik van de test runner is geanalyseerd. Dit is terug te vinden in bijlage E: verbruik van de test runner. Hierbij is bijgehouden hoe intensief de test runner gebruik maakt van het geheugen, CPU, schijf en GPU. Uit deze analyse blijkt dat een instance met beter presterende cores kan leiden tot betrouwbaardere resultaten, mede doordat de test runner grotendeels gebruik maakt van één core. Hierdoor zal er bij elke cloud provider een instance ingezet worden, die geoptimaliseerd is voor taken intensief op de CPU. Hoewel uit de verschillende onderzoeken blijkt dat dit geen significant verschil maakt op de betrouwbaarheid van de performance, zal dit toch getest worden om deze claim te valideren. Hieronder zal worden toegelicht welke instances worden gekozen per cloud provider.

Microsoft Azure

Voor Microsoft Azure zijn twee machines geselecteerd.[12] Uit de standaard familie is de A2m v2 instance gekozen, zoals weergegeven is in tabel 2. Deze biedt twee virtuele cores met 16GB RAM. Alhoewel deze instance meer RAM bevat dan eerder aangegeven als benodigd, is dit de eerste versie van de standaard familie met minimaal twee cores. Daarnaast biedt Microsoft Azure ook een instance met 8GB RAM genaamd A2 v2, maar deze bevat vier

virtuele cores en is daarnaast duurder. Daardoor is de gekozen instance het meest aantrekkelijk.

Naast de standaard instance, worden de testen uitgevoerd op een instance geoptimaliseerd voor intensief CPU gebruik. Hiervoor is de F2 instance geselecteerd. Deze instance beschikt over twee cores die elk krachtiger zijn dan de cores gealloceerd in de A2m v2 instance. De F2 instance is geselecteerd uit de F-familie doordat deze beschikt over voldoende RAM en over twee cores. [12]

AWS

Uit de standaard familie van AWS is de instance m5.large geselecteerd.[13] Dit is de eerste in de standaard instance familie met minimaal twee cores.

Daarnaast is, er uit de familie geoptimaliseerd voor CPU intensieve taken, de c5.large instance geselecteerd. Deze bevat twee cores, die vergelijkbaar met de Microsoft Azure instances, beschikken over meer computerkracht. Net zoals bij de Microsoft Azure families zijn er meer varianten beschikbaar die hetzelfde doel voor ogen hebben, waarbij de verschillen in onderliggende hardware nihil zijn. Daarnaast biedt AWS enkele families alleen aan in een specifieke OS. Zo is de c5 te verkrijgbaar met Windows 10 en is de c6g instance alleen beschikbaar met Linux.

GCE

De e2-standard-2 instance is geselecteerd als standaard instances op GCE. De e2 familie bestaat uit standaard instances, waarbij de geselecteerde instance de eerste in de reeks is met twee cores.[14]

De c2-standard-4 instance is geselecteerd voor CPU intensief gebruik. Dit is de eerste instance uit de reeks geoptimaliseerd voor CPU gebruik. GCE biedt alleen instances aan in deze reeks met minimaal vier cores, wat voor de cloud test overbodig is. De c2 instance zal toch geutiliseerd worden om zo te analyseren hoe deze in verhouding staat tot de andere cloud providers.

Cloud provider	Instance	Type	vCPU	RAM	Prijs per uur
Azure	A2M v2	Standaard	2	16	€0,081
Azure	F2	CPU gebruik	2	4	€0,081
AWS	m5.large	Standaard	2	8	€0,154
AWS	c5.large	CPU gebruik	2	4	€0,145
GCE	e2-standard-2	Standaard	2	8	€0,055
GCE	c2-standard-4	CPU gebruik	4	16	€0,172

Tabel 2: Overzicht instance types

9.3 Operating system

Bij het huren van een server moet een operating system (OS) geselecteerd worden. Hoewel binnen Fonto de meeste servers in de cloud Linux gebruiken, zullen de servers die gebruikt worden tijdens de test uitgevoerd worden op Windows 10. Deze keuze is mede gemaakt doordat de test runner in de huidige situatie ten uitvoer wordt gebracht op een standalone server met Windows 10. Hierdoor zijn de resultaten van de test vergelijkbaar met die van de huidige situatie. Daarnaast is de documentatie over het opzetten van de test runner ingericht voor Windows 10, waardoor het initialisatieproces van de verschillende instances sneller verloopt. Tevens bieden de gekozen cloud providers Windows 10 en Linux aan voor diverse tarieven. Wanneer een reeks van instances op AWS vergeleken wordt, blijkt dat Windows gemiddeld 50% duurder is dan de Linux variant, zoals weergeven in bijlage D: Kosten Linux en Windows op AWS. Deze marges zijn vergelijkbaar voor andere families binnen AWS. Deze kosten komen ook overeen met die van GCE. [17] Daarnaast biedt Microsoft Azure Linux en Windows voor dezelfde prijs aan.[16] Deze verhoogde kosten voor Windows 10 op AWS bieden voor de test geen probleem aangezien deze maar enkele dagen wordt uitgevoerd (zie testselectie). Wanneer de keuze gemaakt wordt om AWS of GCE te gebruiken als platform voor de test runner, kunnen Linux machines overwogen worden. Verder is een nadeel van Windows 10 de automatische updates die de OS biedt. Wanneer deze automatische updates uitgevoerd worden, kan de performance op de machine beïnvloed worden. Om dit te voorkomen worden de automatische updates op alle machines uitgeschakeld.

9.4 Opslag van de testgegevens

De huidige test runner logt na elke uitvoering de testresultaten naar een PostgreSQL database, die gehost wordt op Microsoft Azure. De logica om testresultaten op te slaan in een database bestaat daarmee dus al en zal tijdens de test daarom ook gebruikt worden. Hierbij wordt er een nieuwe database in Microsoft Azure geïnitialiseerd, zodat er geen conflicten ontstaan in de database van de huidige test runner. De verschillende instances voeren de geselecteerde testen uit en slaan de testresultaten vervolgens op in de nieuwe database. Dit resulteert in een toegankelijk en duidelijk overzicht van de testdata.

9.5 Regio

Uit het onderzoek (Leitner & Cito, 2016), beschreven in hoofdstuk 8.2, wordt geconcludeerd dat de regio waarin de instances gealloceerd zijn, invloed heeft op de performance. Hoewel verschillende regio's naar voren komen in het onderzoek, blijkt het dat de onderlinge verschillen niet in performance voorspelbaar zijn en per instance afwijken. Hierdoor is de keuze gemaakt de standaard regio, namelijk de regio US-west, te selecteren

voor elke cloud provider. Onderzoek naar de verschillen in stabiliteit per regio en cloud provider valt buiten de scope van het huidige onderzoek.

9.6 Testselectie

Om de betrouwbaarheid van de verschillende instances te meten wordt een selectie van performance testen op elke instance uitgevoerd. Deze selectie van testen is voor elke instance identiek, zodat de resultaten onderling vergeleken kunnen worden. Momenteel duurt een uitvoering van de test runner, waarbij alle beschikbare performance testen worden uitgevoerd, vier uur. Hiermee worden er per dag zes meetpunten verzameld, de test runner kan tevens maar zes keer op een dag worden uitgevoerd (zie hoofdstuk 4.1). De onderzoeksresultaten beschikken over een hogere betrouwbaarheid, wanneer deze voortkomen uit meerdere meetpunten. Daarnaast is het overbodig om een scala van performance testen uit te voeren. De reden hiervoor wordt in de volgende alinea's toegelicht. Hierom is er gekozen om de selectie van testen te verkleinen, zodat de totale uitvoeringsduur 30 minuten bedraagt in plaats van 4 uur. Dit verhoogt het aantal dagelijkse meetpunten van een test van 6 naar 48. Welke testen geselecteerd zijn en waarom wordt hieronder toegelicht.

Uit de totale set van test runner testen wordt een selectie gemaakt. Voor het selecteren van performance testen gelden twee criteria. Ten eerst moet de uitvoeringstijd van een performance test groter zijn dan tientallen nanoseconden. Dit blijkt uit onderzoek (Laaber et al., 2018) waarin geconcludeerd is dat performance testen met een korte uitvoeringstijd minder betrouwbare resultaten opleveren. Ten tweede worden performance testen geselecteerd waarbij de testresultaten stabiel zijn over een lange periode, zoals die in figuur 10 en worden er testen geselecteerd die veel ruis bevatten, zoals weergeven in figuur 11. Voor meer informatie over de figuren 10 en 11, zie hoofdstuk 3. In eerste instantie was de keuze gemaakt om alleen stabiele testen te gebruiken, omdat hierbij werd aangenomen dat deze een hogere betrouwbaarheid bieden omtrent de testresultaten in vergelijking met testen die veel ruis bevatten. Tijdens het definiëren van de statistiek methodiek, kwam naar voren dat een testselectie bestaande uit testen met veel en weinig ruis een completer beeld bieden, in tegenstelling tot alleen testen met weinig ruis. (zie hoofdstuk 12) In bijlage C: Testselectie, is de mate van ruis in combinatie met de uitvoeringstijd per geselecteerde performance test opgenomen. De gekozen performance testen die uitgevoerd gaan worden op de verschillende instances worden hieronder besproken.

Tijdens het selecteren van de performance testen zijn een aantal keuzes gemaakt. Om deze keuzes te valideren is er in gesprek gegaan met Jos Verburg, de ontwikkelaar van de test runner.

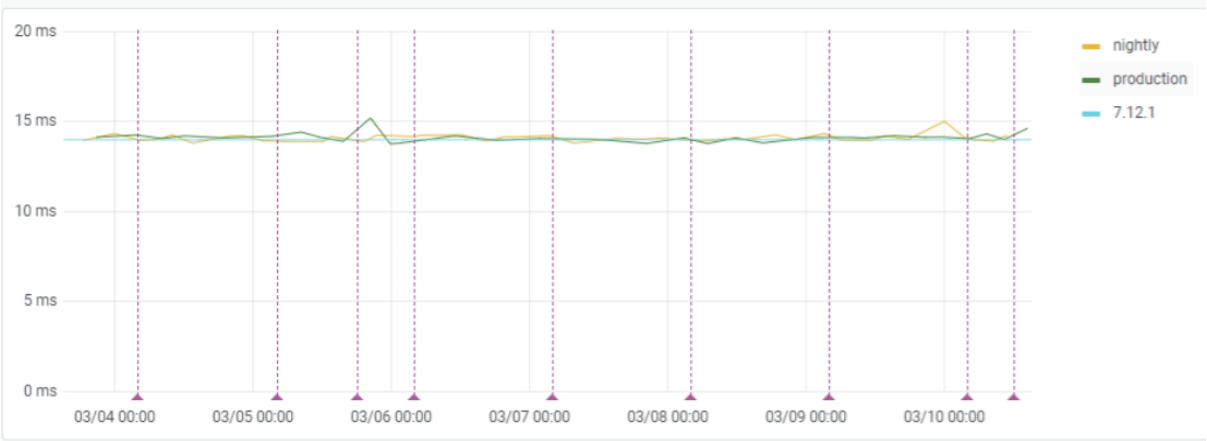
Ten eerste worden alleen de testen van één specifieke editor gebruikt, namelijk de fontoxml-app-dita-sandbox-editor. Zoals vermeld bij hoofdstuk 3, toetst de test runner

verschillende editors. Een aanname hierbij is dat de resultaten van de gekozen editor van toepassing zijn op de overige editors. Jos Verburg valideert dit, waarbij hij aangeeft dat de verschillende editors vergelijkbaar zijn waardoor de resultaten dit ook zijn. De keuze om één editor te gebruiken en niet meerdere komt doordat hierdoor een significante hoeveelheid tijd wordt bespaard. De keuze om de fontoxml-app-dita-sandbox-editor te selecteren in plaats van een andere editor komt doordat deze beschikt over meerdere performance testen, die verschillen in uitvoertijd en hoeveelheid ruis.

Ten tweede worden de performance testen uitgevoerd op Chrome en Firefox, waarbij Edge Legacy achterwege wordt gelaten. De test runner voert performance testen uit op verschillende browsers, zoals vermeld in hoofdstuk 3. De keuze om Edge legacy achterwege te laten komt mede doordat deze beduidend meer tijd kost om op te zetten. Zo moeten onder andere verschillende instellingen in Windows 10 worden aangepast. Tevens verdwijnt Edge Legacy bij de volgende release uit de test runner vanwege het feit dat deze niet meer wordt ondersteund door Microsoft. De nieuwe versie van Microsoft Edge is gebaseerd op Chromium, waardoor deze vergelijkbaar is met Chrome.[19] Hierdoor is het testen van de Edge Legacy browser overbodig.

Ten derde zijn testen geselecteerd variërend in uitvoertijd en hoeveelheid ruis. Zoals eerder vermeld in de huidige kop biedt een testselectie bestaande uit stabiele en onstabiele testen een completer beeld, dan een testselectie bestaande uit enkel en alleen stabiele testen. Daarnaast hebben de performance testen verschillende uitvoeringstijden. Dit resulteert in een completer eindresultaat, doordat allerlei verschillende testen zijn meegenomen in de cloud test. Verder is er geen sprake van testselectie op basis van de impact die een test heeft op de onderliggende hardware. In bijlage E: verbruik test runner is geconcludeerd dat de testen nagenoeg dezelfde impact hebben op de hardware.

Figuur 10: Onstabiele test in Grafana



Figuur 11: Stabiele test in Grafana

9.7 Kosten

Cloud provider	Standaard instance per uur	CPU instance per uur	Totale kosten per dag
Microsoft Azure	€0,081	€0,081	€3,88
AWS	€0,145	€0,154	€7,18
GCE	€0,055	€0,172	€5,45
			€16,47

Tabel 3: Kosten standaard en CPU geoptimaliseerde instances

Database	Totale kosten dag
Microsoft Azure Database	€0,85

Tabel 4: Database kosten

9.8 Betrouwbaarheid

Tijdens de uitvoering van de performance testen kunnen verschillende gebeurtenissen, zoals automatische updates, impact hebben op de resultaten. Het is belangrijk om deze in kaart te brengen, omdat hierdoor een duidelijk beeld ontstaat over welke gebeurtenissen wel en niet te controleren zijn. Hierdoor neemt de betrouwbaarheid van de onderzoeksresultaten toe. Welke gebeurtenissen voort kunnen komen en of deze te controleren zijn wordt hieronder toegelicht. Hierbij zijn enkele maatregelen overgenomen van de huidige test runner omgeving.

Test runner

Tijdens het uitvoeren van de performance testen verandert de code van de test runner niet. Tevens wordt op de verschillende instances exact dezelfde versie gebruikt. Deze komt daarnaast ook overeen met de meest recente versie van de test runner in productie. Hierdoor wordt uitgesloten dat afwijkingen in de resultaten beïnvloed worden door een verschil in test runner programmatuur.

Editor

De performance testen tijdens de cloud test worden uitgevoerd op één editor. Op de verschillende instances zal dezelfde editor versie gebruikt worden die momenteel gebruikt wordt in de huidige test runner omgeving. Hierdoor is een verschil in performance niet te danken aan een afwijkende editor.

Browsers

Chrome en Firefox bieden de mogelijkheid voor automatische updates. Wanneer dit zich voordoet tijdens de uitvoering van een test kan dit de performance op een negatieve manier beïnvloeden. Om dit te voorkomen zullen op alle instances de automatische browser updates uitgeschakeld worden. Daarnaast zullen de Chrome en Firefox versies die gebruikt worden in de huidige test runner omgeving, ook gebruikt op de verschillende instances. Door het gebruik van dezelfde browser versie kan variatie in de performance niet te danken zijn aan verschil in browser software.

OS

Windows 10, zie operating system, wordt gebruikt als OS op de instances. Deze voert automatische updates op zichzelf uit die de performance tijdelijk kunnen beïnvloeden. Deze handeling zullen net zoals in de huidige test runner omgeving uitgeschakeld worden. Verder geeft Jos Verburg aan dat hoewel hiermee de automatische updates uitgeschakeld worden, dit niet uitsluit dat Windows andere processen start die de performance kunnen aantasten. Zo geeft hij aan dat dit fenomeen ook voorkomt in de huidige test runner

omgeving. Hoewel dit de performance negatief kan beïnvloeden, is het mogelijk om na het testproces te achterhalen of Windows 10 processen heeft gestart tijdens het testproces.

Instances

De onderliggende hardware van de instance is niet te controleren. Intensief gebruik van de test runner of van noisy neighbors, zie hoofdstuk 8.1, kan de onderliggende hardware opwarmen wat de performance kan beïnvloeden.[18] Cloud providers voeren daarnaast ook onderhoud en automatische updates uit aan de instances, die de performance tijdelijk kunnen beïnvloeden. Pieken in de resultaten kunnen niet direct verklaard worden als gevolg van de onderliggende hardware. Het is tevens niet bekend wanneer de hardware geupdate wordt en op welke momenten deze een verhoogde temperatuur heeft.

10.0 Opzet testplan

Het testplan wordt uitgevoerd volgens een aantal stappen. Hieronder worden de stappen toegelicht. In bijlage F: Flowchart aanpak zijn de hieronder beschreven stappen afgebeeld in een flowchart.

Stap 1: instances creëren op de cloud providers

De verschillende instances besproken in de kop genaamd type server, worden gecreëerd op de verschillende cloud providers. Hierbij maakt de instance gebruik van Windows 10.

Stap 2: een versie van de test runner lokaal opslaan

Zodra de verschillende instances zijn opgezet, worden de omgevingen ingericht. De meest recente versie van Chrome, Firefox en NodeJs worden geïnstalleerd op elke instance. Daarnaast worden de automatische updates en virusscanner van Windows 10 uitgeschakeld, zodat deze niet interfereren met de uitvoering van de test runner. Verder wordt de meest recente versie van de test runner op de instance geplaatst.

Stap 3: selectie van testen toevoegen aan de test runner

De testselectie besproken in de kop testselectie wordt toegevoegd aan de test runner. Deze selectie zal tijdens de uitvoering elk half uur uitgevoerd worden. Verder is er een script opgezet die elk half uur de test runner start. Dit script wordt gestart met behulp van de Windows taakplanner.

Stap 4: een PostgreSQL database opzetten in Microsoft Azure

Om de testresultaten afkomstig van de verschillende instances op te slaan wordt een PostgreSQL database opgezet in Microsoft Azure.

Stap 5: database configuratie in de test runner aanpassen

In de code van de test runner staat gedefinieerd in welke database de logs moeten worden opgeslagen. Deze code moet zodanig aangepast worden dat de gegevens opgeslagen worden in de bij stap vier opgezette PostgreSQL database. De IP-adressen van de instances worden toegevoegd aan de firewall van de database, zodat deze request van de instances kan ontvangen.

Stap 6: instances testen

Alle instances worden voor één uur gestart om vast te stellen dat deze de juiste testen uitvoeren en dat deze gelogd worden naar de PostgreSQL database. Door de instances vooraf te testen wordt er vastgesteld dat deze correct zijn geïnitialiseerd.

Stap 7: cloud test starten

Alle instances zullen voor een periode van twee dagen constant uitgevoerd worden, waarna de test runner op elke instance in totaal 96 keer tot uitvoer is gebracht. De testperiode van twee dagen is gekozen mede doordat de verschillende instances gecombineerd een significante hoeveelheid geld kosten. Om de kosten voor Fonto realistisch te houden is de keuze gemaakt om de cloud test in eerste instantie twee dagen te laten duren. Ten tweede kan er met behulp van de statistiek methode aangetoond worden of er voldoende data beschikbaar is, zie volgend hoofdstuk. Wanneer blijkt dat er meer testdata benodigd is, is het mogelijk om de test runner een extra paar dagen uit te voeren.

Stap 8: vergelijking resultaten

De resultaten van de test worden vergeleken met die van de on-premise omgeving. Hiermee kan worden vastgesteld hoe onbetrouwbaar de cloud is, in vergelijking met de on-premise omgeving. Hoe deze resultaten vergeleken worden, wordt behandeld in het volgende hoofdstuk.

11.0 Statistiek

De resultaten die vergaard zijn van de verschillende cloud providers worden in hoofdstuk 11 vergeleken, om zo tot een conclusie te komen. Hierbij worden verschillende statistische methodes gebruikt. Hieronder worden de gebruikte statistische methoden, begrippen en aanpak toegelicht.

11.1 Gebruikte variabele

Tijdens het uitvoeringsproces zijn de testresultaten, afkomstig van de zes geselecteerde cloud instances, verzameld in een database. Hierbij bevat elk individueel testresultaat een waarde genaamd de uitvoeringstijd. Deze waarde geeft aan hoelang een specifieke test

duurt om uit te voeren. De uitvoeringstijd is gebruikt in de geselecteerde statistische methodieken als input, om zo tot een conclusie te komen. Hierbij worden de vergaarde uitvoeringstijden van de zes geselecteerde cloud providers dus vergeleken met de uitvoeringstijden van de on-premise omgeving.

11.2 Begrippen

Hieronder worden enkele begrippen toegelicht die van belang zijn tijdens het statistische proces.

Populatie

Investopedia beschrijft een populatie als 'A population is a distinct group of individuals, whether that group comprises a nation or a group of people with a common characteristic. In statistics, a population is the pool of individuals from which a statistical sample is drawn for a study. Thus, any selection of individuals grouped together by a common feature can be said to be a population.'[24]

De populatie is de totale groep, bestaande uit onder andere personen of data, die gebruikt wordt tijdens een onderzoek.[23]

Sample

Investopedia beschrijft een sample als 'A sample refers to a smaller, manageable version of a larger group. It is a subset containing the characteristics of a larger population. Samples are used in statistical testing when population sizes are too large for the test to include all possible members or observations. A sample should represent the population as a whole and not reflect any bias toward a specific attribute.'[25]

Een sample is een gedeelte van de populatie. Samples worden regelmatig gebruikt bij grote datasets doordat hierdoor niet alle data benaderd hoeft te worden.[26]

Variatie

Adam Hayes beschrijft de term variatie als 'The term variance refers to a statistical measurement of the spread between numbers in a data set. More specifically, variance measures how far each number in the set is from the mean and thus from every other number in the set.'[28]

De variatie, in de context van statistiek, geeft de mate van spreiding aan in een dataset. Hiermee kan de volatiliteit van een dataset bepaald worden.[29]

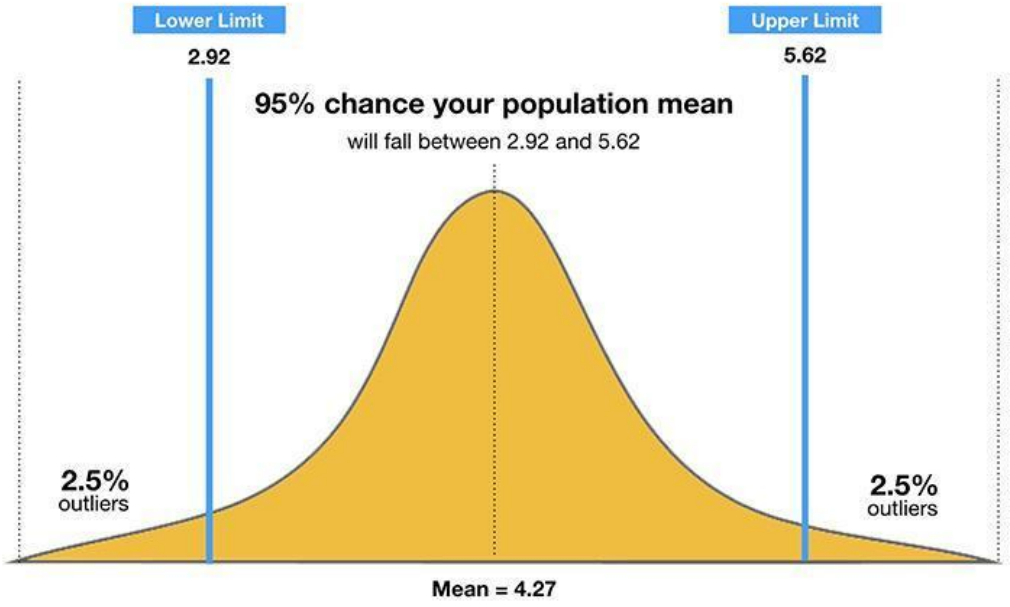
11.3 Gebruikte methodieken

Tijdens het vergelijkingsproces worden verschillende statistische methodieken gebruikt, deze worden hieronder uitgelegd.

Betrouwbaarheidsinterval

Een betrouwbaarheidsinterval geeft een interval aan waarin een bepaalde hoeveelheid van de data valt. Hierbij wordt er een onder-en bovengrens berekend op basis van een gekozen betrouwbaarheidsniveau, zoals 90%, 95% en 99%. Op figuur 12 is een normale verdeling weergegeven, waarbij met blauwe lijnen de onder-en bovengrens zijn aangegeven. De kans is hierbij 95% dat een waarde uit de desbetreffende dataset binnen deze onder-en bovengrens valt. Door middel van een betrouwbaarheidsinterval is het mogelijk om de mate van variabiliteit in een dataset te meten. Hoe kleiner de afstand tussen de onder-en bovengrens, des te lager de variabiliteit is in de desbetreffende dataset.[32]

Met de inzet van betrouwbaarheidsintervallen kan de variabiliteit van een performance test op de on-premise omgeving vergeleken worden met die van een cloud instance. In andere woorden, de volatiliteit van beide omgevingen kan hiermee in kaart worden gebracht. Tijdens het onderzoek wordt een betrouwbaarheidsniveau van 95% gehanteerd. Dit niveau wordt in het algemeen het meest gebruikt. [32] Daarnaast hanteren de besproken onderzoeken uit hoofdstuk 8.2 ook een betrouwbaarheidsniveau van 95%.



Figuur 12: Weergave normale verdeling met een betrouwbaarheidsinterval van 95% [31]

Toets van Levene

Om vast te stellen of twee datasets, in dit geval een van de cloud instances en de on-premise omgeving, beschikken over een identieke variatie is de toets van Levene gebruikt. De toets van Levene is een statistische toets die een nulhypothese test. Deze nulhypothese stelt dat beide populaties beschikken over een gelijke variatie, ook wel

bekend als homoscedasticity.[33] Wanneer het resultaat van toets van Levene onder het significantie niveau valt, wordt de hypothese verstoten.

Wanneer blijkt dat de uitkomst van de toets van Levene onder het significantie niveau valt, kan met zekerheid vastgesteld worden dat de twee gebruikte datasets een verschillende variatie hebben. Hierdoor kunnen de gevonden resultaten geïnterpreteerd worden met een hogere betrouwbaarheid. Wanneer blijkt dat de uitkomst van de toets van Levene boven het significantie niveau valt, bijvoorbeeld bij 0,8 als uitkomst, kan er niet met zekerheid aangetoond worden dat onder andere de standaarddeviaties en betrouwbaarheidsintervallen betrouwbaar zijn. Er is in dit geval een kans van 80% (0,8) dat beide datasets beschikken over een gelijke variatie.

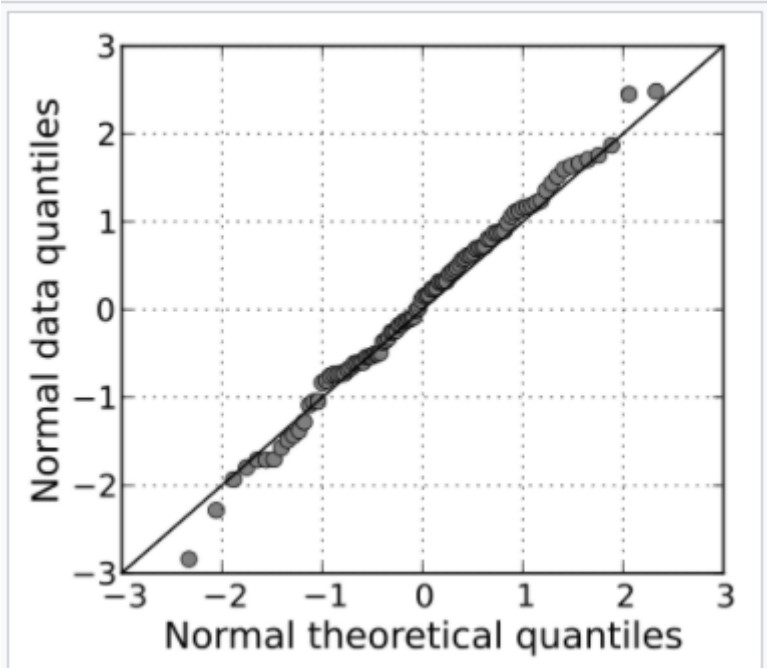
Voor het onderzoek is een significantie niveau aangehouden van 0,05. Dit niveau komt overeen met die van de betrouwbaarheidsintervallen. Daarnaast wordt dit niveau ook gehanteerd in de besproken onderzoeken in hoofdstuk 8.2.

Naast de toets van Levene kan de Brown-Forsythe test toegepast worden voor dezelfde doeleinden. De toets van Levene is geselecteerd in plaats van de Brown-Forsythe test, omdat deze betere resultaten levert voor normaal verdeelde populaties. De Brown-Forsythe test levert betere resultaten in het geval van niet normaal verdeelde populaties.[35] Uit het uitgevoerde onderzoek blijkt dat de gebruikte data normaal verdeeld is, waardoor de toets van Levene geselecteerd is. (zie hoofdstuk 12)

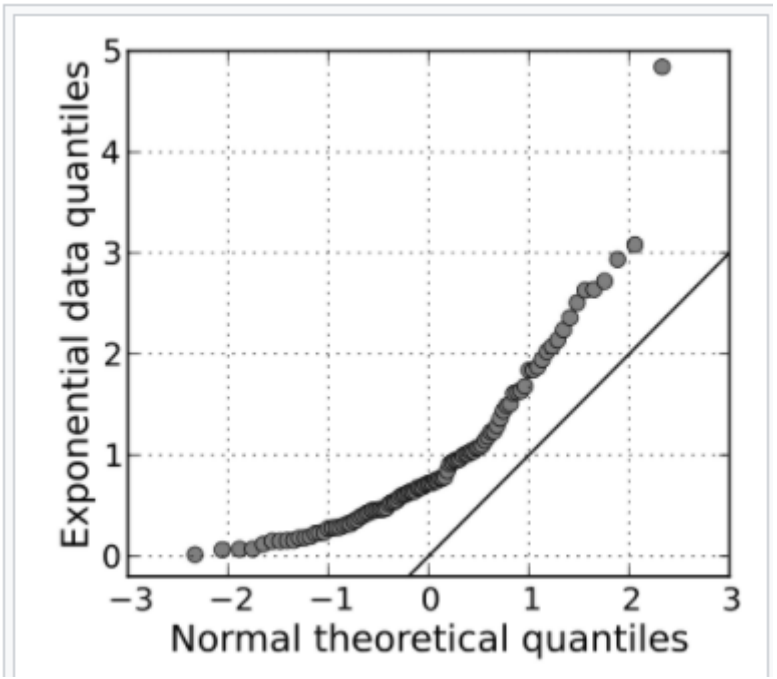
Q-Q plot

De Q-Q plot is een waarschijnlijkheidsgrafiek die de distributie van een dataset weergeeft. Met behulp van een Q-Q plot kan er vastgesteld worden of de geselecteerde dataset normaal verdeeld is of dat deze bijvoorbeeld een lognormale verdeling heeft. Een Q-Q plot wordt opgezet door de dataset te normaliseren en deze af te zetten op de y-axis. Op de x-axis worden datapunten afgezet die een normale verdeling volgen. Tot slot wordt er een trendline opgesteld, die een normale verdeling representeert.[30] Wanneer de datapunten deze trendline volgen is er sprake van een normaal verdeling, zoals weergeven in figuur 13. Wanneer de trendline niet gevolgd wordt is er sprake van een andere verdeling, zoals weergeven in figuur 14.

Wanneer twee datasets vergeleken worden, is het van belang dat deze beide normaal verdeeld zijn. Als dit niet het geval is zijn de resultaten die voortkomen uit de berekeningen niet betrouwbaar. Zo is het bij de toets van Levene een eis dat beide gebruikte datasets normaal verdeeld zijn. Dit geldt ook voor de betrouwbaarheidsintervallen.[32] Dit kan bepaald worden met behulp van de Q-Q plots.



Figuur 13: Voorbeeld Q-Q plot waarbij een normaal verdeling present is.[30]



Figuur 14: Voorbeeld Q-Q plot waarbij de data niet normaal verdeeld is [30]

Normalisatie van data

Wikipedia beschrijft data normalisatie als 'In statistics and applications of statistics, normalization can have a range of meanings In the simplest cases, normalization of ratings means adjusting values measured on different scales to a notionally common scale, often prior to averaging'[27]

Door het normaliseren van data is het mogelijk om meerdere datasets op een correcte manier met elkaar te vergelijken, terwijl deze verschillen in absolute waardes. Door het gebruik van de formule, zichtbaar in figuur 15, worden de geselecteerd datasets aangepast naar een gezamenlijke schaal. In het huidige onderzoek hanteert de schaal waardes tussen de nul en één. In tabel 5 is een voorbeeld weergegeven waar normalisatie op twee datasets wordt toegepast. Hierbij is te concluderen dat beide datasets zich na het normaliseren op dezelfde schaal bevinden, namelijk tussen de nul en één. Daarnaast is de genormaliseerde data met elkaar te vergelijken, terwijl de absolute data van datasets A en B dat niet is.

Tijdens het onderzoek blijkt dat er een verschil bestaat tussen de uitvoeringstijd op de geselecteerde cloud instances. Hierbij is het mogelijk dat een performance test op een specifieke cloud instance twee keer zo traag is dan diezelfde test op de on-premise omgeving. Alhoewel de test op de cloud omgeving trager is, hoeft dat niet te beteken dat deze ook beschikt over een hogere variabiliteit. Om deze twee datasets op een correcte manier te vergelijken wordt normalisatie toegepast.

```
X_new = (X - X_min)/(X_max - X_min)
```

Figuur 15: Formule data normalisatie [34]

Dataset A	Dataset B	Dataset A genormaliseerd	Dataset B genormaliseerd
8	270	0	0
9	278	0,09	0,42
12	280	0,36	0,52
15	283	0,63	0,68
17	286	0,81	0,84
19	289	1	1

Tabel 5: Voorbeeld van normalisatie

Data filteren

Wanneer blijkt dat de distributie van een dataset, weergegeven in een QQ-plot, aangetast wordt door extreme waardes kunnen deze waardes gefilterd worden. Door het uitsluiten van de extreme waardes kan in enkele gevallen de distributie van de desbetreffende dataset met een hogere betrouwbaarheid vastgesteld worden.[40] Het filteren van data gebeurd met de methodiek genaamd Winsorizing.[39] Deze methode wordt gebruikt voor het filteren van de extreme waardes in een dataset aan de hand van een betrouwbaarheidsniveau. Daarnaast kan het filteren van data de betrouwbaarheid van de standaarddeviatie verhogen.[41]

11.4 Aanpak

De stappen die zijn doorlopen om, met behulp van de statistische methodes, tot een conclusie te komen zijn hieronder beschreven. Hierbij wordt elke cloud instance vergeleken met de on-premise omgeving. Daarnaast zijn deze stappen gevisualiseerd in een flowchart, te vinden in Bijlage G: Flowchart resultaten.

Python applicatie

Tijdens de berekeningen wordt er gebruik gemaakt van een significante hoeveelheid data. Om het calculatieproces grotendeels te automatiseren is een Python programma ontwikkeld. Deze applicatie haalt onder andere de data op uit de verschillende CSV bestanden en voert de verschillende statistische methodes uit op de datasets. Tijdens meerdere stappen beschreven hieronder, worden code snippets uit het Python programma gebruikt, om zo een duidelijker beeld te schetsen van de het desbetreffende proces. Tevens biedt de ontwikkelde Python applicatie de opdrachtgevers de mogelijkheid om in de toekomst het onderzoek te herhalen, doordat merendeel van het statistische traject geautomatiseerd is. Hierdoor is Fonto in staat om in de toekomst meerdere vervolgonderzoeken te doen, door bijvoorbeeld de gevonden resultaten te vergelijken met andere cloud instances, zonder hierbij alle bijbehorende statistische berekeningen handmatig uit te hoeven voeren.

De programmeertaal Python is geselecteerd om meerdere redenen. Ten eerste is de nodige ervaring voor Python present, waardoor de opstart en het gebruik van Python eenvoudiger worden. Ten tweede beschikt Python over een aanzienlijke hoeveelheid interne en externe bibliotheken. Door het gebruik van deze interne en externe bibliotheken wordt een significante hoeveelheid tijd bespaard, mede doordat de statistische methodes niet herschreven worden.

Verschillende externe bibliotheken zijn geütiliseerd in de python applicatie, namelijk Numpy[36], Scipy[37] en Matplotlib[38]. Numpy en Scipy bieden de benodigde statistische methodes aan. Berekeningen van standaarddeviatie of de normalisatie van data worden

uitgevoerd door middel van de functies aanwezig in de Numpy bibliotheek. Het berekenen van de QQ-plots en de toets van Levene worden bewerkstelligd door de Scipy bibliotheek. Daarnaast is een module, genaamd Pylab, afkomstig uit de Matplotlib bibliotheek gebruikt om de QQ-plots visueel weer te geven. Deze bibliotheken zijn geselecteerd doordat ze beschikken over de juiste functionaliteit en daarnaast eenvoudig geïmplementeerd kunnen worden.

Stap 1: cloud data verzamelen

Zoals vermeld in hoofdstuk 9 is de data van de verschillende cloud instances verzameld in een PostgreSQL database. Deze data is omgezet naar een CSV formaat, omdat het uitvoeren van calculaties op de data hiermee eenvoudiger wordt. Integratie met het python programma kost hierdoor minder tijd. Hierbij zijn er tijdens de uitvoering van de testen 140 testresultaten verzameld per test en browser combinatie, op elke cloud instance.

Stap 2: on-premise data verzamelen

Naast de data afkomstig van de cloud instances is de data van de on-premise omgeving opgehaald. Het gaat hier om de testresultaten van de test runner, die gemeten zijn over een periode van een jaar. Deze data is, net zoals de cloud instance data, opgeslagen in een PostgreSQL database die in beheer is van Fonto. Deze data is verkregen in CSV formaat en is beschikbaar gesteld door de databasebeheerder bij Fonto, genaamd Erik van der Valk. Omdat elke uitgevoerde test en browser combinatie beschikt over 140 testresultaten per cloud instance, zullen er tijdens de uitvoering van de statistische methodieken evenveel testresultaten gebruikt worden afkomstig van de on-premise omgeving.

Stap 3: data normaliseren

De data van beide omgevingen is lokaal opgeslagen, waardoor het mogelijk wordt om verschillende berekeningen uit te voeren. Allereerst wordt de data genormaliseerd, zodat de berekeningen die later volgen een betrouwbaar resultaat opleveren. Het normaliseren van data kan volgens verschillende formules. In figuur 15 is de standaard formule voor normalisatie weergegeven. Hier wordt uitgegaan dat de laagste en hoogste waarde van de dataset bekend zijn, omdat deze gebruikt worden in de formule. Aangezien de vergaarde data van de on-premise en verschillende cloud instances een momentopname is, zijn de laagste en hoogste waarde niet bekend. De dataset is immers een sample en geen populatie. Hierdoor is de keuze gemaakt om in plaats van de laagste en hoogste waarde, de gemiddelde waarde te gebruiken. Doordat de laagste en hoogste waardes niet bekend zijn biedt het delen door de gemiddelde waarde, resultaten met een hogere betrouwbaarheid.

De data wordt genormaliseerd met behulp van een Python bibliotheek, genaamd Numpy.[36] Hieronder is de code weergegeven die de data, met behulp van het gemiddelde,

normaliseert. Allereerst wordt de data, afkomstig van een van de cloud instances, van een specifieke test op de Chrome of Firefox browser opgehaald. Deze data wordt met behulp van de Numpy bibliotheek gedeeld door het gemiddelde. Dit proces is identiek voor de datasets afkomstig van de on-premise omgeving.

```
def getCloudDataNorm(browser, test, instance):
   rawCloudData = getCloudData(browser, test, instance)
   n = [float(i)/numpy.average(rawCloudData) for i in
rawCloudData]
   return n
```

Code snippet 1: Normalisatie van een dataset

Uitvoertijd in milliseconden	Genormaliseerde uitvoertijd
40,367	0,935
40,585	0,940
40,642	0,941
40,951	0,950

Tabel 6: Enkele genormaliseerde waardes uit een on-premise dataset ter voorbeeld

Stap 4: QQ-plot genereren

De data is genormaliseerd en gesorteerd om zo een QQ-plot te creëren. Zoals eerder vermeld wordt er voor elke uitgevoerde test op de on-premise omgeving en op de cloud instances een QQ-plot gegenereerd. De QQ-plot van een test op de on-premise omgeving wordt vergeleken met de zes QQ-plots, afkomstig van de cloud instances met als doel om vast te stellen welke distributie elke dataset heeft.

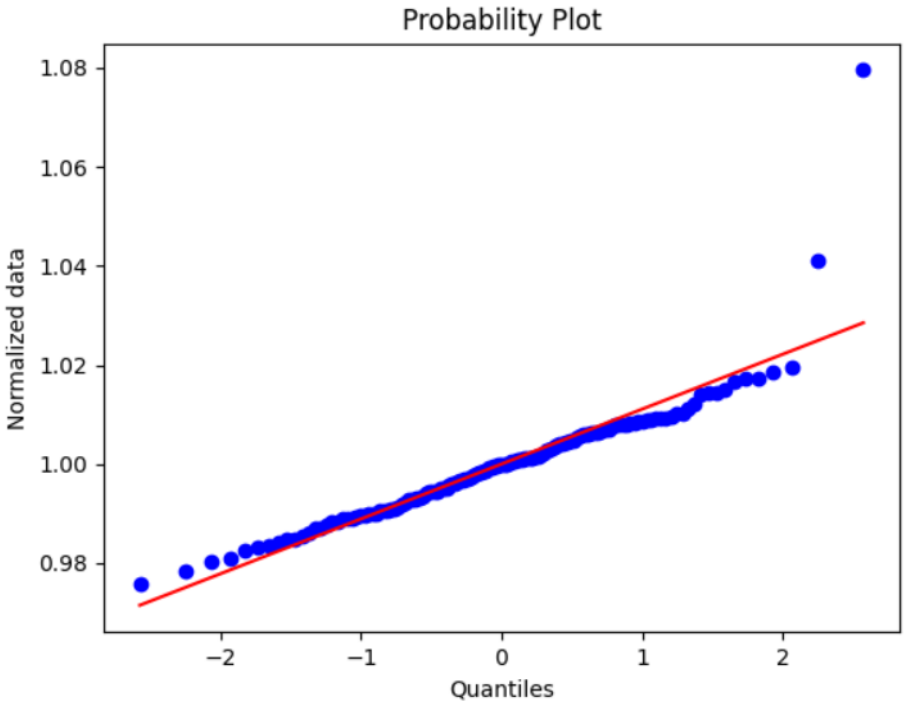
In code snippet 2 is de functionaliteit weergegeven die door middel van een dataset een QQ-plot genereert. De QQ-plot wordt gegenereerd door middel van de Scipy bibliotheek.[37] Deze wordt door middel van een Matplotlib module, genaamd Pylab, zichtbaar gesteld op het scherm.[38]

```
def plotQQ(measurements):
    scipy.probplot(measurements, dist="norm", plot=pylab)
    pylab.show()
```

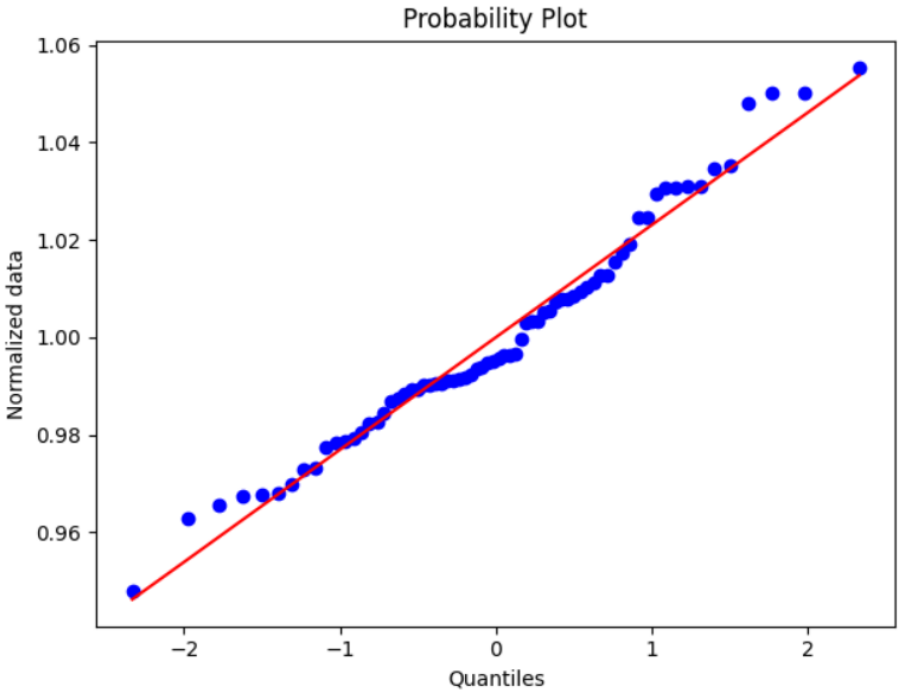
Code snippet 2: Plotten QQ-plot

Wanneer blijkt dat een specifieke test niet normaal verdeeld is kunnen de extreme waardes gefilterd worden, zoals weergeven in Bijlage G: Flowchart resultaten. Zodra het filteren van de data niet leidt tot een normaal verdeelde distributie, kan de data niet op een betrouwbare vergeleken worden. Uit de QQ-plots bleek dat alle uitgevoerde testen normaal verdeeld zijn. Hierbij komt het regelmatig voor dat de testen extreme waardes

bevatten. De mate waarin deze extreme waardes voorkomen is laag, gemiddeld bevat een dataset twee tot drie extreme waardes, en beïnvloedt daarom niet de distributie. In figuur 16 is een voorbeeld weergeven van een QQ-plot waaruit een normale distributie blijkt, terwijl deze beschikt over enkele extreme waardes. Op de y-axis is de genormaliseerde data afgezet, op de x-axis de quantile die benodigd zijn voor een QQ-plot. Deze quantile worden berekende door de Scipy bibliotheek en geven een normale verdeling aan. De trendlijn is weergegeven met de kleur rood en geeft een normaal verdeelde distributie aan. Figuur 17 geeft de resultaten weer van dezelfde test als bij figuur 16, alleen dan uitgevoerd op de standaard Azure instance. Hieruit is te concluderen dat beide beschikken over een normaal verdeelde distributie, maar dat de standaard Azure instance meer afwijkende waardes heeft ten opzichte van een normaal verdeelde distributie. Over de variabiliteit kan in dit stadia nog geen conclusie getrokken worden. In bijlage H: QQ-plots zijn de QQ-plots uit figuur 16 en 17 opgenomen, in combinatie met de andere instances. Hierbij zijn de QQ-plots weergegeven voor de "*findAndScroll*" test. De QQ-plots afkomstig van deze test zijn representatief voor de andere testen, omdat de overige cloud instances vergelijkbare resultaten opleveren. Hierdoor zijn deze testresultaten opgenomen in de bijlagen.



Figuur 16: QQ-plot afkomstig van de test "findAndScroll" uit de on-premise omgeving



Figuur 17: QQ-plot afkomstig van de test "findAndScroll" uit de standard Azure instance

Stap 5: toets van Levene uitvoeren

De toets van Levene kan uitgevoerd worden op de datasets nu is gebleken dat deze beschikken over een normaal verdeelde distributie. In de code snippet hieronder is de functie weergegeven die de toets van Levene berekend, aan de hand van twee datasets. In de context van het onderzoek zijn dat de datasets van een cloud instance in combinatie met een dataset van de on-premise omgeving. Elke cloud instance moet hierbij immers vergeleken worden met de on-premise omgeving. De toets van Levene wordt berekend door middel van de Scipy bibliotheek.[37]

```
def calculateLevene(instanceNorm, premiseNorm):
    levene = scipy.stats.levene(instanceNorm, premiseNorm)
    return levene
```

Code snippet 3: Uitvoeren van de toets van Levene

Uit deze toets blijkt hoe groot de kans is dat beide datasets beschikken over dezelfde variabiliteit. Wanneer het resultaat van deze toets onder het significantieniveau van 0.05 uitkomt, wordt er aangenomen dat beide datasets beschikken over een verschillende variatie. De gevonden standaarddeviaties en betrouwbaarheidsintervallen bieden in dit geval een hogere betrouwbaarheid.

Stap 6: betrouwbaarheidsinterval berekenen

Om de mate van variabiliteit binnen een dataset vast te stellen wordt het betrouwbaarheidsinterval berekend. Door middel van het betrouwbaarheidsinterval kan de

on-premise omgeving vergeleken worden met een van de cloud instances. Hierbij wordt de stabiliteit van beide omgevingen vastgesteld.

In de code snippet hieronder wordt met behulp van de Scipy bibliotheek het betrouwbaarheidsinterval van een genormaliseerde dataset berekend. Hierbij bevat de CI variable twee waardes, genaamd de boven-en ondergrens. De boven-en ondergrens van de cloud instance kunnen vervolgens vergeleken worden met die van de on-premise omgeving.

```
def calculateCI(datasetNorm):
    CI = scipy.stats.norm.interval(alpha=0.95,
    loc=np.mean(datasetNorm),scale=np.std(datasetNorm))
    return CI
```

Code snippet 4: Betrouwbaarheidsinterval berekenen

Resultaten inventariseren

Voor elke test uitgevoerd op de zes cloud instances en de on-premise omgeving is een QQ-plot opgesteld, is de toets van Levene berekend en is het betrouwbaarheidsinterval berekend. Deze resultaten zijn vergaard en worden in het volgende hoofdstuk besproken.

12.0 Conclusies

De conclusies afkomstig van de statistische resultaten worden hieronder besproken. Daarnaast wordt antwoord gegeven op de deelvraag "Bieden performance testresultaten, afkomstige van een cloud instance, dezelfde betrouwbaarheid als performance testresultaten afkomstig van de on-premise omgeving?".

12.1 Distributie

Door het gebruik van QQ-plots zijn de distributies van de testen op de verschillende omgevingen vastgelegd. Hieruit blijkt dat alle datasets afkomstig van de uitgevoerde testen beschikken over een normaal verdeelde distributie. Hierbij verschilt het per dataset in welke mate deze concordeert met de trendline. In bijlage H: QQ-plots is weergegeven dat de compute GCE instance minder sterk correspondeert met de trendline, in vergelijking met de compute Azure instance. Verder is de data filter methodiek Winsorizing niet toegepast. Hoewel de meeste datasets beschikte over enkele extreme waardes, taste deze de distributie niet op een significante manier aan. Hierbij is de Winsorizing methodiek achterwege gelaten.

12.2 Toets van Levene

De toets van Levene geeft aan of een test uitgevoerd op een cloud instance dezelfde mate van variatie heeft als de on-premise omgeving. De nulhypothese van deze toets stelt dat beide populaties beschikken over een gelijke variatie, ook wel bekend als homoscedasticity. Uit de berekeningen blijkt dat bij het merendeel van de datasets, afkomstig van de uitgevoerde testen, de null hypothese verstoten wordt. De nulhypothese wordt verstoten wanneer het resultaat van de toets van Levene onder het significantie niveau valt, wat in het onderzoek gedefinieerd staat als 0,05. Hierbij wordt geconcludeerd de variatie bij een test uitgevoerd op een cloud instance en de on-premise omgeving heteroscedastisch zijn, oftewel niet gelijk.

Bij een van de testen, genaamd "*arrowKeys*", is de nulhypothese op de twee van de cloud instances niet verstoten. Een verklaring hiervoor is de volatiliteit van de test. In bijlage C: Testselectie is de mate van ruis in de "*arrowKeys*" test weergegeven. Doordat deze test een hogere mate van ruis bevat zijn de gemeten resultaten over het algemeen minder nauwkeurig. Daarnaast beschikt de test over een relatief korte uitvoeringstijd rond de vijf tot tien milliseconden, waardoor de metingen omtrent de uitvoeringstijd over het algemeen minder betrouwbaar zijn dan testen met een lange uitvoeringsduur.

12.3 Variabiliteit

Voor elke uitgevoerde test op de cloud instances en de on-premise omgeving is een betrouwbaarheidsinterval berekend met een betrouwbaarheidsniveau van 95%. Elk interval beschikt over een onder-en bovengrens.

Hieronder wordt de berekening verder toegelicht aan de hand van een voorbeeld. Dit voorbeeld gebruikt een willekeurig geselecteerde test en cloud instance, namelijk de "*findAndScroll*" test uitgevoerd op de compute Azure instance. Hierbij is de test uitgevoerd op Firefox.

In de twee tabellen hieronder zijn de betrouwbaarheidsintervallen voor de genormaliseerde en absolute data opgenomen. De absolute data is hierbij opgenomen als vergelijkingsmiddel. De genormaliseerde data is abstract, waarbij de absolute data weergegeven wordt in milliseconden. In de tabellen zijn de onder-en bovengrens van het betrouwbaarheidsinterval opgenomen. Daarnaast is het verschil van deze twee waardes opgenomen. Door de verschillen van de cloud instance te vergelijken met de on-premise omgeving wordt er een conclusie getrokken over de stabiliteit van de desbetreffende omgeving. In tabel 7 is weergegeven dat de variabiliteit van de compute Azure instance in vergelijking met de on-premise omgeving -65,94% is. Hieruit is geconcludeerd "*findAndScroll*" test, uitgevoerd in Firefox, minder stabiel is op de Azure compute instance, dan de on-premise omgeving.

In tabel 8 is de absolute data weergegeven, deze bevat de uitvoertijd in milliseconden. Hierbij is de kans 95% dat een uitvoering van de test "*findAndScroll*" op de compute Azure instance een uitvoeringstijd heeft afgerond tussen de 126 en 131 milliseconden. Hetzelfde proces uit tabel 7 geldt hier voor de absolute data. Hieruit blijkt dat de compute Azure instance een spreiding heeft van afgerond 15 milliseconden waarbij de on-premise omgeving een spreiding heeft van afgerond 3 milliseconden. Daarnaast is het procentuele verschil hoger dan bij de genormaliseerde data. Dit komt doordat de testen op de on-premise een hogere uitvoeringstijd hebben. Door de data te normaliseren wordt dit verschil weggewerkt. De absolute data wordt in combinatie met de genormaliseerde data gebruikt om een completer beeld van de situatie te schetsen.

Machine	Ondergrens	Bovengrens	Verschil	Procentueel verschil compute AWS en on-premise
Compute Azure	0,9809	1,0190	0,0381	-65,94%
On-premise	0,9935	1,0065	0,0130	

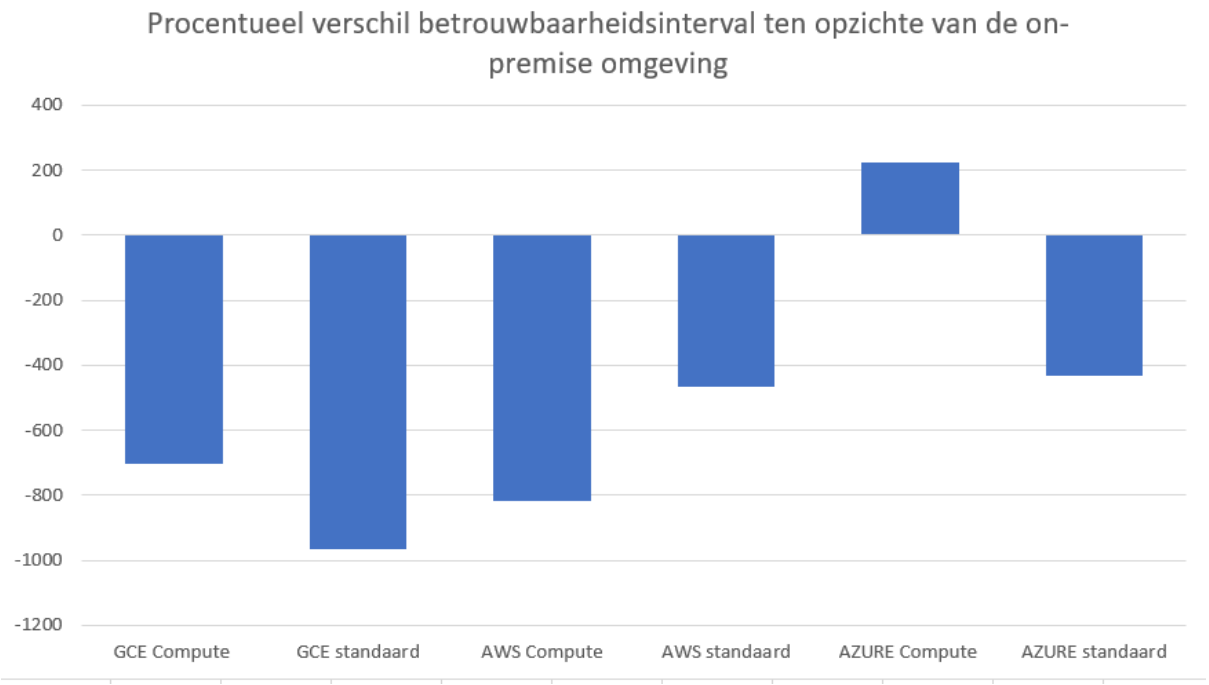
Tabel 7: Betrouwbaarheidsinterval genormaliseerde data

Machine	Ondergrens (ms)	Bovengrens (ms)	Verschil (ms)	Procentueel verschil compute AWS en on-premise
Compute Azure	125,8983	130,7992	14,6340	-79,68%
On-premise	75,2325	76,2174	2,9730	

Tabel 8: Betrouwbaarheidsinterval absolute data in milliseconden

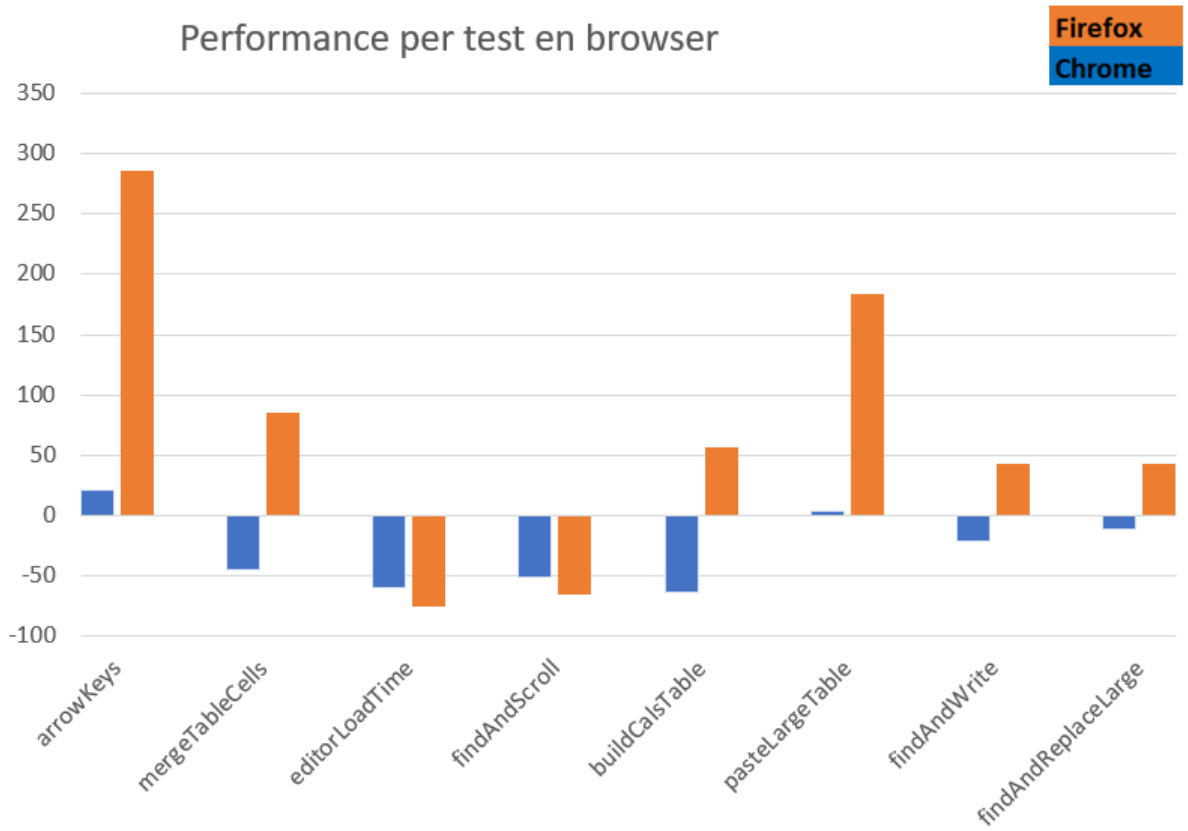
12.4 Uitkomsten

De resultaten van de berekeningen zijn tentoongesteld in meerdere grafieken. In figuur 18 is het procentuele verschil van de betrouwbaarheidsintervallen ten opzichte van de on-premise omgeving opgenomen. Hieruit is geconcludeerd dat de compute Azure instance de minste variabiliteit heeft in relatie tot de cloud andere omgevingen. Daarnaast is de compute Azure instance gemiddeld gezien stabieler dan de on-premise omgeving. Verder presteren de GCE en Azure compute instances over het algemeen beter dan de standaard instances, waarbij de standaard instance gehost op AWS beter presteert dan de compute instance gehost op AWS.



Figuur 18: Procentueel verschil van het betrouwbaarheidsinterval ten opzichte van de on-premise omgeving

De compute Azure instance biedt de meest stabiele resultaten van de geselecteerde cloud instances. Daarom wordt deze instance op basis van andere parameters met de on-premise omgeving vergeleken, om zo tot een completere conclusie te komen. In figuur 19 is voor elke test het procentuele verschil tussen de onder-en bovengrens van het betrouwbaarheidsinterval van de compute Azure instance en de on-premise omgeving weergegeven. Daarnaast is er onderscheid gemaakt tussen testen uitgevoerd op Firefox en Chrome. Hierbij zijn de testen uitgevoerd op Chrome blauw gekleurd en de testen uitgevoerd op Firefox Oranje gekleurd. De x-axis geeft de gebruikte test. De y-axis geeft het procentuele verschil tussen de onder-en bovengrens van het betrouwbaarheidsinterval weer, van de compute Azure instance ten opzicht van de on-premise omgeving. Uit figuur 19 is geconcludeerd dat de stabiliteit per test een significante hoeveelheid afwijkt. Zo bieden enkele testen zoals “*arrowKeys*” en “*pastLargeTable*” een betere stabiliteit op de compute Azure instance en bieden testen zoals “*editorLoadTime*” en “*findAndScroll*” op de on-premise omgeving een hogere stabiliteit.



Figuur 19: Procentueel verschil van het betrouwbaarheidsinterval van de compute Azure instance ten opzichte van de on-premise omgeving

Uit de toets van Levene blijkt dat de resultaten afkomstig van de "*arrowKeys*" test minder betrouwbaar zijn. Daarnaast heeft deze test veel impact op de algemene variabiliteit van de Azure compute instance, omdat het verschil tussen de betrouwbaarheidsinterval nagenoeg 300% is. Wanneer de resultaten van deze test niet meegenomen worden in de berekening weergegeven in figuur 19 verandert het procentuele verschil van de Azure compute instance van 220% naar -60%.

12.5 Antwoord op de deelvraag

De deelvraag "Bieden performance testresultaten, afkomstige van een cloud instance, dezelfde betrouwbaarheid als performance testresultaten afkomstig van de on-premise omgeving?" kan met behulp van de onderzoeksconclusie beantwoord worden. De betrouwbaarheid van de performance testresultaten is afhankelijk van de geselecteerde cloud provider in combinatie met de geselecteerde cloud instance. Uit het onderzoek blijkt dat de F2 instance, gehost door de Microsoft Azure, de selectie testen met nagenoeg dezelfde betrouwbaarheid kan uitvoeren in vergelijking met de on-premise omgeving. Hierbij verschilt de variabiliteit per test, ten opzichte van de on-premise omgeving. Enkele

testen bieden een significant hogere variabiliteit en andere bieden een significant lagere variabiliteit.

In hoofdstuk 3 is aangegeven dat regressies vanaf 10 tot 20% worden genoteerd, maar dat de focus ligt op significante regressies waarbij de uitvoeringstijd van een test verdubbeld. Alhoewel de F2 instance, na het buitensluiten van de "*arrowKeys*" test, gemiddeld 60% volatieler is dan de on-premise omgeving, beschikt de F2 instance over voldoende betrouwbaarheid om de gewenst regressies waar te nemen. Hoewel hiermee regressies van 10% niet meer met betrouwbaarheid waargenomen kunnen worden, biedt de F2 instance Fonto de mogelijkheid om regressies vanaf 20% met een hoge betrouwbaarheid waar te nemen.

De overige instances beschikken over een te hoge variabiliteit om de benodigde regressies waar te nemen. Deze instances, zoals weergeven in figuur 18, beschikken over een variabiliteit die vier tot tien keer zo groot is dan de on-premise omgeving, waardoor regressies van 10 tot 100% niet met zekerheid kunnen worden vastgesteld.

13.0 Software architectuur

Uit het onderzoek blijkt dat de Microsoft Azure F2 instance een valide vervanging is voor de on-premise omgeving. Hieronder wordt de software architectuur van de test runner in een cloud omgeving behandeld, waarbij de requirements en ontwerpmethodiek worden toegelicht.

13.1 Orchestrator

Zoals eerder beschreven in hoofdstuk 3 bestaat de huidige opzet uit één fysieke machine, gealloceerd bij Fonto, waarop de test runner actief is. Het voornaamste probleem bij deze opzet is het onvermogen om op te schalen, de test runner beschikt namelijk maar over één fysieke machine. Uit het hiervoor uitgevoerde onderzoek blijkt dat de cloud een valide omgeving is om de performance testen van Fonto in uit te voeren.

De cloud biedt de mogelijkheid om eenvoudig nieuwe instances te initialiseren en benutten. Om deze instances te beheren en aan te sturen wordt er een prototype ontwikkeld. Om dit te realiseren worden de requirements geïnventariseerd en wordt een software architectuur opgesteld. Fonto wenst hierbij een orchestrator oplossing, zoals vermeld in hoofdstuk 4.2. De orchestrator is een programma die verschillende workers beheert en aanstuurt. Een worker is in de huidige context een cloud instance die een selectie van performance testen uitvoert.

13.2 Requirements

In de tabel hieronder zijn de requirements van het prototype, genaamd test runner 2.0, weergeven. De requirements zijn vergaard door middel van gesprekken met Jos Verburg en andere stakeholders bij Fonto. Tijdens deze gesprekken is het doel van de test runner besproken en zijn de benodigde functionaliteiten die ontwikkeld moeten worden om het doel te realiseren behandeld.

ID	Requirement
1	De performance testen worden uitgevoerd op een F2 Microsoft Azure instance.
2	Een F2 instance beschikt over de laatste versie van de benodigde editors.
3.1	Een F2 instance kan Chrome opstarten.
3.2	Een F2 instance kan Firefox opstarten.
3.3	Een F2 instance kan Edge Legacy opstarten.
4	Een F2 instance slaat de logs lokaal op.
5	Een F2 instance moet automatische gedeployed kunnen worden.
6	De editors moeten automatisch geupdate worden na een nieuwe build in de productie pipeline.
7	De editors moeten automatisch geupdate worden na een nieuwe build in de nightly pipeline.
8	Nieuwe performance testen kunnen worden toegevoegd aan de orchestrator.
9	Nieuwe editors kunnen worden toegevoegd aan de orchestrator.
10	Nieuwe editors kunnen worden toegevoegd aan cloud instances.
11	De performance testresultaten van een uitgevoerde test worden opgeslagen in een database.
12	De logs moeten buiten de F2 instance worden opgeslagen.
13	De logs moeten toegankelijk zijn via een web interface.
14	De performance testresultaten worden weergegeven in Grafana, in de vorm van een grafiek.
15	De status van de performance testen wordt weergeven in Slack.

Tabel 9: Requirements test runner 2.0

Nadat de requirements zijn geïnventariseerd worden deze opgenomen in een software architectuur. De software architectuur weergeeft het volledige ontwerp voor de nieuwe test

runner, waarbij tijdens de realisatiefase enkele componenten uit het ontwerp ontwikkeld worden als Proof of Concept (POC). De opgestelde software architectuur maakt de communicatie omtrent design keuzes en functionaliteiten eenvoudiger. Daarnaast kan Fonto, na afloop van de afstudeerperiode, aan de hand van de software architectuur de ontwikkeling van de test runner 2.0 voortzetten.

Voor het realiseren van de software architectuur is C4 is geselecteerd als ontwerpmethodiek.[45] Een software architectuur wordt binnen C4 weergegeven aan de hand van verschillende abstractieniveaus. Deze ontwerpmethodiek is geselecteerd omdat de nodige ervaring voor C4 present is, waardoor het ontwerpproces effectiever verloopt. Verder geven de stakeholders bij Fonto de voorkeur aan een software architectuur die ontworpen is met C4 als ontwerpmethodiek.

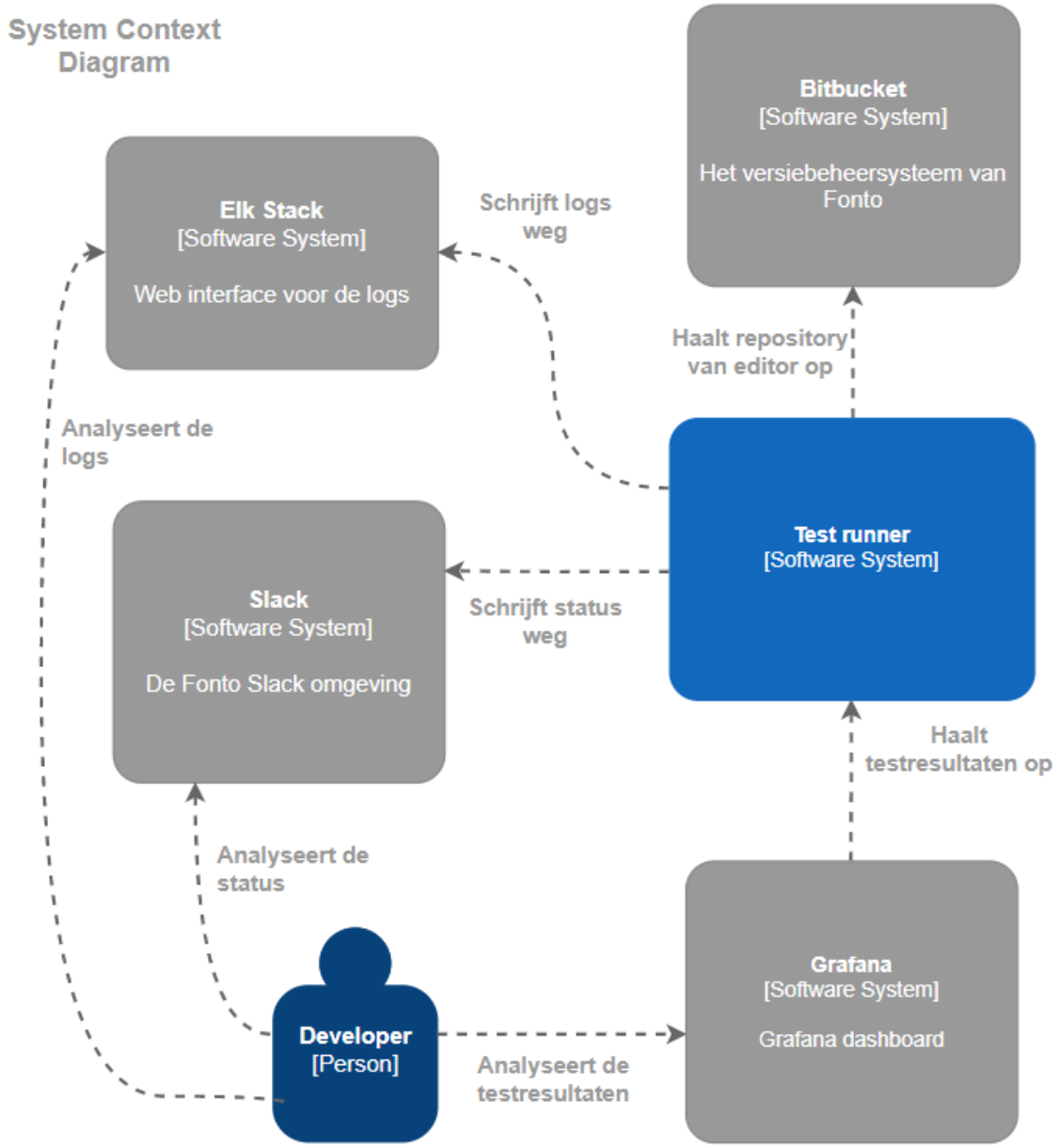
Op de volgende pagina's is de opgestelde software architectuur toegelicht. Voor meer informatie over het proces van C4 kan de Bijlage K: hoe werkt C4, geraadpleegd worden.

13.3 Het ontwerp

De opgestelde software architectuur wordt hieronder toegelicht. Daarbij worden de system context, container en component diagrammen behandeld. De diagrammen op code niveau zijn op advies van de C4 methodiek achterwege gelaten, omdat deze te gedetailleerd zijn.[45]

System context diagram

Het system context diagram bevat de software systemen op de hoogste abstractie. De softwaresystemen weergeven in figuur 20 worden hieronder toegelicht. Softwaresystemen die blauw gekleurd zijn geven systemen aan die ontwikkeld moeten worden. De grijs gekleurde softwaresystemen zijn van belang voor de werking van de blauw gekleurde systemen, maar hoeven niet ontwikkeld te worden, omdat deze al bestaan.



Figuur 20: System context diagram

Test runner

Het software systeem genaamd test runner bevat alle componenten en functionaliteiten die ontwikkeld gaan worden. Deze bevat onder andere de orchestrator, cloud instances en databases.

Elastic Stack (ELK Stack)

De ELK Stack is een combinatie van meerdere open source programma's, namelijk Kibana, Beats en Logstash.[47] De ELK Stack biedt de mogelijkheid om de logs afkomstig van de cloud instances op te slaan op een gedeelde locatie, die toegankelijk is via een web interface. De ELK Stack is geselecteerd omdat het voldoet aan de requirements en op advies van de opdrachtgever. De cloud instances schrijven de logs weg naar de ELK Stack, waarbij de developer deze kan inzien via een web interface.

Bitbucket

Fonto maakt gebruik van Bitbucket als versiebeheersysteem, die onder andere de repositories van de editors bevat. Wanneer een van de cloud instances een performance test uitvoert, moet deze beschikken over de meest recente versie van de benodigde editor. Voordat een performance test uitgevoerd wordt, haalt de cloud instances de benodigde editors op uit de repository.

Slack

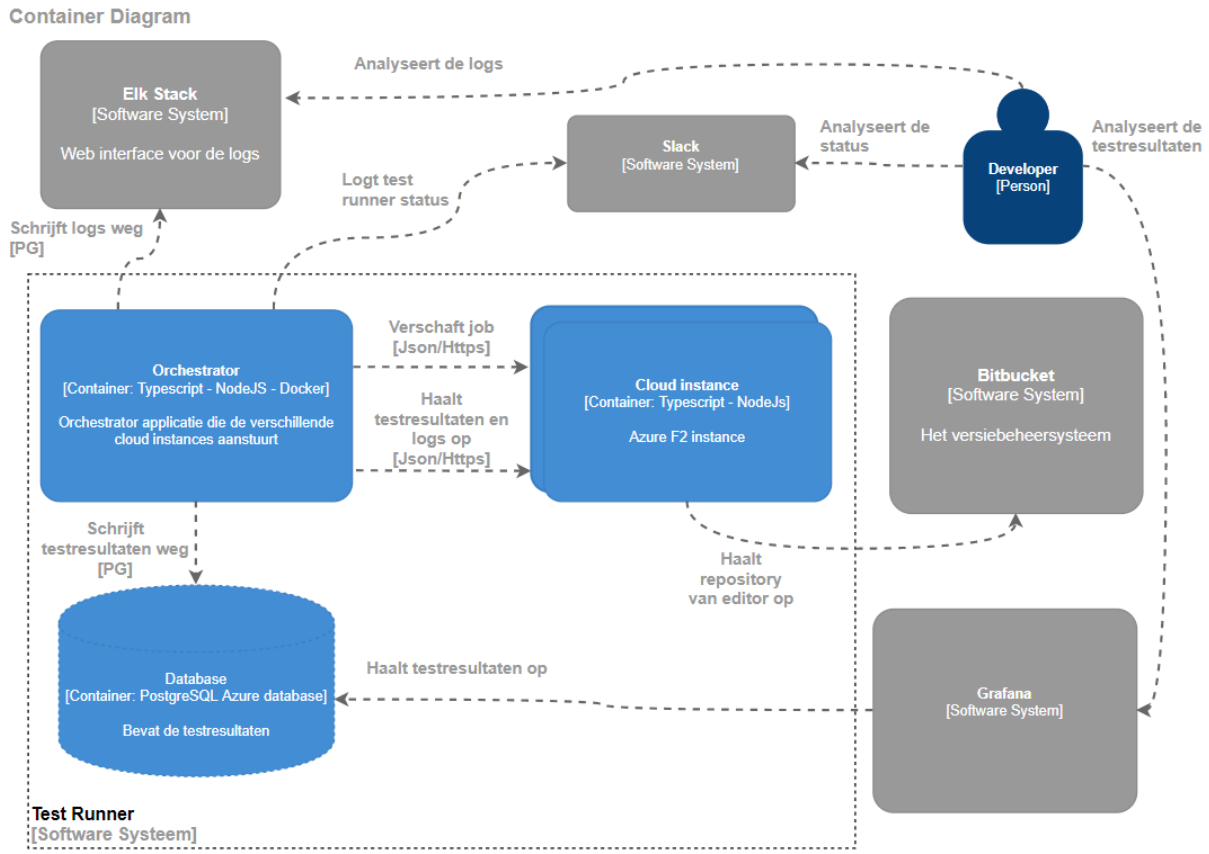
Momenteel wordt na elke uitvoering van de test runner informatie verstuurd naar Slack, waarna het weergegeven wordt in een Slack kanaal. Deze informatie moet in de test runner 2.0 ook gelogd worden naar een Slack kanaal. De developer kan door middel van het Slack kanaal de status van de verschillende cloud instances monitoren.

Grafana

In de huidige opzet van de test runner worden de testresultaten weergeven in Grafana. De test runner 2.0 moet, zoals aangeduid in tabel 9, ook beschikken over deze functionaliteit. Het Grafana dashboard haalt de testresultaten op uit een database, die gedefinieerd is binnen de het softwaresysteem genaamd test runner. De developer kan hierbij de testresultaten inzien in Grafana.

Container diagram

Binnen het container diagram is het softwaresysteem genaamd test runner, uit figuur 20, tot op meer detail beschreven. Hieronder is het diagram weergeven en zijn de drie toegevoegde containers toegelicht.



Figuur 21: Test runner container diagram

Orchestrator

Zoals eerder aangegeven wenst Fonto een orchestrator oplossing. Hierbij stuurt een programma genaamd de orchestrator de beschikbare cloud instances aan en verwerkt het de logs en performance testresultaten afkomstig van de cloud instances. Allereerst verschaft de orchestrator een job aan de cloud instances. Een job bevat de informatie die benodigd is om één performance test uit te voeren, zoals weergegeven in het figuur hieronder.

```
{
  "RepositoryURL": "https://DijckD@bitbucket.org/DijckD/fontoxml-app-dita-sandbox-editor-production",
  "branch": "develop",
  "browsers": "chrome",
  "editor": "fontoxml-app-dita-sandbox-editor-production",
  "environment": "production",
  "path": "fontoxml-app-dita-sandbox-editor-production",
  "tests": [
    {
      "name": "findAndScroll",
      "documentIds": "performance/clogs.ditamap"
    }
  ]
}
```

Figuur 22: Inhoud van een job in JSON formaat

De inhoud van een job kan omgezet worden naar een object, die vergelijkbaar is met de inhoud van het testConfig.json, beschreven is in hoofdstuk 3. In de huidige opstelling voert de test runner de performance testen uit aan de hand van het testConfig.json bestand. Om de flexibiliteit van een cloud instance te verhogen wordt het testConfig.json bestand verschaft door de orchestrator. Hierdoor kan de developer op één centraal punt (orchestrator) de testconfiguratie van de verschillende cloud instance beheren. De waarde gedefinieerd als "*RepositoryURL*" wordt niet opgenomen in het testconfiguratie object, maar wordt gebruikt om de juiste editor op te halen uit de repository. Wanneer een performance test wordt uitgevoerd op een cloud instance behoort deze gebruik te maken van de meest recente versie van de editor, zodat verandering in de editor direct getest worden. Het testConfig.json bestand dat aanwezig is op de orchestrator wordt hiernaast hernoemd naar jobConfig.json.

De developer creëert daarnaast een selectie testen voor elke cloud instance, waarbij dezelfde performance testen telkens uitgevoerd worden op dezelfde cloud instance. Hierdoor zijn de vergaarde performance testresultaten betrouwbaarder, omdat deze afkomstig zijn van dezelfde cloud instance. Het nadeel van deze aanpak betreft de schaalbaarheid. Wanneer een nieuwe cloud instance wordt toegevoegd zullen de test selecties voor elke cloud instance veranderd moeten worden, om testen vrij te maken voor de nieuw toegevoegde cloud instance. De opdrachtgever heeft aangegeven dit geen probleem te vinden, mede doordat het creëren van test selecties geautomatiseerd kan worden en doordat Fonto niet van plan is om frequent cloud instances toe te voegen. Wanneer nieuw gecreëerde performance testen worden toegevoegd aan de orchestrator, kunnen deze uitgevoerd worden op een nieuw gecreëerde cloud instance waardoor de test selecties voor de overige cloud instance identiek blijven.

verder slaat de orchestrator de ontvangen performance testresultaten op in een database, slaat het de logs op in ELK Stack en schrijft het een status weg naar Slack. De testresultaten en logs worden niet opgeslagen op de orchestrator, omdat de orchestrator hierdoor stateless wordt. Dit houdt in dat de orchestrator geen waardevolle data opslaat, waardoor deze eenvoudig opnieuw te deployen is. Hierbij kan de orchestrator omgezet worden naar een Docker image, waarbij deze zonder het testproces te verstoren opnieuw gedeployed kan worden.

Cloud instance

Een cloud instance is verantwoordelijk voor het uitvoeren van de performance testen, het rapporteren van de performance testresultaten, het rapporteren van de logs en behoort daarnaast gebruik te maken van de meest recente versie van een editor. Elke cloud instance bevat een aangepast versie van de test runner programmatuur, die verantwoordelijk is voor het uitvoeren van de performance testen. De keuze om de huidige test runner aan te passen is gebaseerd op het feit dat een significant deel van de huidige test runner hergebruikt kan worden. Hierdoor is het beduidend efficiënter om de huidige

test runner aan te passen, ten opzichte van de realisatie van een nieuw systeem. De gemaakte aanpassingen in de test runner worden in het component diagram tot op meer detail beschreven. Naast de orchestrator zijn de cloud instances ook stateless. De performance testresultaten en logs worden via de orchestrator opgeslagen in de database en ELK Stack. De logs worden hierbij, als requirement van de opdrachtgever, wel opgeslagen op de cloud instances als back-up.

Database

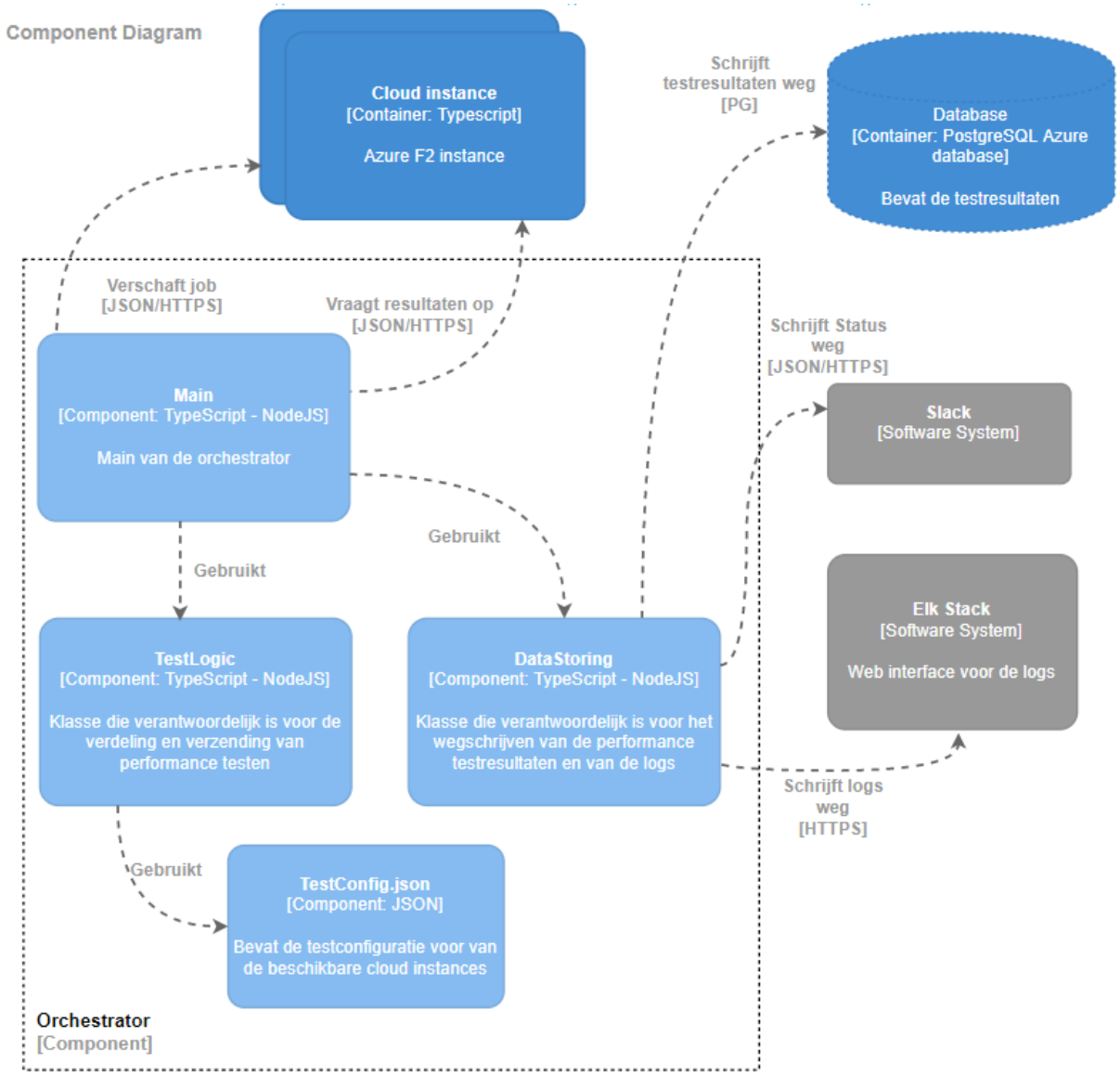
De vergaarde performance testresultaten worden opgeslagen in een postgreSQL database. Hierbij wordt de huidige database configuratie hergebruikt, omdat de informatie die opgeslagen wordt in de database niet veranderd ten opzichte van de huidige situatie. De performance testresultaten worden in de database weggeschreven door middel van de NodeJS bibliotheek genaamd PG.[48] Deze bibliotheek is geselecteerd doordat de huidige test runner gebruik maakt van de PG bibliotheek.

Component diagrams

Binnen de component diagrammen zijn de containers genaamd orchestrator en cloud instances, uit figuur 21, tot op meer detail beschreven. Hieronder worden deze component diagrammen toegelicht.

Orchestrator component diagram

Het component diagram van de orchestrator is hieronder weergegeven, waarbij de toegevoegde klassen zijn toegelicht.



Figuur 23: Orchestrator component diagram

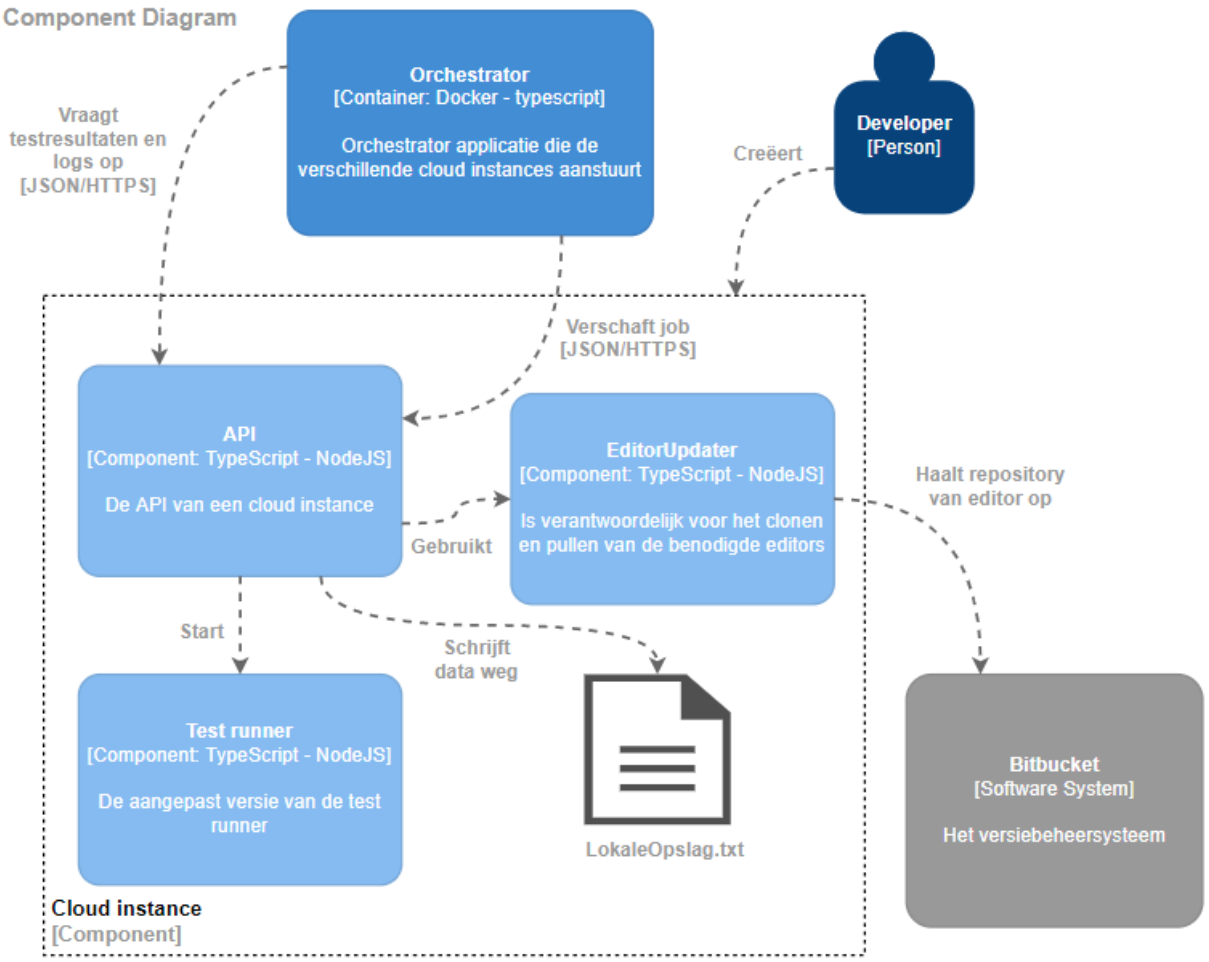
Het testproces start zodra de orchestrator een job verschaft aan een van de cloud instances. Deze stuurt de orchestrator via HTTPS als POST request naar de cloud instance. Vervolgens checkt de orchestrator, door middel van een GET request, of de cloud instance de performance test al uitgevoerd heeft. Als dat niet het geval is wordt er na één minuut een nieuwe GET request gestuurd. Wanneer de orchestrator na tien GET request geen testresultaten heeft ontvangen, wordt deze informatie gelogd in het Slack kanaal, zodat de developers kunnen onderzoeken of de cloud instance nog actief is. De keuze om na tien GET request een foutmelding te loggen is gebaseerd op de uitvoeringstijd van de performance testen. De performance testen in de cloud omgeving duren niet langer dan zes minuten, waardoor er na tien GET request een resultaat verwacht is.

De klasse genaamd "*DataStoring*" is verantwoordelijk voor het wegschrijven van de data. Zo worden de testresultaten door middel van een query opgeslagen in de PostgreSQL database en worden de verkregen logs opgeslagen in de Elk Stack. Daarnaast wordt de

status van de cloud instances verzonden naar een Slack kanaal. Naast de "*DataStoring*" klasse is de klasse "*TestLogic*" opgesteld. Deze klasse is verantwoordelijk voor het samenstellen van de POST en GET requests naar de cloud instances, waarbij de testconfiguratie wordt toegevoegd aan de request parameters. Bij het opstellen van de testconfiguratie maakt deze klasse gebruik van het jobConfig.json bestand, waarin onder andere staat aangegeven welke testen op welke cloud instance uitgevoerd moeten worden.

Cloud instance component diagram

Het component diagram van de cloud instance is hieronder weergegeven, waarbij de toegevoegde klassen zijn toegelicht.



Figuur 24: Cloud instance component diagram

De cloud instances hebben een port openstaan waarnaar de orchestrator POST requests kan sturen. Zodra de API een POST request ontvangt van de orchestrator start de API het testproces. Allereerst wordt de benodigde editor opgehaald door middel van de "*EditorUpdater*" klasse. Deze cloned de desbetreffende editor van de bitbucket omgeving. Wanneer de editor al lokaal beschikbaar is worden de eventuele veranderingen in de repository binnengehaald. Vervolgens wordt een aangepaste versie van de test runner

gestart. De test runner is zo aangepast dat deze vanaf buitenaf aangeroepen kan worden waarbij de testconfiguratie als parameter mee kan worden gegeven. Na het uitvoeren van de testrun worden de testresultaten geretourneerd aan de API, die deze vervolgens wegschrijft in een tekstbestand. Wanneer de orchestrator door middel van een GET request de resultaten opgevraagd gaat de API na of de opgevraagde testresultaten al weggeschreven zijn in het tekstbestand. Wanneer deze testresultaten niet beschikbaar zijn, is de performance test nog in uitvoering.

14.0 Implementatie orchestrator

Op basis van de opgestelde software architectuur uit hoofdstuk 13 wordt een prototype opgesteld. Hieronder zijn verschillende componenten van het prototype toegelicht.

14.1 Afbakening prototype

Door een gebrek aan tijd is in overleg met de stakeholders de scope van het prototype afgebakend. Hierbij zijn de belangrijkste componenten en functionaliteiten geselecteerd die gerealiseerd gaan worden in het prototype. In de tabel hieronder zijn de componenten en functionaliteiten weergegeven die in overleg met de stakeholders zijn geselecteerd.

Component	Functionaliteit
Orchestrator	Het formuleren van een job.
	Het uitsturen van een job in de vorm van een POST request.
	De cloud instances pollen door middel van een GET request.
	De testresultaten opslaan in een PostgreSQL database.
Cloud instance	Het ontvangen van een job.
	Het clonen van de benodigde editor.
	Aan de hand van de ontvangen job een testconfiguratie opstellen.
	Aan de hand van de testconfiguratie een performance test in de test runner starten.
	De performance testresultaten afkomstig van de test runner programmatuur beschikbaar stellen voor de orchestrator.
	De logs afkomstig van de test runner programmatuur beschikbaar stellen voor de orchestrator.

Tabel 10: Cloud instance component diagram

De bovengenoemde requirements zijn opgenomen in het prototype. Hieronder worden meerdere van de ontwikkelde componenten uit het prototype toegelicht met behulp van

code snippets. Daarnaast is de code lokaal ontwikkeld. Door de code lokaal te ontwikkelen hoeft er geen tijd besteed te worden aan het inrichten van de cloud instances. waardoor er sneller tot een eindresultaat kan worden gekomen. Tevens vindt de communicatie plaats via HTTPS, waardoor de applicatie lokaal en op een cloud instance ten uitvoer kan worden gebracht.

14.2 Orchestrator

Enkele functionaliteiten uit tabel 10 die betrekking hebben op de orchestrator zijn hieronder toegelicht. De orchestrator is ontwikkeld in Typescript en NodeJs, mede doordat de bestaande test runner hiermee ontwikkeld is. Hierdoor zijn componenten die afkomstig zijn van de test runner eenvoudig te hergebruiken. Daarnaast beschikt NodeJs over bibliotheek genaamd NodeFetch, die communicatie via HTTPS ondersteund.[50] Door het gebruik van deze bibliotheek kan communicatie tussen de orchestrator en cloud instances gerealiseerd worden.

Het starten van testrun

```
/**
 * Sends out the job and handles the response (main function)
 * @param PORT: the port on which the cloud instance is active
 * @param CI: the cloud instance configuration number
 * @param URL: the URL on which the cloud instance is active
*/
 async function startPerformanceTest(PORT: number, CI: number,
URL:
 string) {
    var testSetup = new Test(CI, PORT, URL);
    testSetup.sendTest(testSetup.getTestParams());
    var result = await testSetup.pollCloudInstance();
    testSetup.handleResults(result);
}
startPerformanceTest(8888, 1, 'http://localhost:');
```

Code snippet 5: Het starten van een testrun vanaf de orchestrator

De orchestrator kan een performance test starten op een van de cloud instances door middel van de bovenstaande functie. Hierbij zijn drie parameters vereist, namelijk de poort waarop de cloud instance actief is, de benodigde testconfiguratie (jobConfig.json) en de URL waarop de cloud instance te bereiken is. Allereerst wordt er een Test object aangemaakt. Elke performance test die uitgevoerd wordt beschikt over een Test object, waarin de benodigde configuratie present is. Bij het aanmaken van een Test object wordt een een globally unique identifier (GUID) toegewezen aan het object.[53] De GUID maakt de uitgevoerde performance test runs onderscheidbaar van elkaar en wordt later gebruikt

tijdens het ophalen van de resultaten. Na het opstellen van het Test object wordt de performance test configuratie verstuurd naar een cloud instance door middel van een POST request. Wanneer de cloud instance (in het prototype een proces dat lokaal actief is) de POST request ontvangt wordt de desbetreffende performance test uitgevoerd, dit proces is verder toegelicht in hoofdstuk 14.3. Na het versturen van de testconfiguratie, vraagt de orchestrator om de minuut de testresultaten op door middel van de pollCloudInstance functie. Binnen deze functie wordt het meest recente testresultaat van de cloud instance opgehaald, waarna deze wordt geverifieerd via de GUID. Hierbij wordt de GUID uit het ontvangen testresultaat vergeleken met de GUID binnen het Test object. Wanneer de resultaten ontvangen zijn, worden deze afgehandeld in de "*handleResults*" functie. Binnen deze functie worden de resultaten weggeschreven naar de postgreSQL database.

Het versturen van een job

In de code snippet hieronder is de code weergeven die verantwoordelijk is voor het versturen van een job naar een van de cloud instances. Hierbij worden de opties, bestaande uit de headers, body etc., gedefinieerd in de variabele genaamd options. De POST request wordt verstuurd naar de desbetreffende cloud instance door middel van NodeFetch, een NodeJS bibliotheek.[50] De NodeFetch bibliotheek is geselecteerd omdat deze eenvoudig in gebruik is en daarnaast het verzenden van POST en GET requests ondersteund.

```
/**
 * Sends a job via POST request to a cloud instance
*/
 public async sendTest() {
     var options =
this.createOptionsPostMessage(this.testParams);
     await nodeFetch(`${this.URL}${this.PORT}/job`, options)
            .then(response => {
                console.log("Job started on:", response.url);
            }).catch(err => console.log(err));
 }
```

Code snippet 6: Het versturen van een job

Het pollen van een cloud instance

Na het verzenden van de testconfiguratie door middel van een POST request, haalt de orchestrator de testresultaten op van de cloud instances door middel van een GET request. De testresultaten worden opgehaald door het gebruik van een GET request, omdat er hierdoor geen langdurige verbinding open hoeven te staan. De testresultaten kunnen

tevens dienen als response in de POST request, maar dit zorgt voor verbindingen die enkele minuten open kunnen staan.

Het proces van het opvragen van de testresultaten is weergeven in onderstaande code snippet. Hierbij wordt er een interval opgezet, zodat het opvragen van de resultaten om de opgegeven tijd kan plaatsvinden. Tijdens een interval worden de meest recente testresultaten opgevraagd aan de cloud instance door middel van een GET request. Deze actie vindt plaats binnen de *"retrieveTestResults"* functie Binnen deze functie wordt de GUID van het opgehaalde testresultaat vergeleken met de GUID opgeslagen binnen het Test object. Wanneer deze overeenkomen worden de testresultaten geretourneerd.

```
 * Polls a specific cloud instance for the results
 * @param testSetup: the testSetup object
  *  @resolves  either  returns  the  matching  test  results  or  the
message
            saying no matching GUID is found.
*/
public async pollCloudInstance(testSetup: Test) {
     return await new Promise(resolve => {
       var c = 0;
       var timeout = setInterval(async function () {
         var isCorrectGUID = await testSetup.retrieveTestResults
           (this.getGUID(), testSetup);
           c++;
          if (isCorrectGUID == true) {
           clearInterval(timeout);
           var result = await testSetup.getResults(
           `${this.URL}${this.PORT}/results`);
           resolve(result);
          } else if (c > 10) {
           clearInterval(timeout);
           resolve("No matching GUID found");
         }
       }, 60000);
     });
};
```

Code snippet 7: Poll functionaliteit

14.3 Cloud instance

Enkele functionaliteiten uit tabel 10 die betrekking hebben op de cloud instance zijn hieronder toegelicht door middel van code snippets. Om dezelfde redenen als beschreven bij de orchestrator is de logica van de cloud instances ontwikkeld met Typescript en NodeJs.

Ontvangen en verwerken van een job

Een cloud instance ontvangt, door middel van een POST request, de benodigde testconfiguratie. Aan de hand van deze informatie haalt de cloud instance de benodigde editor op uit de repository. Door het ophalen van de benodigde editor worden de performance testen uitgevoerd op de meest recente versie van de editor, waardoor veranderingen in de editor direct getest worden. Na het ophalen van de benodigde editor wordt de performance test gestart aan de hand van de testconfiguratie verkregen uit de POST request, waarna de testresultaten opgeslagen worden in een lokaal tekstbestand. Door de testresultaten weg te schrijven naar een lokaal tekstbestand kunnen deze later opgehaald worden door de orchestrator.

```
/**
 * Cloud instance accepts a job
*/
app.post('/job', async (req: any, res: any) => {
  //Get the parameters from the POST body
  const {URL,branch,test,path,editor,browser,environment
        ,documentIds,GUID} = req.body;

  //Clone or pull the repository for the needed editor
  initializeEditorFolder(URL, editor);

  //Create the jobConfig.json content
  var jobConfig = createjobConfig(branch,test,editor,browser,
                  environment, documentIds, path);

  //Start the performance test
  const t = new PerformanceDashboard();
  var testResults = await t.main(jobConfig);
  //Start the performance test
  writeToFile(GUID, testResults);
});
```

Code snippet 8: Ontvangen en verwerken van een POST request

Afhandelen van een GET request

De orchestrator haalt de performance testresultaten op van de cloud instance door middel van een GET request. Wanneer een cloud instance een GET request ontvangt, wordt de onderstaande code uitgevoerd. Hierbij wordt er een *"ReadText"* object aangemaakt. Dit object bevat de functionaliteiten om te itereren door een tekst bestand en daarbij de laatste regel in het tekstbestand op te halen. De laatste regel in het tekstbestand wordt als response geretourneerd aan de orchestrator.

```
/**
 * Cloud instance handles a GET request to /results
*/
app.get('/results', async function (req: any, res: any) {
  var resultsReader = new ReadText();
  var latestResult = await resultsReader.readFile();
  res.send(latestResult);
});
```

Code snippet 9: Afhandelen van een GET request

15.0 Adviesrapport

Na afloop van het onderzoek, de software architectuur en de implementatie van het prototype is er een adviesrapport opgesteld. Fonto kan aan de hand van onderstaande adviezen de ontwikkeling van de test runner voortzetten.

15.1 Advies

Uit het onderzoek blijkt dat van de geselecteerde cloud instances de Microsoft Azure F2 instance de laagste variabiliteit biedt. Alhoewel de F2 instance minder stabiel blijkt dan de on-premise omgeving, biedt deze genoeg stabiliteit om de gewenste regressies waar te nemen (zie hoofdstuk 12). Daarnaast wenste Fonto de implementatie van een orchestrator oplossing. Uit de opgestelde software architectuur en het opgestelde prototype blijkt dat dit een valide oplossing is, die voldoet aan de opgestelde requirements van Fonto (zie hoofdstuk 13 en 14).

Fonto wordt geadviseerd om de huidige test runner te migreren naar de cloud. Hierbij wordt het gebruik van de F2 instance, die beschikbaar is op de cloud provider Microsoft Azure, aanbevolen. Daarbij wordt het gebruik van een orchestrator oplossing, zoals weergeven in hoofdstuk 13 en 14 aanbevolen.

15.2 Vervolgstappen

Om de ontwikkeling van de test runner voort te zetten, zijn er meerdere vervolgstappen opgesteld. Deze zijn hieronder toegelicht.

Docker image opstellen voor de cloud instances

Momenteel moeten de cloud instances handmatig opgezet worden, waarbij de benodigde programma's zoals NodeJS en Chrome handmatig geïnstalleerd moeten worden. Om dit proces eenvoudiger en tijdsefficiënt te maken kan Fonto een Docker image opstellen van de cloud instances. Hierdoor hoeft er alleen een cloud instance geinstantieerd worden, waarop de Docker image als container uitgevoerd kan worden.

Docker image maken voor de orchestrator

Door het omzetten van de orchestrator naar een Docker image kan deze stateless ten uitvoer gebracht worden in de cloud. Daarnaast biedt een docker container een hogere mate van onderhoudbaarheid, omdat deze eenvoudig herinzetbaar is.

Onderzoek naar ELK Stack als vervanging voor Grafana

Uit gesprekken met Jos Verburg, de ontwikkelaar van de test runner, is voortgekomen dat Fonto zoekt naar een vervanging van Grafana, omdat deze een trage responstijd heeft. Binnen de software architectuur is de ELK Stack ingezet als opslag voor de logs. Naast deze functionaliteit biedt de ELK Stack een soortgelijk dashboard als Grafana, die geutiliseerd kan worden voor het weergeven van de performance testresultaten. Fonto kan onderzoek doen naar de ELK Stack, als eventuele vervanging voor Grafana.

Performance van een Docker container testen als vervanging van F2 instance

Om het toevoegen van nieuwe cloud instances te vereenvoudigen kunnen de F2 instance omgezet worden naar een Docker image. Hiervoor moet Fonto onderzoek uitvoeren naar de stabiliteit van een Docker container. Tijdens dit onderzoek kan Fonto de gevonden metingen vergelijken in het opgestelde Python script.

Onderzoek naar effectieve Slack logging

Momenteel wordt er, na elke uitvoering van de test runner, een bericht gestuurd naar het Slack kanaal. Dit bericht beschikt over de status van de test runner en eventuele foutmeldingen. Wanneer de test runner wordt geïmplementeerd in een cloud omgeving moet de berichtgeving naar het Slack kanaal aangepast worden, omdat de duratie van een test run significant korter is. Daarnaast moet er in de Slack logging onderscheid gemaakt worden tussen de verschillende cloud instances. Fonto kan onderzoeken welke berichtinformatie en welke frequentie van berichtgeving gewenst is.

Onderzoek naar bestaande orchestrator oplossingen

Fonto kan, naast het doorontwikkelen van het prototype, onderzoek doen naar bestaande orchestrator oplossingen. Onderzoek naar bestaande softwarepakketten valt buiten de scope van het afstudeerproject, maar kan mogelijk een alternatief bieden voor Fonto.

Het build proces van de test runner efficiënter maken

In het prototype is het testproces, weergeven in figuur 6, hergebruikt. Zoals weergeven in de software architectuur bevat een test run één test, waarbij voor iedere uitvoering de editor gebouwd moet worden. Omdat de editor voor elke test gebouwd moet worden, gaat er een significante hoeveelheid tijd verloren. Fonto kan het testproces, zoals weergeven in figuur 6, efficiënter maken door een gebouwde editor te hergebruiken voor meerdere test runs.

Testen van het prototype

Tijdens de afstudeerperiode is er geen tijd besteed aan het testen van het prototype. De statistische kant van het onderzoek bedroeg meer tijd dan initieel verwacht, waardoor het testen van het prototype achterwege is gelaten. Deze keuze is gemaakt in overleg met de stakeholders bij Fonto, die daarbij aangaven dat het prototype meer waarde biedt dan het testrapport.

15.3 Opgeleverde producten

Na afloop van de afstudeerperiode worden meerdere producten opgeleverd aan Fonto. Deze producten kan Fonto benutten tijdens het vervolgonderzoek of tijdens de ontwikkeling van de nieuwe test runner. De opgeleverde producten zijn hieronder toegelicht.

Python script

Tijdens het onderzoek is een Python script opgesteld die de statistische berekeningen uitvoert, zie hoofdstuk 11.4. Wanneer Fonto vervolgonderzoek uitvoert op andere cloud instances of docker containers kan het geschreven Python script geutiliseerd worden. Hierdoor is Fonto in staat om in korte tijdsperiode de variabiliteit van onder andere meerdere cloud instances of docker containers vast te stellen, zonder zich hierbij hoeven te verdiepen in de statistische methodieken.

Software architectuur

De opgestelde software architectuur, beschreven in hoofdstuk 13, kan Fonto benutten tijdens de realisatie van de nieuwe test runner. Hierbij kan Fonto componenten uit de software architectuur hergebruiken of kan Fonto de opgestelde software architectuur uitbreiden.

Prototype

Het prototype, beschreven in hoofdstuk 14, biedt Fonto een basis voor de nieuwe test runner. Hierbij kan Fonto het gehele prototype utiliseren of verschillende componenten hergebruiken, zoals de communicatie tussen de orchestrator en cloud instances of de aangepaste test runner componenten.

Onderzoek

Het uitgevoerde onderzoek kan door Fonto onder andere benut worden tijdens de ontwikkeling van de test runner of tijdens vervolgonderzoek. Hierbij kan Fonto terug refereren naar het onderzoek en kunnen onderdelen uit het onderzoek hergebruikt worden, zoals de de aanpak of statistische methodieken.

16.0 Reflectie

Hoofdstuk 16 bevat een reflectie op de afstudeerperiode, waarin ik beschrijf welke onderdelen goed zijn gegaan, wat mijn verbeterpunten zijn en mijn algemene visie op de afstudeerperiode.

16.1 Wat ging goed

Het uitvoeren van onderzoek

Tijdens de afstudeerperiode is het merendeel van de tijd besteed aan onderzoek, zowel deskresearch als het uitvoeren van een eigen onderzoek. De deskresearch heeft de basis gevormd voor het uitgevoerde onderzoek, waarbij de geselecteerde cloud instances met elkaar vergeleken zijn.

Communicatie met de stakeholders

Ondanks de corona situatie is er tijdens de afstudeerperiode constant contact gehouden met de begeleiders. Wanneer ik tegen een probleem aanliep werd dit via Slack besproken met de begeleiders, om zo tot een oplossing te komen. Daarnaast heb ik regelmatig belangrijke keuzes en problemen besproken met andere stakeholders, zoals Jos Verburg. Hierdoor konden problemen snel opgelost worden en kon ik na een korte tijd de weg binnen Fonto vinden.

Systematische aanpak

Tijdens de afstudeerperiode is herhaaldelijk gebruik gemaakt van een systematische aanpak. Bijvoorbeeld tijdens de realisatie van de software architectuur door middel van C4, tijdens het testen van de cloud instances (hoofdstuk 10) of tijdens het uitvoeren van de statistische methodes (hoofdstuk 11). Door het gebruik van een systematische aanpak worden de processen efficiënter en zorgvuldiger doorlopen en is daarom iets wat meegenomen wordt voor toekomstige projecten.

16.2 Wat kan ik verbeteren

Een uitgebreide risicoanalyse opstellen

Tijdens de periode waarin de cloud instances geinstantieerd en getest werden, deed er zich een probleem voor die niet opgenomen was in de risicoanalyse. Dit probleem ontstond doordat een benodigd commando niet opgenomen was in het opstartscript (hoofdstuk 10). Hierdoor werden specifieke achtergrond processen niet afgesloten, wat leidde tot onbetrouwbare testresultaten. Het constateren en oplossen van het probleem bedroeg twee dagen. In de toekomst kan de duur van zulke problemen significant ingekort worden, door vooraf de mogelijke risico's en oplossingen in kaart te brengen.

16.3 Algemene visie

Terugkijkende naar de afstudeerperiode en de opgeleverde producten ben ik zeer tevreden. De afstudeeropdracht bevatte veel uitdagingen en nieuwe onderwerpen die buiten de leerstof van de opleiding vallen. Om deze reden heb ik mij in meerdere onderwerpen extra moeten verdiepen, waardoor ik een aanzienlijke hoeveelheid kennis heb opgedaan.

Daarnaast zijn de stakeholders bij Fonto zeer tevreden, omdat ik tijdens mijn afstudeerperiode veel complexe vraagstukken heb opgelost die nieuwe mogelijkheden bieden voor de test runner. Verder is Fonto, door middel van de opgeleverde producten, in staat de ontwikkeling van de test runner voort te zetten.

17.0 Toelichting beroepstaken

Binnen hoofdstuk 17 zijn de geselecteerde HBO-ICT competenties bewezen.

A1 Analyseren probleemdomein

- Het analyseren en opstellen van het probleemdomein (hoofdstuk 2.4), aanleiding (hoofdstuk 4.1) en probleemstelling (hoofdstuk 4.2).
- Het opstellen van een bedrijfsanalyse en organogram (hoofdstuk 2).
- Het analyseren en beschrijven van de context (hoofdstuk 3).
- De werking van de test runner in kaart brengen (hoofdstuk 3).
- Het opstellen van een risicoanalyse (bijlage A).

B1 Gemotiveerd selecteren van ICT-gerelateerde oplossingen

- Deskresearch naar de cloud (hoofdstuk 7 en 8).
- Het opgesteld adviesrapport, die onder andere beschikt over een advies en vervolgstappen (hoofdstuk 15).

C1 Ontwerpen software

- De software architectuur opgesteld door middel de ontwerpmethodiek C4 (hoofdstuk 13).
- Het opstellen van een orchestrator oplossing (hoofdstuk 13).
- Het ontwerpen van stateless componenten in de software architectuur (hoofdstuk 13).

D1 Realiseren van software

- Het realiseren van het Python script (hoofdstuk 11.4).
- Het realiseren van een prototype (hoofdstuk 14).
- Het analyseren, aanpassen en hergebruiken van componenten uit de test runner (hoofdstuk 13 en 14).

D2 Testen & evalueren

- Het analyseren en vergelijken van de geselecteerde cloud instances (hoofdstuk 9, 10, 11 en 12).
- Het uitvoeren van performance testen op de cloud instances (hoofdstuk 10).

Gc Kritisch, onderzoekend en methodisch werken

- De uitgevoerde deskresearch omtrent de cloud (hoofdstuk 7 en 8).
- Een agile werkwijze, met componenten van Scrum en waterval (hoofdstuk 6).
- Communicatie met de stakeholders en de ontwikkelaar van de test runner omtrent de voortgang en gemaakte keuzes (hoofdstuk 9, 10, 11, 12, 13 en 14).
- Procesgericht te werk gaan (Bijlage F en G).

Gf Leren leren

- Het analyseren, selecteren en toepassen van statistische methoden (hoofdstuk 11 en 12).
- Het gebruik van PostgreSQL (hoofdstuk 9 en 10), Typescript en NodeJS (hoofdstuk 13 en 14) en het gebruik van Docker (hoofdstuk 13 en 14).
- Het analyseren, aanpassen en hergebruiken van componenten uit de test runner (hoofdstuk 13 en 14).
- Het selecteren, analyseren en beschrijven van bestaande onderzoeken omtrent performance testing in de cloud (hoofdstuk 8).

Literatuurlijst

1. What is the Cloud - Definition. (z.d.). Microsoft Azure. https://azure.microsoft.com/en-us/overview/what-is-the-cloud

2. Verschil tussen openbare cloud, privécloud en hybride cloud. (z.d.). Microsoft Azure. https://azure.microsoft.com/nl-nl/overview/what-are-private-public-hybrid-clouds

3. Vennam, S. (2021, 17 mei). Hybrid Cloud. IBM. https://www.ibm.com/cloud/learn/hybrid-cloud

4. Laaber, C., Scheuner, J., & Leitner, P. (2018). Performance testing in the cloud. How bad is it really? PeerJ Prepr. Published. https://doi.org/10.7287/peerj.preprints.3507v1

5. NexGenT. (2019, 2 augustus). What is Virtualization? [Video]. YouTube. https://www.youtube.com/watch?app=desktop&v=L8A9PHeyRrY&ab_channel=NexGenT

6. Virtualization. (z.d.). Dynatrace. https://www.dynatrace.com/resources/ebooks/javabook/why-virtualization-has-impact-on-performance-management

7. Amazon EC2 Instance Types - Amazon Web Services. (z.d.). Amazon Web Services, Inc. https://aws.amazon.com/ec2/instance-types

8. Instance families. (z.d.). [Afbeelding]. Devopsschool. https://www.devopsschool.com/slides/aws/aws-certified-solutions-architect-associate/images/ec2-instance-family-for-aws.jpg

9. Leitner, P., & Cito, J. (2016). Patterns in the Chaos. ACM Transactions on Internet Technology, 16(3), 1–23. https://doi.org/10.1145/2885497

10. The 'noisy neighbor' effect. (2021, 28 januari). IONOS Digitalguide. https://www.ionos.com/digitalguide/hosting/technical-matters/noisy-neighbor-effect-explanation-and-solutions/

11. Vellante, D. (2021, 13 februari). Big 4 cloud providers. SiliconANGLE. https://siliconangle.com/2021/02/07/big-4-cloud-providers-revenue-poised-surpass-115b-year

12. Virtual Machines (VMs) for Linux and Windows. (z.d.). Microsoft Azure. https://azure.microsoft.com/en-us/services/virtual-machines/

13. Amazon EC2. (z.d.). Amazon Web Services, Inc. https://aws.amazon.com/ec2

14. VM instances pricing | Compute Engine Documentation |. (z.d.). Google Cloud. https://cloud.google.com/compute/vm-instance-pricing

15. Kolosovskyi, I. (2021, 17 mei). Is JavaScript Single-Threaded? Simple Talk. https://www.red-gate.com/simple-talk/development/dotnet-development/javascript-single-threaded

16. Pricing – Linux Virtual Machines. (z.d.). Microsoft Azure. https://azure.microsoft.com/en-ca/pricing/details/virtual-machines/linux/

17. All pricing | Compute Engine Documentation |. (z.d.). Google Cloud. https://cloud.google.com/compute/all-pricing#premiumimages

18. S. (2020, 4 december). Worry about Computer Heat? You Should Know These Things. MiniTool. https://www.minitool.com/data-recovery/computer-heat.html

19. Microsoft. (z.d.). What's the difference between the new Microsoft Edge and Microsoft Edge Legacy? https://support.microsoft.com/en-us/microsoft-edge/what-s-the-difference-between-the-new-microsoft-edge-and-microsoft-edge-legacy-a01258e5-8c05-7bbb-bed2-c65bec0eb126

20. Russinovich, M. (2021, 26 mei). Process Explorer - Windows Sysinternals. Microsoft Docs. https://docs.microsoft.com/en-us/sysinternals/downloads/process-explorer

21. Fonto. (z.d.). Fonto. https://www.fontoxml.com

22. Wikipedia contributors. (z.d.-g). Wilcoxon signed-rank test. Wikipedia. https://en.wikipedia.org/wiki/Wilcoxon_signed-rank_test

23. Taylor, C. (2021, 9 mei). What Is a Population in Statistics? ThoughtCo. https://www.thoughtco.com/what-is-a-population-in-statistics-3126308

24. Momoh, O. (2021, 18 maart). Understanding Population Statistics. Investopedia. https://www.investopedia.com/terms/p/population.asp

25. Kenton, W. (2021, 4 januari). Sample. Investopedia. https://www.investopedia.com/terms/s/sample.asp

26. Taylor, C. (2019, 17 februari). What Is Statistical Sampling? ThoughtCo. https://www.thoughtco.com/what-is-statistical-sampling-3126366

27. Wikipedia contributors. (z.d.-d). Normalization (statistics). Wikipedia. https://en.wikipedia.org/wiki/Normalization_(statistics)

28. Hayes, A. (2020, 2 april). Using the Variance Equation. Investopedia. https://www.investopedia.com/terms/v/variance.asp

29. Bhandari, P. (2020, 12 oktober). Understanding and calculating variance. Scribbr. https://www.scribbr.com/statistics/variance

30. Wikipedia contributors. (z.d.-d). Q–Q plot. Wikipedia. https://en.wikipedia.org/wiki/Q-Q_plot

31. Mcleod, S. (2019, 10 juni). Z-Score: What are Confidence Intervals in Statistics? Simplypsychology. https://www.simplypsychology.org/confidence-interval.html

32. Wikipedia contributors. (z.d.-b). Confidence interval. Wikipedia. https://en.wikipedia.org/wiki/Confidence_interval

33. Wikipedia contributors. (z.d.-b). Levene's test. Wikipedia.
https://en.wikipedia.org/wiki/Levene%27s_test

34. GeeksforGeeks. (2020, 2 juli). Normalization vs Standardization.
https://www.geeksforgeeks.org/normalization-vs-standardization

35. Wikipedia contributors. (z.d.-a). Brown–Forsythe test. Wikipedia.
https://en.wikipedia.org/wiki/Brown-Forsythe_test

36. NumPy. (z.d.). Numpy.
https://numpy.org

37. SciPy. (z.d.). SciPy.
https://www.scipy.org

38. Matplotlib - PyLab module - Tutorialspoint. (z.d.). Tutorialspoint.
https://www.tutorialspoint.com/matplotlib/matplotlib_pylab_module.html

39. Wikipedia contributors. (z.d.-b). Winsorizing. Wikipedia.
https://en.wikipedia.org/wiki/Winsorizing

40. Wicklin, R. (2017, 8 februari). Winsorization: The good, the bad, and the ugly. The DO Loop.
https://blogs.sas.com/content/iml/2017/02/08/winsorization-good-bad-and-ugly.html

41. Z. (2021, 22 januari). How to Winsorize Data: Definition & Examples. Statology.
https://www.statology.org/winsorize

42. Rando, N. (2014, 2 december). Noisy neighbor. SearchCloudComputing.
https://searchcloudcomputing.techtarget.com/definition/noisy-neighbor-cloud-computing-performance

43. Wang, G., & Eugene Ng, T. S. (2010). The Impact of Virtualization on Network Performance of Amazon EC2 Data Center. Dept. of Computer Science, Rice University. Published. https://doi.org/10.1109/INFCOM.2010.5461931

44. Wikipedia contributors. (z.d.). Software development kit. Wikipedia.
https://en.wikipedia.org/wiki/Software_development_kit

45. Brown, S. (z.d.). The C4 model. C4 model.
https://c4model.com

46. UML. (z.d.). UML.
https://www.uml.org

47. Elastic Stack. (z.d.). Elastic Stack.
https://www.elastic.co/elastic-stack

48. PostgreSQL. (z.d.). node-postgres.
https://node-postgres.com

49. HTTPS | Node.js v16.2.0 Documentation. (z.d.). Node.Js.
https://nodejs.org/api/https.html

50. Node-fetch. (2020, 5 september). Npm.
https://www.npmjs.com/package/node-fetch

51. How to create a GUID / UUID. (2008, 19 september). Stack Overflow.
https://stackoverflow.com/questions/105034/how-to-create-a-guid-uuid

52. What is Cloud Computing. (z.d.). Amazon Web Services, Inc.
https://aws.amazon.com/what-is-cloud-computing/

53. Kolosovskyi, I. (2021b, mei 17). Is JavaScript Single-Threaded? Simple Talk.
https://www.red-gate.com/simple-talk/development/dotnet-development/javascript-single-threaded/

Bijlage A: Plan van Aanpak

Plan van Aanpak

Uitvoeren van een onderzoek en het realiseren van een prototype omtrent de test runner bij Fonto.

Student
Naam: Dijck Dessens
Email: 17091934@student.hhs.nl
Opleiding: HBO-ICT
Jaar: 2020/2021
Onderwijsinstelling: De Haagse Hogeschool

Bedrijfsinformatie
Naam: Dhr. Bert Willems
Email: bert.willems@fontoxml.com
Bedrijf: Fonto Group B.V.
Locatie: Rijswijk

Begeleidend examinator
Naam: Rob van der Krog
Email: r.p.f.vanderkrogt@hhs.nl

Inhoudsopgave

1.0 Inleiding

Ten behoeve van de afstudeeropdracht van Dijck Dessens is er een plan van aanpak opgesteld. Het project wordt uitgevoerd voor het bedrijf Fonto en beoordeeld door de Haagse Hogeschool. Tijdens de afstudeeropdracht wordt er onderzoek gedaan naar de huidige testomgeving bij Fonto met als doel deze op een schaalbare manier in te richten.

In het plan van aanpak wordt duidelijk gemaakt waarom de afstudeeropdracht wordt uitgevoerd en waarom deze van waarde is voor Fonto. Daarnaast wordt er toegelicht hoe de afstudeeropdracht uitgevoerd gaat worden. Het doel van het plan van aanpak is om een concrete basis voor het project op te zetten, die op een ondersteunende manier werkt voor de student, de begeleiders en Fonto.

2.0 Organisatieomschrijving

Om een beter beeld te krijgen van het bedrijf waar het project wordt uitgevoerd, worden in dit hoofdstuk de organisatie, missie, kernactiviteiten en probleemdomein behandeld.

2.1 Organisatie

Liones is een bedrijf die zich focust op intelligente content creatie. Zo bieden ze consultancy aan op het gebied van *XML authoring* en geven ze training omtrent Fonto. Het bedrijf is gevestigd in Rijswijk en beschikt momenteel over circa 50 medewerkers. Liones is opgericht in 1999 en beschikt onder andere over back-end ontwikkelaars, front-end ontwikkelaars, UX designers, marketing experts en interaction designers.

Fonto is een product dat in 2013 ontwikkeld is door Liones. Dit product is een client-side editor die het structureren van content eenvoudig en toegankelijk maakt voor gebruikers die niet beschikken over kennis van XML. Fonto richt zich tot meerdere branches van de samenleving. Zo verstrekt Fonto hun applicatie aan onder andere de zorg, luchtvaartmaatschappijen en verschillende wetgevende organisaties. Enkele voorbeelden hiervan zijn: Smith+Nephew, SunExpress en Stabilimenti Tipografici Carlo Colombo.

Een belangrijk aspect van Fonto is de hechte bedrijfscultuur. Deze cultuur wordt gevormd door medewerkers die bereid zijn elk ander te helpen en elkaar als familie zien. Daarnaast is Fonto gecertificeerd door het bedrijf Great Place To Work. Dit bedrijf heeft Fonto beoordeeld op basis van enquêtes onder de medewerkers, waarbij de enquêtes gericht waren op: prestatie, geloofwaardigheid, respect, eerlijkheid, trots en kameraadschap. Dit resulteerde in een gemiddelde score van 100%.

2.2 Missie

Missie

Het structureren van content eenvoudig en toegankelijk maken voor iedereen.

2.3 Kernactiviteiten

De activiteiten van Fonto hebben betrekking tot hun applicatie, genaamd Fonto. Zo werken ze aan nieuwe functies en testen ze de huidige en verouderde versie. Daarnaast ontwikkelen ze nieuwe tools van de Fonto applicatie. Hierbij wordt er geduid op onder andere review en geschiedenis tools. Fonto houdt zich tevens ook bezig met de marketing en sales omtrent hun product.

Fonto beschikt over verschillende teams. De teams die betrekking hebben op mijn opdracht zijn het performance- en SDK team. Het performance team richt zich op de kwaliteit van de applicatie. Hierbij voeren ze onder andere prestatietesten uit om de prestatie van verschillende versies te bewaken. Het SDK team houdt zich vooral bezig met het toevoegen van nieuwe functionaliteit en het optimaliseren van de code.

2.4 Probleemdomein

Fonto is een client-side applicatie die een gebruiksvriendelijke manier biedt voor het structureren van content. Gebruikers kunnen met behulp van de editor, genaamd Fonto, onder andere metadata toevoegen, specifieke structuren opslaan om later te hergebruiken, taal en grammatica met behulp van Fonto Content Quality controleren en verandering bijhouden in het document.

Deze applicatie wordt geïntegreerd met een server-side CMS, zodat documenten, structuren etc. van de gebruiker opgeslagen worden. Binnen een bedrijf kunnen verschillende afdelingen gebruik maken van een eigen repository. Hierin kunnen ze vervolgens documenten opzoeken, de verandering inzien en structuren hergebruiken die betrekking hebben op hun eigen afdeling.

Om de kwaliteit van de Fonto applicatie te waarborgen, moet deze getest worden. Om de applicatie te testen maakt Fonto gebruik van een onder andere unit testen, acceptatietesten en integratie testen. Daarnaast voeren ze ook performance testen uit. Deze testen worden uitgevoerd door een programma genaamd test runner.

3.0 Opdrachtomschrijving

In hoofdstuk 3 wordt er ingegaan op de opdrachtomschrijving. Hierbij worden de aanleiding, probleemstelling, hoofdvraag en deelvragen behandeld.

3.1 Aanleiding

Het programma dat de performance testen uitvoert, genaamd de test runner, heeft zijn grenzen bereikt en kan niet meer worden uitgebreid. Momenteel kosten de performance testen vier uur om te voltooien. Fonto heeft vastgesteld dat er minimaal drie testen opeenvolgend uitgevoerd moeten, om regressie aan te tonen. Dit leidt ertoe dat Fonto binnen een halve dag kan aantonen of er regressies zijn opgetreden binnen de applicatie. Zodra de selectie van performance testen langer duurt, kan deze doelstelling niet meer behaald worden. Hierdoor is het dus niet mogelijk om nieuwe testen toe te voegen, omdat er anders teveel tijd zit tussen de regressies en de constatering daarvan. Daarnaast is het hierdoor niet mogelijk om in de test runner stresstesten uit te voeren, die de bovengrens bewaken van de Fonto applicatie.

Verder voldoet de huidige test runner niet aan de ISO 27001, wat tevens een doelstelling is binnen Fonto. De test runner beschikt niet over disaster recovery waardoor er bij hardware fouten, inbraken, natuurrampen etc. een hoog risico ontstaat, doordat er geen back-up beschikbaarbaar is. Daarbij kunnen deze incidenten zorgen voor een lange hersteltijd, wat resulteert in een periode waarin de applicatie van Fonto niet getest kan worden op het gebied van performance. Hiernaast is Fonto zelf verantwoordelijk voor het serverbeheer. Dit brengt hoge risico's met zich mee op het gebied van security.

3.2 Probleemstelling

De test runner, waar uitsluitend performance testen worden uitgevoerd, heeft zijn limiet bereikt en is daarnaast ook niet schaalbaar. De huidige test runner draait op één server en is niet in staat op te schalen. Hierdoor kan Fonto geen nieuwe performance testen toe voegen, kunnen ze geen stress testen uitvoeren en kunnen ze geen effectieve performance testen starten. Dit leidt ertoe dat Fonto de kwaliteit van hun applicatie niet kan waarborgen. Fonto heeft de cloud geselecteerd als mogelijke oplossing, name door de mogelijkheden die de cloud verschaft op het gebied van schaalbaarheid. Daarnaast is Fonto van plan de nieuwe test runner te ontwikkelen met behulp van een orchestrator architectuur.

3.3 Hoofdvraag

Hoe kan de test runner bij Fonto schaalbaar gemaakt worden?

3.4 Deelvragen

1. Bieden performance testresultaten, afkomstige van een cloud instance, dezelfde betrouwbaarheid als performance testresultaten afkomstig van de on-premise omgeving?

2. Hoe ontwerp je de orchestrator zo dat hij constant kan draaien in een cloud / CI omgeving, waarbij hij de verschillende workers continue blijft aansturen?
3. Is het mogelijk om de log gegevens te verplaatsen naar een centrale plek, die toegankelijk is via een web interface?
4. Hoe plot je de testdata in een grafiek en welke tools kan je daarvoor gebruiken?

4.0 Doelstelling en eindproducten

Om een helder beeld te krijgen van het doel, wordt deze behandeld in hoofdstuk 4. Hiernaast wordt dit verder toegelicht door middel van het gewenste resultaat.

4.1 Doelstelling

Na afloop van de opdracht is er onderzoek gedaan naar de mogelijkheden van een nieuwe test runner bij Fonto. Op basis van het uitgevoerde onderzoek is er een ontwerp opgesteld die de nieuwe test runner in kaart brengt. Aan de hand van dit ontwerp is er ook een prototype opgesteld.

4.2 Gewenste resultaat

Aan het eind de afstudeerperiode wordt er een prototype opgeleverd van de nieuwe test runner samen met het onderzoek, ontwerp en adviesrapport. Dit prototype vormt de basis voor de test runner en kan in de toekomst verder doorontwikkeld worden door Fonto. Het uitbreiden van het prototype kan op basis van het uitgevoerde onderzoek in combinatie met het adviesrapport. Met behulp van de opgeleverde producten is Fonto beter in staat om een hogere kwaliteit software te waarborgen, doordat ze onder andere meer performance testen kunnen uitvoeren, eenvoudig hun testomgeving kunnen opschalen en doordat ze beter inzicht hebben gekregen in de teststatistieken.

In het geval dat er niet voldoende tijd is om alle fases te doorlopen is de keuze gemaakt, in samenwerking met Fonto, om kwaliteit te kiezen boven kwantiteit. In andere woorden zullen de eerdere fases, in dit geval de analyse-en ontwerpfase, in meer detail worden uitgewerkt, waarbij er minder aandacht wordt besteed aan de realisatie- en testfase.

5.0 Theoretisch kader

In hoofdstuk 5 worden belangrijke begrippen, methodieken en modellen behandeld die betrekking hebben op het project.

5.1 Begrippen

Begrip: *performance testen*

Wikipedia (Wikipedia-bijdragers, 2021) beschrijft het begrip performance testen als 'in software quality assurance, performance testing is in general a testing practice performed to determine how a system performs in terms of responsiveness and stability under a particular workload. It can also serve to investigate, measure, validate or verify other quality attributes of the system, such as scalability, reliability and resource usage.'[3]

De test runner bij Fonto is uitsluitend gericht op performance testen. Deze testen worden uitgevoerd op verschillende browser om onder andere de prestatie van verschillende Fonto versies te monitoren om regressies te detecteren.

Begrip: *test runner*

De test runner duidt op een programma dat gebruikt wordt om de Fonto applicatie te testen. Dit programma voert onder andere performance testen uit op de applicatie waarbij de gegevens worden opgeslagen en vergeleken. Door behulp van dit programma is Fonto in staat om de prestatie van verschillenden Fonto versies te testen waardoor ze een hogere kwaliteit software kunnen waarborgen.

Begrip: *cloud computing*

Wikipedia (Wikipedia-bijdragers, 2020) beschrijft het begrip cloud computing als 'cloud computing is het via een netwerk – vaak het internet – op aanvraag beschikbaar stellen van hardware, software en gegevens, ongeveer zoals elektriciteit uit het lichtnet. De term is afkomstig uit de schematechnieken uit de informatica, waar een groot, decentraal netwerk (zoals het internet) met behulp van een wolk wordt aangeduid.'[1]

Tijdens het onderzoek wordt er uitgezocht wat de mogelijkheden zijn van de cloud in combinatie met de test runner. Hiervoor zal er onder andere moeten worden onderzocht wat hier de voor- en nadelen van zullen zijn.

Begrip: *schaalbaarheid*

Drs. Frank A.G. Hoes (Hoes, 2017) beschrijft de term schaalbaarheid als 'Schaalbaarheid (scalability: meegroeimogelijkheid) is een begrip uit de IT en de techniek. Het beschrijft de mogelijkheden van een systeem om mee te groeien in de toename van het gebruik dat systeem. Een niet-schaalbaar systeem is gebonden aan een maximale productiviteit door gebondenheid aan uren, capaciteit en ruimte.'[2]

Een van de grootste problemen van de huidige test runner is dat deze niet schaalbaar is. Hierdoor is Fonto niet in staat om hun testomgeving uit te breiden met nieuwe testen. Tijdens het onderzoek staat schaalbaarheid daarom ook centraal.

5.2 Methodieken en modellen

Model: *master en slave model*

Wikipedia (Wikipedia-bijdragers, 2021) beschrijft het master en slave model als 'Master/slave is a model of asymmetric communication or control where one device or process (the "master") controls one or more other devices or processes (the "slaves") and serves as their communication hub. In some systems, a master is selected from a group of eligible devices, with the other devices acting in the role of slaves.'[4]

Het master en slave model kan gebruikt worden als om de deelvragen op te lossen. Dit model staat ook bekend als het orchestrator en worker model en is aanbevolen vanuit Fonto. In het onderzoek wordt er dieper ingegaan op deze materie.

Methodiek: *Scrum*

Wikipedia (Wikipedia-bijdragers, 2020) beschrijft Scrum als 'Scrum is een framework om op een flexibele manier (software)producten te maken. Er wordt gewerkt in multidisciplinaire teams die in korte sprints, met een vaste lengte van 1 tot 4 weken, werkende (software) producten opleveren. Scrum is een term die afkomstig is uit de rugbysport. Bij een scrum probeert een team samen een doel te bereiken en de wedstrijd te winnen. Samenwerking is heel belangrijk en men moet snel kunnen inspelen op veranderende omstandigheden.'[5]

Tijdens het project wordt de Scrum methodiek toegepast. Deze keuze is gemaakt omdat deze methodiek centraal stond tijdens de opleiding en voorafgaande stages. Daarnaast wordt Scrum, binnen Fonto, ook als standaard gebruikt.

Er worden sprints van twee weken gehanteerd, omdat deze tijdsduur ook binnen de development teams van Fonto wordt aangehouden. Daarnaast wordt elke vrijdag door middel van een review de voortgang, planning en eventuele problemen behandeld in samenwerking met de begeleiders vanuit Fonto. Verder wordt er dagelijks contact gelegd met de begeleiders. In het begin van het project zal dit niet via de traditionele stand-up procedure gaan, maar zal er contact gelegd worden zodra er zich problemen voordoen of om een update van de voortgang te geven. Later in het traject kan dit eventueel omgezet worden naar de traditionele stand-up, zodra er behoefte is aan een vast dagelijks contact moment.

Methodiek: *UML*

Stefan Cruysberghs (Cruysberghs, 2002) beschrijf UML als 'De Unified Modeling Language (UML) is een van de meest gebruikte methodes om de visies van systeemontwikkelaars begrijpelijk en gestandaardiseerd vast te leggen. Het laat je toe om diagrammen te tekenen die begrijpelijk zijn voor de klant, de analist en de programmeur. UML is ontstaan begin jaren '90 maar sinds de oprichting van het UML-consortium in 1997 is het de facto standaard geworden in de software-industrie.'[6]

UML is een methode om ontwerpen vast te leggen. Deze methode gaat gebruikt worden om de huidige situatie en het nieuwe ontwerp vast te leggen. UML is vastgelegd, omdat het een veelgebruikte methode is die wel bekend is bij techbedrijven.

Methodiek: *Jira*

De website Airfocus (What is Jira?, z.d.) beschrijft Jira als 'Jira is a project management tool developed by Atlassian. It was first launched in 2002 as an issue tracking system, but now encompasses several key processes in agile project development.'

Met behulp van Jira wordt de agile development ondersteund. Zo worden de sprints en backlog hierin opgenomen. Hierdoor hebben alle betrokken partijen beter zicht op de planning, de taken die nog uitgevoerd moeten worden en de taken die al uitgevoerd zijn.

6.0 Activiteiten

6.1 Initiatiefase

Plan van aanpak

Gedurende de eerste week wordt het plan van aanpak opgesteld. In het plan van aanpak wordt duidelijk gemaakt waarom de afstudeeropdracht wordt uitgevoerd en waarom deze van waarde is voor Fonto. Daarnaast wordt er toegelicht hoe de afstudeeropdracht uitgevoerd gaat worden. Het doel van het plan van aanpak is om een concrete basis van het project op te zetten, die op een ondersteunende manier werkt voor de student, de begeleiders en Fonto.

Requirements inventariseren

Om aan de wensen en eisen van Fonto te voldoen worden de requirements geïnventariseerd. Doordat de requirements in een vroeg stadia opgesteld worden, kan er op een gerichtere wijze onderzoek worden uitgevoerd. Dit resulteert in een concreet en specifiek onderzoek, waarbij de wensen en eisen van Fonto centraal staan.

6.2 Analysefase

Onderzoek

Om de hoofd- en deelvragen te beantwoorden wordt er onderzoek uitgevoerd. Tijdens de projectperiode zal er voornamelijk deskresearch worden uitgevoerd. Hierbij wordt er literatuur verzameld om zo antwoord te geven op de hoofd- en deelvragen. Daarnaast wordt toegepast onderzoek gebruikt om tot een geschikt prototype te komen.

In kaart brengen huidige situatie

Om in staat te kunnen zijn een nieuw ontwerp op te stellen, is het van belang de huidige situatie in kaart te brengen. Dit ontwerp stelt verschillende componenten tentoon, zoals de werking met de database, logging en de algemene functionaliteit.

6.3 Ontwerpfase

Ontwerp opstellen

Op basis van de requirements, het onderzoek en het ontwerp van de huidige situatie zal het nieuwe ontwerp worden opgesteld. Dit ontwerp zal leidend zijn voor het prototype en het biedt beide partijen de mogelijkheid om kritisch de gemaakte keuzes te behandelen. Dit resulteert in ontwerp dat gebaseerd is op het onderzoek, maar ook gestuurd naar de eisen en wensen van Fonto.

6.4 Realisatiefase

Prototype realiseren

Op basis van het ontwerp wordt een proof of concept ontwikkeld. Het doel hiervan is om aan te tonen dat het voorafgaande ontwerp in combinatie met het uitgevoerde onderzoek functioneel is. Wanneer blijkt dat het prototype bruikbaar is, kan deze door Fonto in de nabije toekomst gebruikt worden als uitgangspunt. Blijkt dit niet het geval, dan kan de reden tot falen opgenomen worden in het adviesrapport. Hiermee biedt het prototype Fonto altijd een positieve uitkomst.

6.5 Testfase

Prototype testen

Zodra het prototype ontwikkeld is kan deze getoetst worden. Door middel van verschillende testen wordt er vastgelegd in welke mate het prototype bruikbaar is. Deze informatie zal opgenomen worden in het adviesrapport en bepaald uiteindelijk hoe succesvol het prototype is geworden

6.6 Afsluitfase

Adviesrapport opstellen

Aan het eind van de projectperiode wordt een adviesrapport opgeleverd. Dit rapport bevat op basis van het onderzoek en alle gemaakte keuzes een advies. Dit advies ondersteund Fonto in toekomstige keuzes, die betrekking hebben op de test runner.

7.0 Risicoanalyse

Aan het begin van het afstudeertraject is het van belang om de projectrisico's te inventariseren. Hierdoor kunnen beide partijen stappen nemen om deze te voorkomen en weten ze hoe ze moeten reageren, zodra een specifiek risico zich voordoet. In de risicoanalyse zijn verschillende risico's opgenomen. Hierbij zijn ook stappen beschreven waarmee het desbetreffende risico voorkomen en opgelost kan worden.

Risico	Hoe te voorkomen	Hoe op te lossen
Verlies van data	Om het verlies van data te voorkomen zal informatie op verschillende plekken moeten worden opgeslagen. Onderzoek en verslagen moeten lokaal en op de cloud worden opgeslagen. Gemaakte programmatuur zal lokaal en in de repository bewaard moeten worden.	Wanneer er data verloren gaat moet dit hersteld worden vanuit de back-up. Daarnaast moet het probleem vastgesteld worden, zodat dit zich niet nogmaals voordoet.
Onduidelijke verwachtingen vanuit het bedrijf	Er moeten in samenwerking met het bedrijf concrete verwachtingen en eisen worden opgesteld, zodat beide partijen en heldere verwachtingen hebben.	In het geval van onduidelijke verwachtingen zullen beide partijen met elkaar in gesprek zullen moeten gaan om de verwachtingen op te helderen.
Defecte laptop	Er zal voorzichtig om moeten worden gegaan met de apparatuur die benodigd is voor het afronden van de afstudeeropdracht.	Er zal overgeschakeld worden naar een oude laptop. Hierdoor kunnen de activiteiten hervat worden.
Tekort aan expertise bij begeleiders	Om dit te voorkomen zal de opdracht grotendeels binnen het domein van het bedrijf moeten vallen.	Zodra de begeleiders niet beschikken over project gerelateerde kennis, wordt er om hulp gevraagd bij andere medewerkers. Zo bevatten de verschillende development teams meerdere ontwikkelaars die elk over kennis beschikken van andere materie.
Student ziek voor een lange periode	-	Wanneer er zich een langdurige ziekte voordoet, zal er zoveel mogelijk worden doorgewerkt. Als de ziekte zo significant is dat het afstudeerproject vertraging op loopt, zal dit besproken moeten worden met de begeleidend examinator.

Planning niet haalbaar	De planning moet besproken worden met de begeleiders, zodat deze extra gecontroleerd wordt. Verder moeten de deadlines worden opgesteld met oog op de resterende hoeveel tijd en werk.	Dit zal overlegd moeten worden met de begeleiders vanuit Fonto. Er zal een keuze gemaakt moeten op basis van de resterende tijd en hoeveelheid werk. Deze keuze kan leiden tot het versimpelen of afzien van enkele eindproducten.

Tabel 11: Risicoanalyse

8.0 Werkplanning

De planning is opgesteld zodat alle partijen een duidelijk beeld krijgen van de projecttijdlijn. Elke twee weken wordt aangeduid als een sprint wat resulteert in een totaal aantal van zeven sprints. Tevens loopt de planning van 8 februari 2021 tot 4 juni 2021.

	Sprint	Sprint 0		Sprint 1		Sprint 2		Sprint 3		Sprint 4		Sprint 5		Sprint 6		Sprint 7		Sprint 8
	Datum	8-feb.	15-feb.	22-feb.	1-mrt.	8-mrt.	15-mrt.	22-mrt.	29-mrt.	5-apr.	12-apr.	19-apr.	26-apr.	3-mei	10-mei	17-mei	24-mei	31-mei
	Week	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Initiatiefase																		
Plan van aanpak																		
Requirements opstellen																		
Analysefase																		
Huidige situatie in kaart brengen																		
Onderzoeksrapport																		
Ontwerpfase																		
Ontwerp test runner 2.0																		
Realisatiefase																		
Prototype																		
Testfase																		
Testrapport																		
Afsluitfase																		
Adviesrapport																		
Scriptie schrijven																		
Deadlines																		
Bedrijfsbezoek																		
Bespreking concept afstudeerplan																		
Tussentijds assesment																		
Inleveren scriptie																		
Voortgangsgesprek																		
Evaluatieformulier																		

Figuur 25: Planning afstudeertraject

Literatuurlijst

1. *Wikipedia-bijdragers. (2020, 20 oktober). Cloud computing. Wikipedia.* https://nl.wikipedia.org/wiki/Cloud_computing
2. *Hoes, A. G. (2017, mei). Schaalbaarheid. Ensie.* https://www.ensie.nl/frank-hoes/schaalbaarheid
3. *Wikipedia-bijdragers. (2021, 15 januari). Software performance testing. Wikipedia.* https://en.wikipedia.org/wiki/Software_performance_testing
4. *Wikipedia- bijdragers. (2021, februari 6). Master/slave (technology). Wikipedia.* https://en.wikipedia.org/wiki/Master/slave_(technology)
5. *Wikipedia-bijdragers. (2020, september 2). Scrum. Wikipedia.* https://nl.wikipedia.org/wiki/Scrum_(softwareontwikkelmethode)
6. *Cruysberghs, S. (2002, juni). Wat is UML?* http://www.scip.be/PDF/Wat_is_UML.pdf
7. *What is Jira? (z.d.). Airfocus.* https://airfocus.com/glossary/what-is-jira/

Bijlage B: Cloud provider families

Microsoft Azure

Familie	Ingericht voor	Prijs per maand vanaf
A-Series	Beginners	€9,02
Bs-Series	Economisch	€1,64
D-Series	Standaard gebruik	€33,63
DC-Series	Data protectie	€39,37
E-Series	Geheugen	€47,57
F-Series	Vermogen	€28,71
G-Series	Geheugen & opslag	€264,46
H-Series	Performance	€476,52
Ls-Series	Opslag	€373,18
M-Series	Geheugen	€919,41
Mv2-Series	Geheugen	€13.357,34
N-series	GPU	€538,85

Tabel 12: Families Microsoft Azure [12]

AWS

Ingericht voor	Prijs per maand vanaf
Standaard gebruik	€14,76
Performance	€20,50
Geheugen	€29,53
Opslag	€91,86
GPU	€19.487,31

Tabel 13: Families AWS [13]

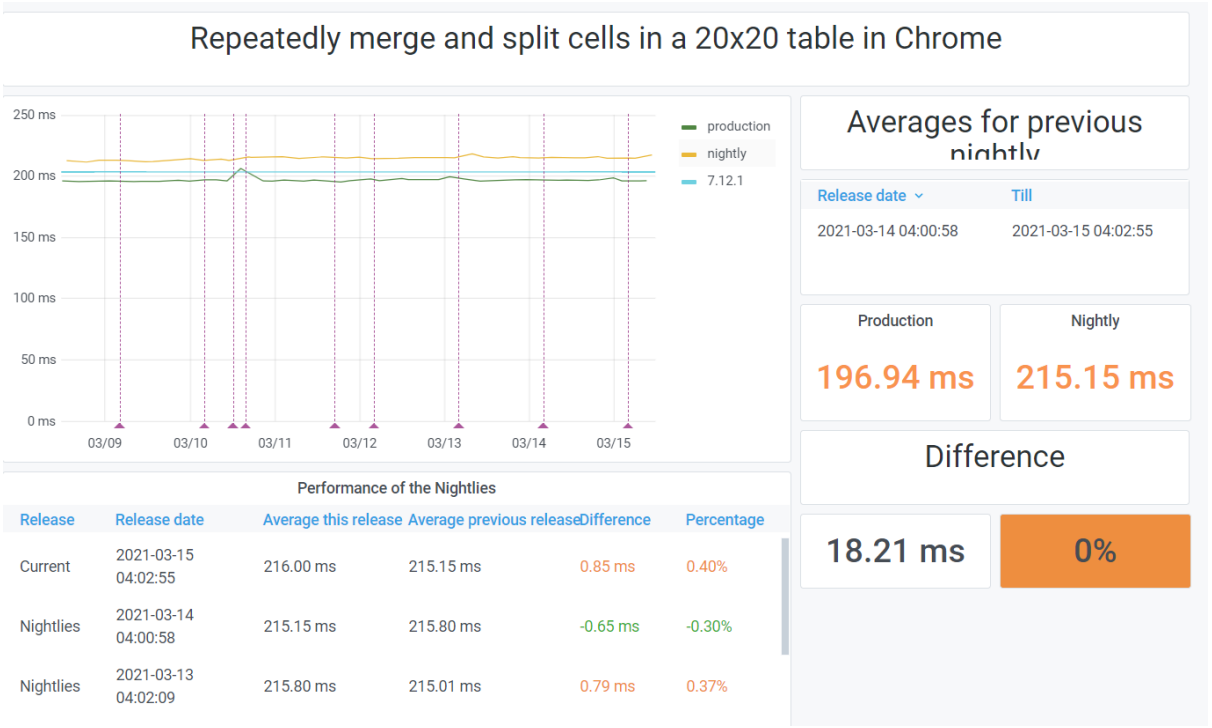
GCE

Familie	Ingericht voor	Prijs per maand vanaf
E2	Standaard gebruik	€39,37
N2/N2D/N1	Standaard gebruik	€57,41
M1/M2	Geheugen gebruik	€785,73
C2	Performance gebruik	€123,85
A2	GPU gebruik	€649,58

Tabel 14: Families GCE [14]

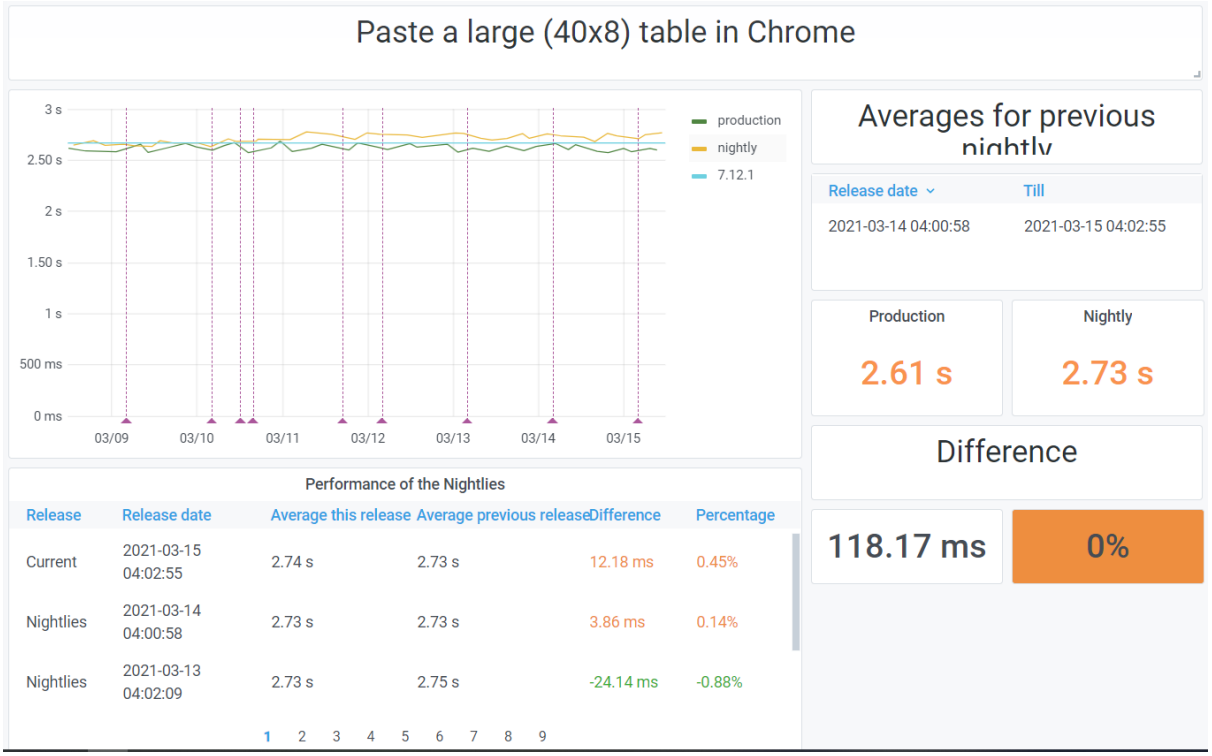
Bijlage C: Testselectie

In bijlage C: Testselectie zijn de testen afgebeeld die gebruikt worden tijdens de test in de cloud. De afbeeldingen zijn afkomstig van het Grafana dashboard. De onderbouwing voor het gebruik van deze selectie testen is gedefinieerd in hoofdstuk 9.6.

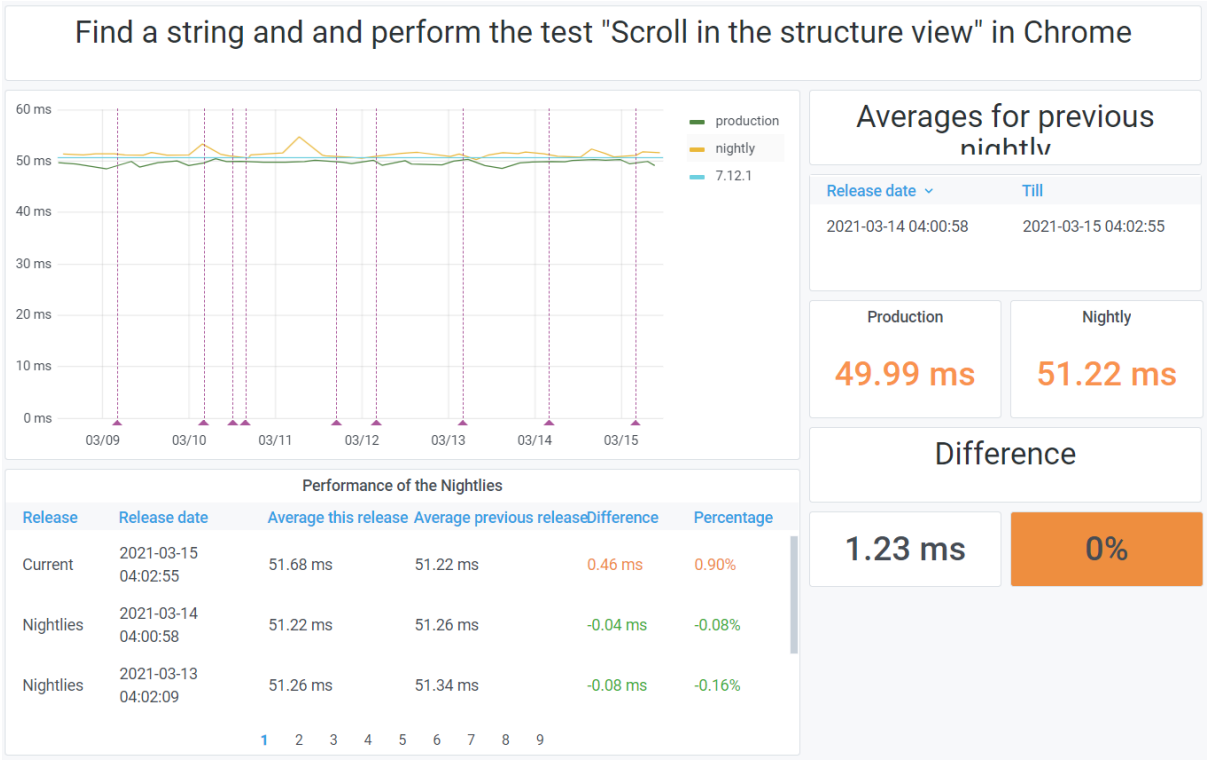


Figuur 26: "Merge and split table cells" performance test

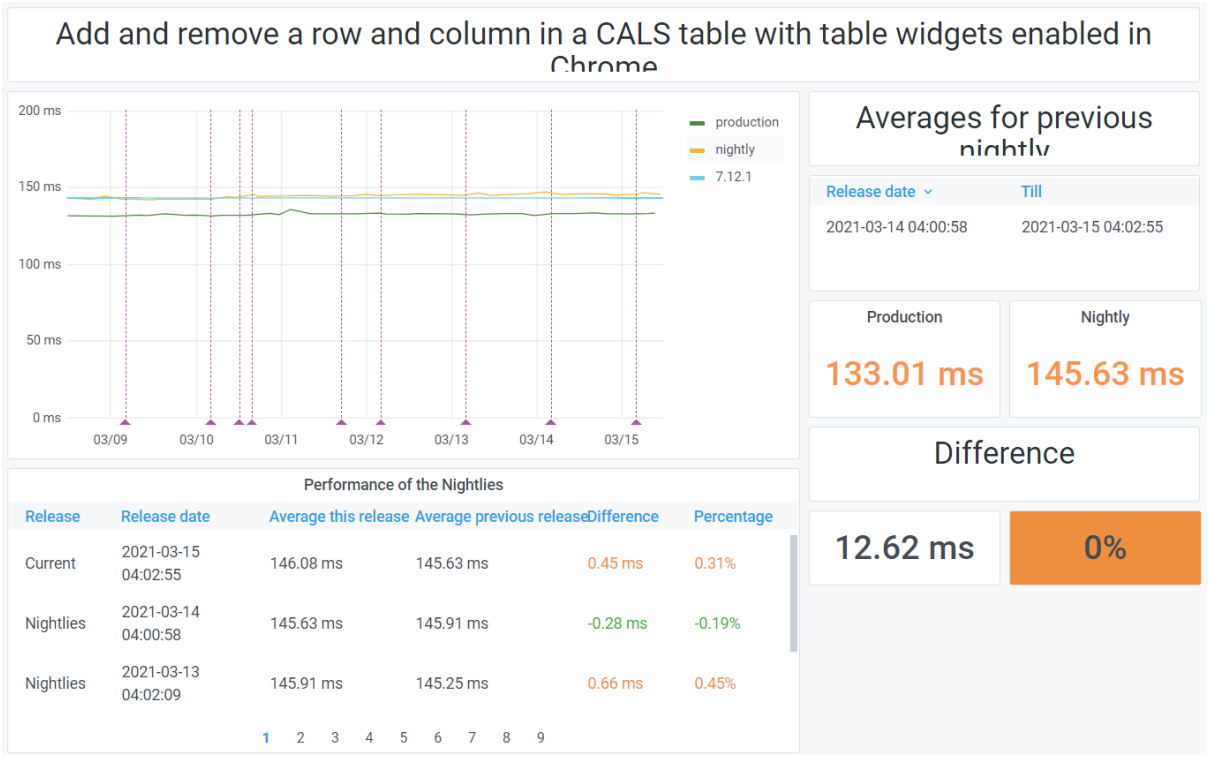
Figuur 27: "Find and write" performance test



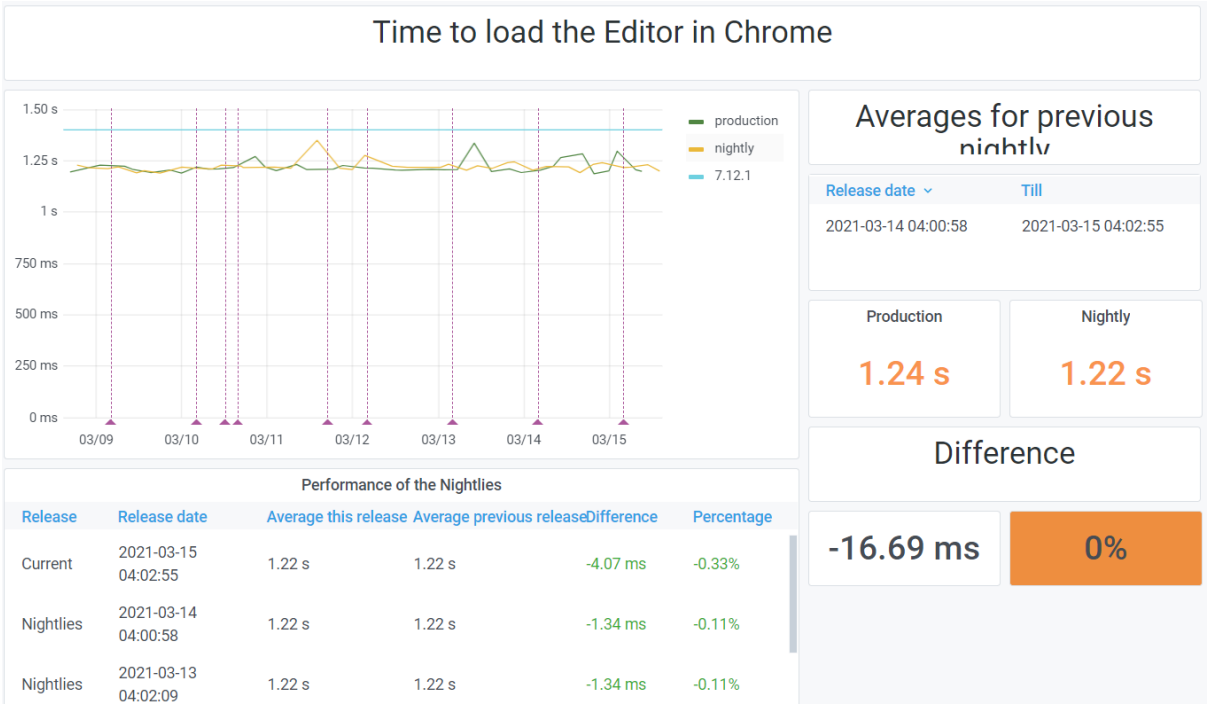
Figuur 28: "Paste large table" performance test



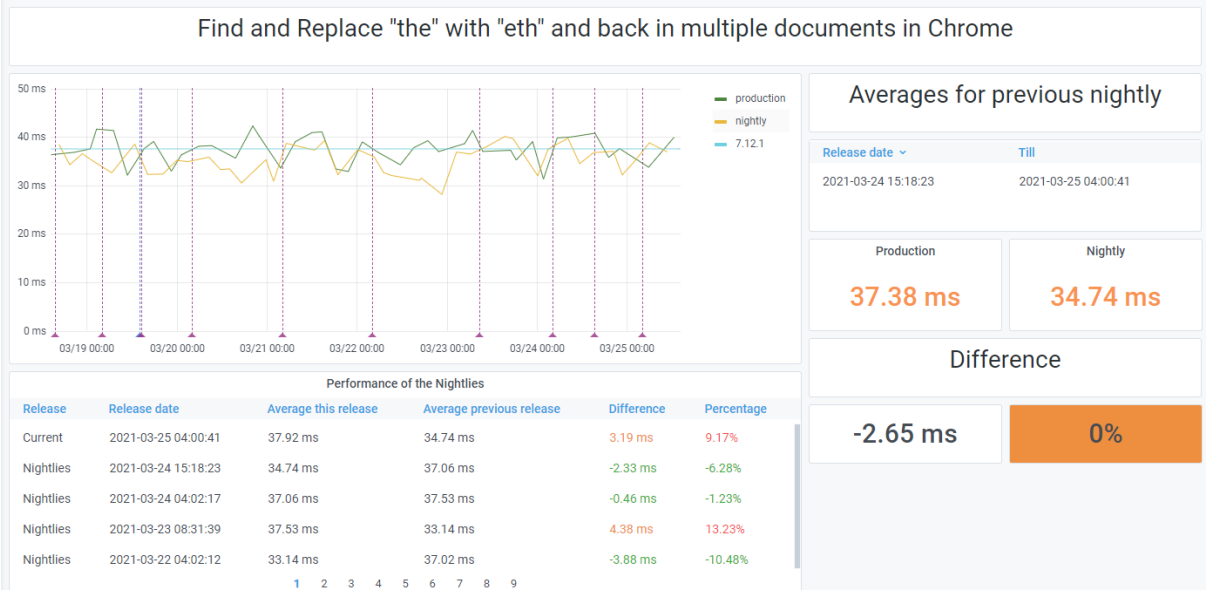
Figuur 29: "Find and scroll" performance test



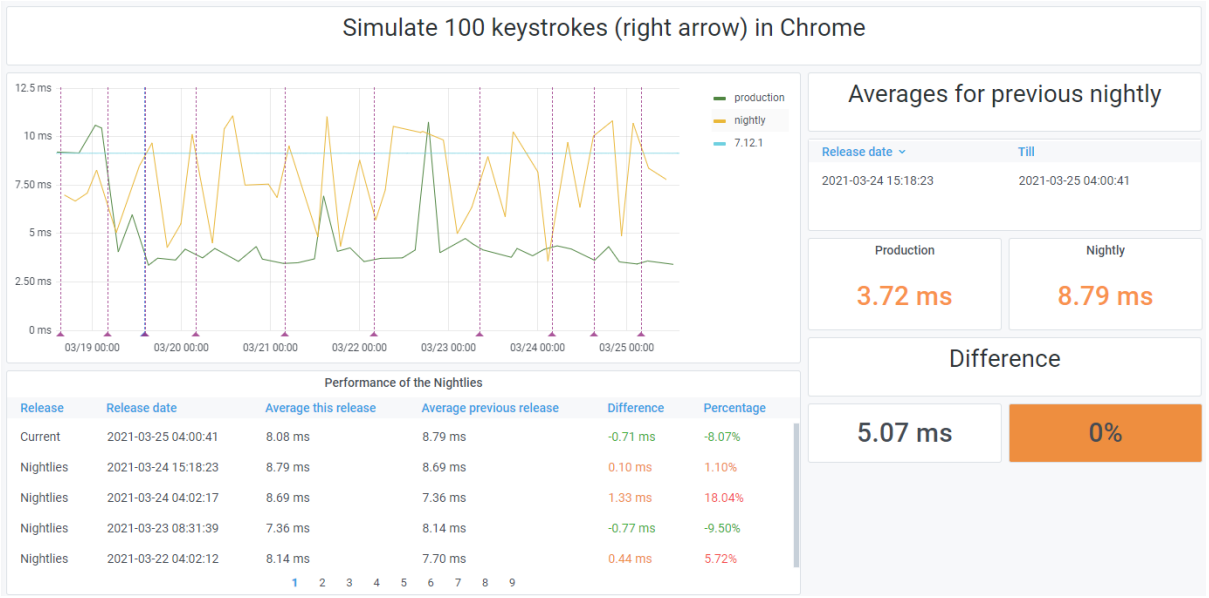
Figuur 30: "Add and remove CALS table" performance test



Figuur 31: "Editor load time" performance test



Figuur 32: "Find and replace" performance test



Figuur 33: "100 keystrokes" performance test

Bijlage D: Kosten Linux en Windows op AWS

Linux			Windows	
t3.nano	$0.0052 per Hour		t3.nano	$0.0098 per Hour
t3.micro	$0.0104 per Hour		t3.micro	$0.0196 per Hour
t3.small	$0.0208 per Hour		t3.small	$0.0392 per Hour
t3.medium	$0.0416 per Hour		t3.medium	$0.06 per Hour
t3.large	$0.0832 per Hour		t3.large	$0.1108 per Hour
t3.xlarge	$0.1664 per Hour		t3.xlarge	$0.24 per Hour
t3.2xlarge	$0.3328 per Hour		t3.2xlarge	$0.48 per Hour
t3a.nano	$0.0047 per Hour		t3a.nano	$0.0093 per Hour
t3a.micro	$0.0094 per Hour		t3a.micro	$0.0186 per Hour
t3a.small	$0.0188 per Hour		t3a.small	$0.0372 per Hour
t3a.medium	$0.0376 per Hour		t3a.medium	$0.056 per Hour
t3a.large	$0.0752 per Hour		t3a.large	$0.1028 per Hour
t3a.xlarge	$0.1504 per Hour		t3a.xlarge	$0.224 per Hour
t3a.2xlarge	$0.3008 per Hour		t3a.2xlarge	$0.448 per Hour

Figuur 34: Kosten Linux en Windows instances op AWS [13]

Bijlage E: Verbruik van de test runner

De impact van de test runner op de hardware is gemeten op een computer met de specificaties weergeven in tabel 16. Om de impact accuraat te meten is er gebruik gemaakt van een software programma genaamd Process Explorer.[20] Doormiddel van dit programma kan het gebruik van de CPU, geheugen, schijf en GPU geanalyseerd worden per actief proces. Het doel van deze meting is om in kaart te brengen welke onderdelen van de hardware intensief gebruikt worden door de test runner. Vervolgens wordt er met behulp van de metingen naast de standaard instance ook een instance geutiliseerd die opgezet is voor een bepaald doel, zoals zwaar CPU gebruik.

Om de resultaten van de meting zo betrouwbaar mogelijk te houden zijn er meerdere metingen gedaan. Daarnaast zijn alle applicaties en achtergrondprocessen afgesloten zodat al het verbruik weergegeven op de figuren hieronder afkomstig is van de test runner.

Tijdens een meting worden de testen uitgevoerd op één browser. Hierdoor blijven de grafieken kort en overzichtelijk. De testen die uitgevoerd worden tijdens de meting zijn gedefinieerd in hoofdstuk 9.6. Dit zijn tevens ook de testen die uitgevoerd gaan worden op de geselecteerde instances. Voor elke browser is de meting drie keer uitgevoerd waarbij bleek dat deze vergelijkbare resultaten opleveren. Hierdoor zijn drie metingen per browser voldoende. In de figuren 35 en 37 is één van de metingen van Chrome en Firefox weergegeven.

Bij de meting is te concluderen dat het geheugen nauwelijks wordt aangepast door de test runner. Dit komt mede doordat de computer waarop de test is uitgevoerd beschikt over een significante hoeveelheid geheugen. Hierdoor zijn veranderingen van een paar honderd MB lastig waar te nemen. De verschillende instances bieden over het algemeen dezelfde hoeveelheid geheugen aan als de computer waarop de meting is uitgevoerd. Verder is te concluderen dat Firefox intensiever gebruik maakt van de hardware dan Chrome. De pieken in CPU en I/O gebruik liggen bij Firefox significant hoger dan bij Chrome. Jos Verburg, de ontwikkelaar van de test runner, geeft hierbij aan dat de editor beter geoptimaliseerd is voor Chrome. Dit verklaart het hogere verbruik van Firefox. Naast het hogere gebruik van hardware resources ligt de uitvoeringstijd van testen die uitgevoerd worden op Firefox gemiddeld 50% tot 100% hoger dan dezelfde testen uitgevoerd op Chrome. Verder blijkt dat de CPU niet hevig gebruikt wordt door de test runner, de pieken zijn tevens een stuk lager vergeleken met die van de I/O, zoals weergegeven in figuur 36. De grafieken die het verbruik van de CPU weergeven doen dit door het verbruik van alle cores samen te voegen. Zodra het verbruik van elke CPU in kaart wordt gebracht blijkt dat één van de cores verantwoordelijk is voor het merendeel van het verbruik. Dit is te verklaren doordat de test runner ontwikkeld is in Javascript. Javascript is een single threaded programmeertaal wat er toe leidt dat het tijdens uitvoering alleen gebruik maakt van één core.[53]

Weerlegging I/O

In de figuren 35 en 37 is te zien dat de test runner hevig gebruik maakt van de I/O. Hierbij blijkt na nader onderzoek dat deze pieken voortkomen uit het opstartproces van de browser en niet van de uitgevoerde test. Omdat er tijdens het uitvoeren van de testen nauwelijks gebruik wordt gemaakt van de I/O, is het niet vereist om een instance te selecteren die geoptimaliseerd is voor intensief I/O gebruik.

Conclusie

Hieronder worden de conclusies besproken die voortkomen uit de metingen. Deze conclusies worden gebruikt om op elke cloud provider een extra instance te selecteren, die geoptimaliseerd is voor specifiek gebruik.

a. *Een instance met veel geheugen is niet van toepassing*

Het geheugen wordt tijdens de uitvoering van de test runner nauwelijks aangetast. De instances bieden daarnaast minimaal dezelfde hoeveelheid geheugen aan. Het is dus overbodig om een instance in te schakelen die geoptimaliseerd is voor zwaar geheugen gebruik. Een instance met een vergelijkbare hoeveelheid RAM is hierbij voldoende.

b. *Meer dan twee cores zijn overbodig*

De test runner is ontwikkeld in een single threaded programmeertaal genaamd Javascript.[15] Dit houdt in dat de test runner tijdens uitvoering alleen gebruik maakt van één core, zoals weergegeven op figuur 39. Doordat de test runner maar één core utiliseert is het overbodig om een instance te utiliseren die meerdere cores tot zijn beschikking heeft. Daarnaast biedt een extra core de mogelijkheid om achtergrondprocessen af te handelen.

c. *Instance met beter presterende cores kan betrouwbaardere resultaten bieden*

Zoals weergegeven in figuur 39 wordt de CPU in zijn geheel niet intensief gebruikt. Hierbij wordt één core wel intensief gebruikt, mede doordat Javascript een single threaded programmeertaal is. Een of twee krachtige cores zijn in dit geval beter dan zes standaard cores.

d. *Een verhoogde I/O capaciteit kan betrouwbaardere resultaten opleveren*

Zoals weergegeven in figuur 35 en 37 maakt de test runner intensief gebruik van de I/O. In het onderzoek (Laaber et al., 2018) beschreven in hoofdstuk 8.2 is geconcludeerd dat programmatuur dat hevig gebruikt maakt van de I/O minder betrouwbare resultaten oplevert. Hierbij wordt vermeld dat het kiezen van een instance geoptimaliseerd voor hevig I/O gebruik de betrouwbaarheid van de resultaten zou kunnen verhogen. In eerste instantie zou de test runner hiervan

kunnen profiteren. Na verder onderzoek, zoals beschreven onder de kop weerlegging I/O, blijkt dat tijdens het testproces nauwelijks I/O hevige operaties uitvoert.

e. *Firefox weegt zwaarder op de hardware dan Chrome*

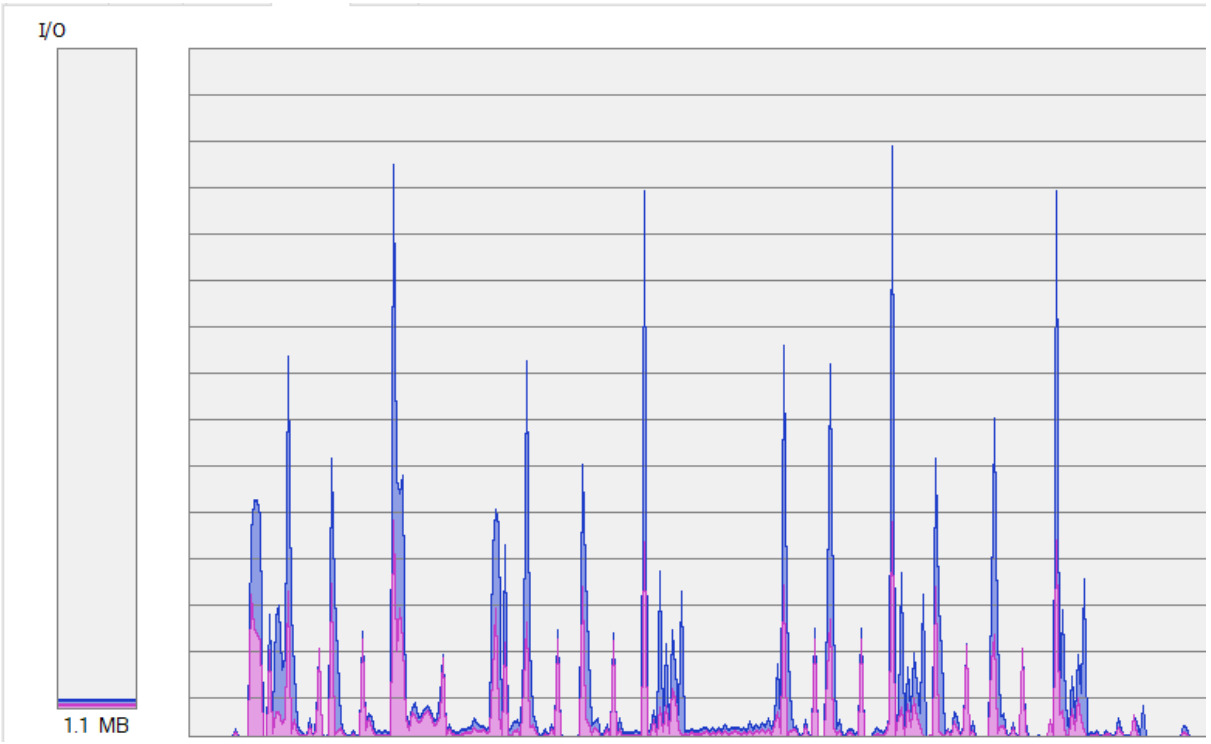
Alhoewel de testen uitgevoerd op Firefox intensiever gebruik maken van de hardware, heeft deze specifieke conclusie geen invloed op de instance keuze. Beide browser moeten namelijk zonder performance problemen uitgevoerd kunnen worden.

Conclusie	Impact op instance keuze
a	Instance hoeft niet geoptimaliseerd te worden voor zwaar geheugengebruik.
b	Instance heeft voldoende aan twee cores.
c	Instance met beter presterende cores biedt betrouwbaardere resultaten.
d	Instance hoeft niet geoptimaliseerd te worden voor zwaar I/O gebruik.
e	Heeft geen invloed op de instance keuze.

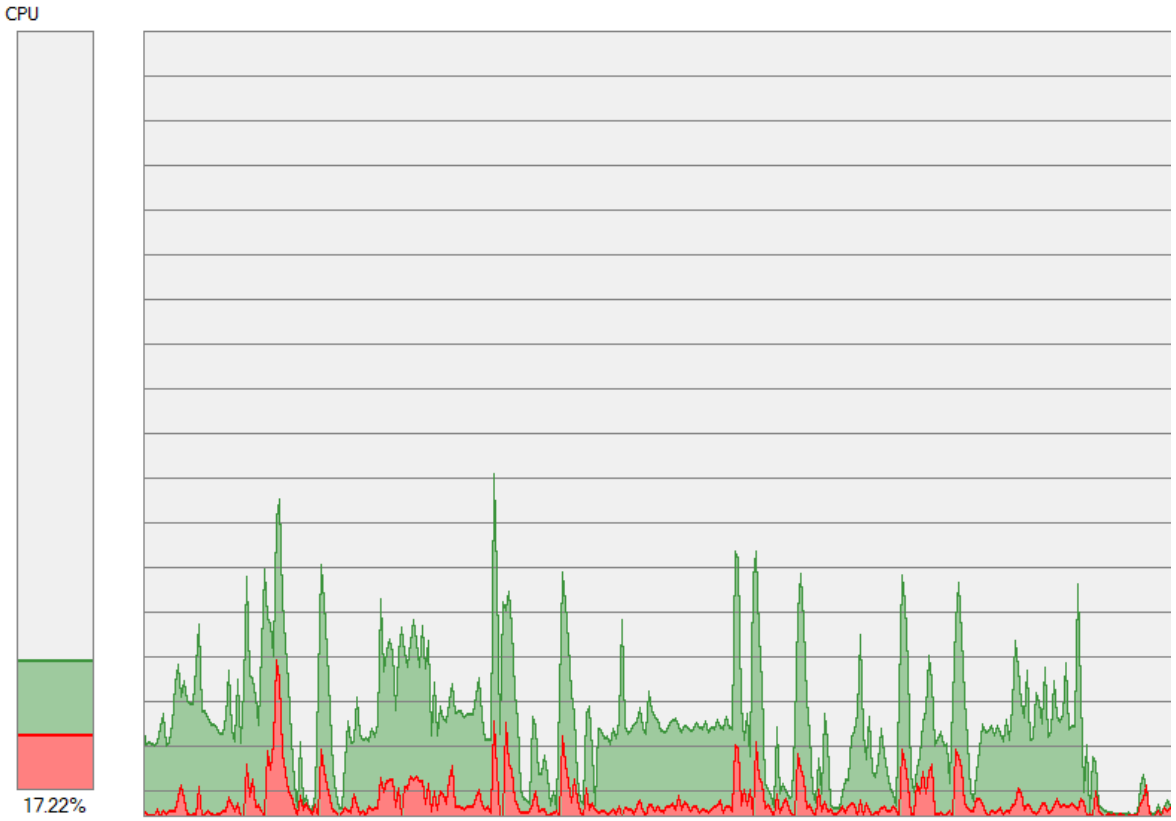
Tabel 15: Conclusies verbruik test runner

Onderdeel	informatie	
Processor	2.6GHz	Intel i7 - 6 cores
Geheugen	16GB RAM	41.8 GB/s
Schijf	SSD	256GB
GPU	350 MHz	64GB
Systeem	64-bits	

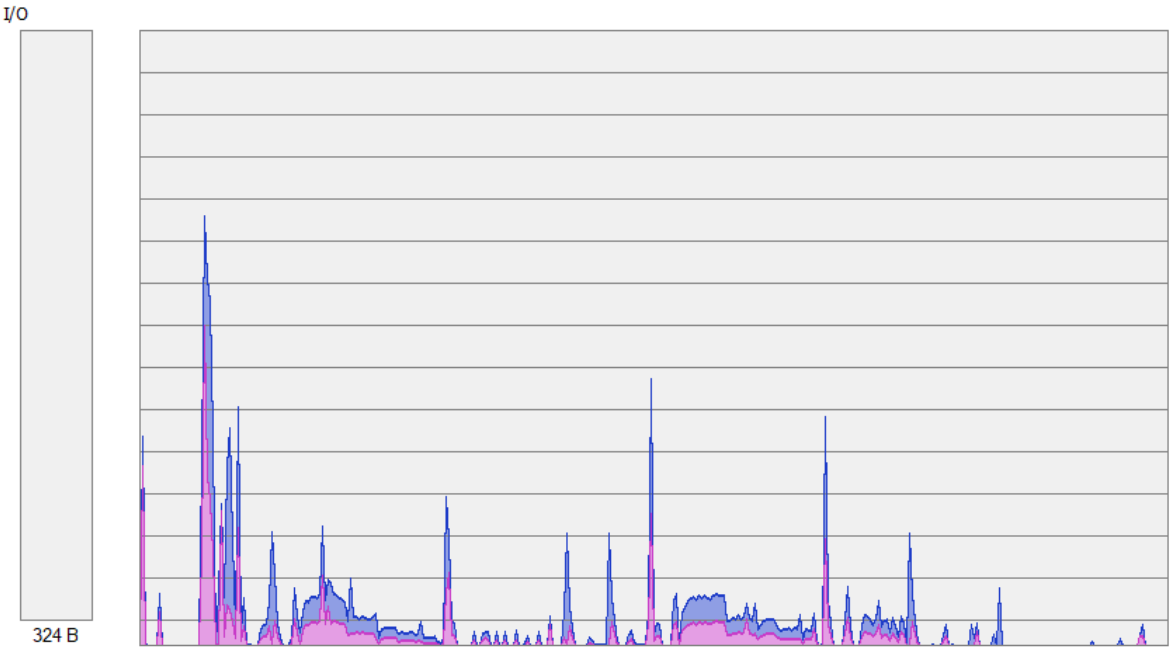
Tabel 16: Specificaties van de geteste computer



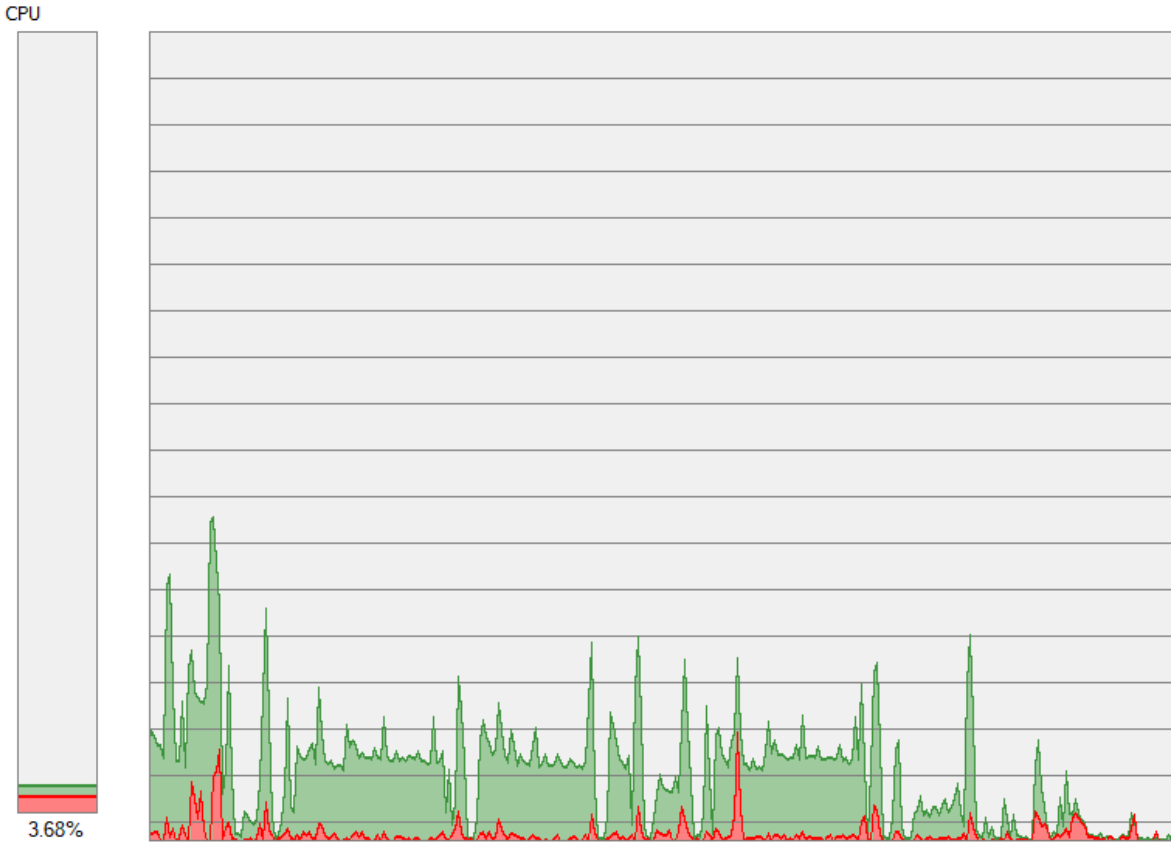
Figuur 35: Het I/O verbruik van performance test uitgevoerd op Firefox



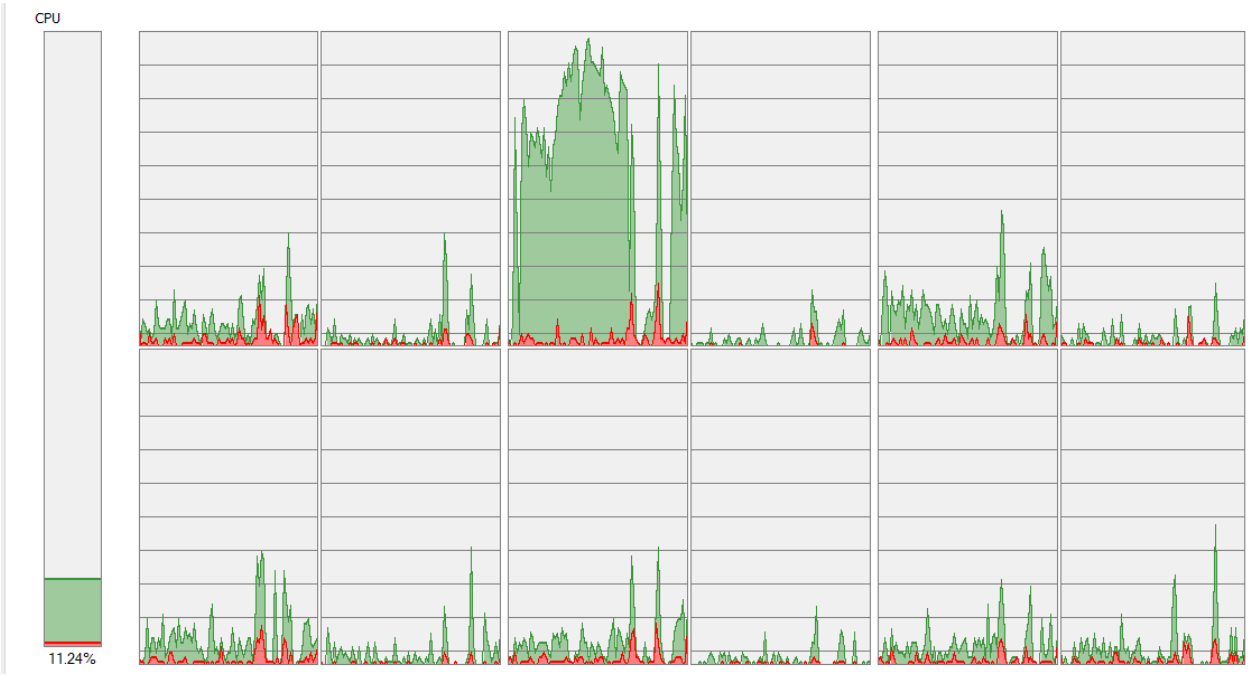
Figuur 36: Het CPU verbruik van performance test uitgevoerd op Firefox



Figuur 37: Het I/O verbruik van performance test uitgevoerd op Chrome

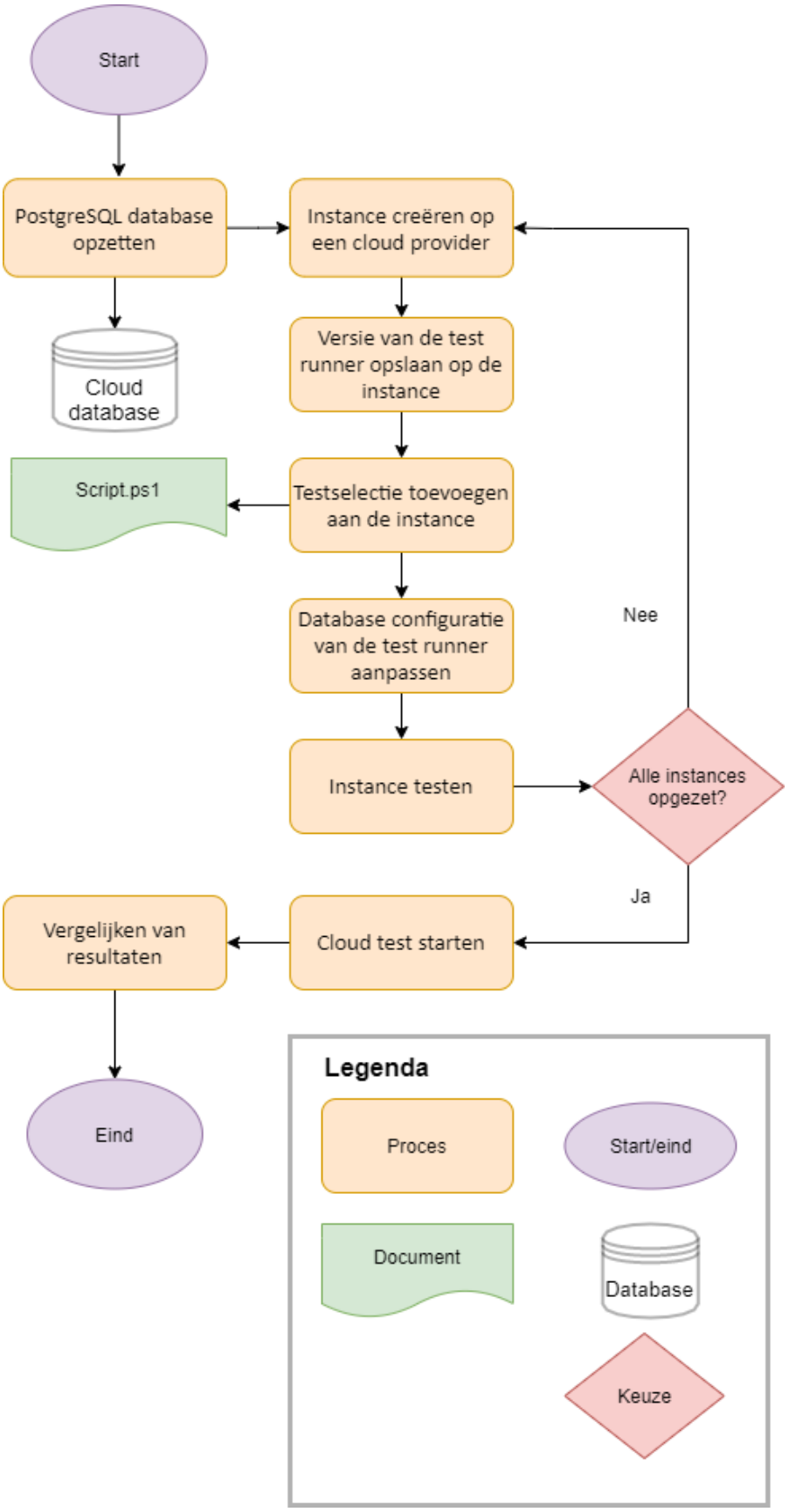


Figuur 38: Het CPU verbruik van performance test uitgevoerd op Chrome



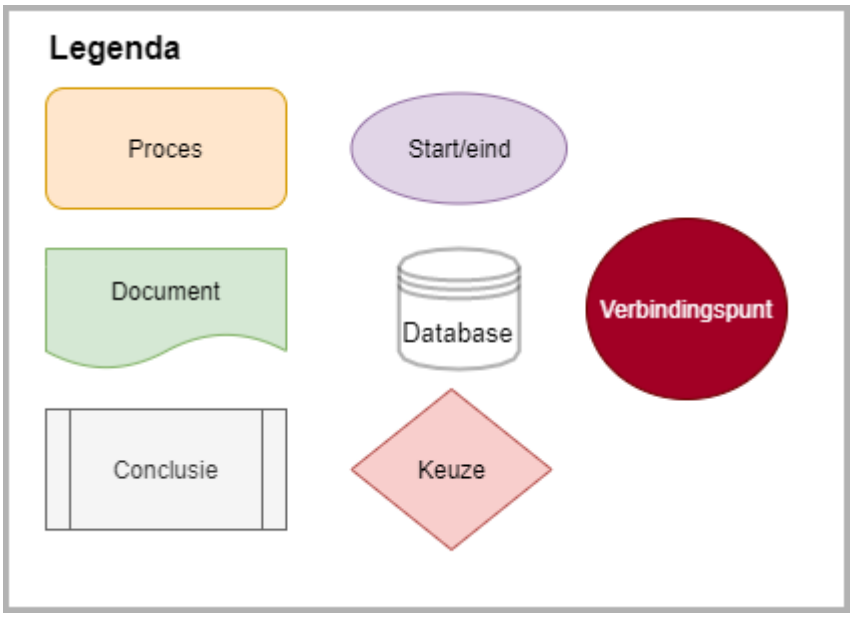
Figuur 39: CPU Verbruik verschillende cores test runner

Bijlage F: Flowchart aanpak

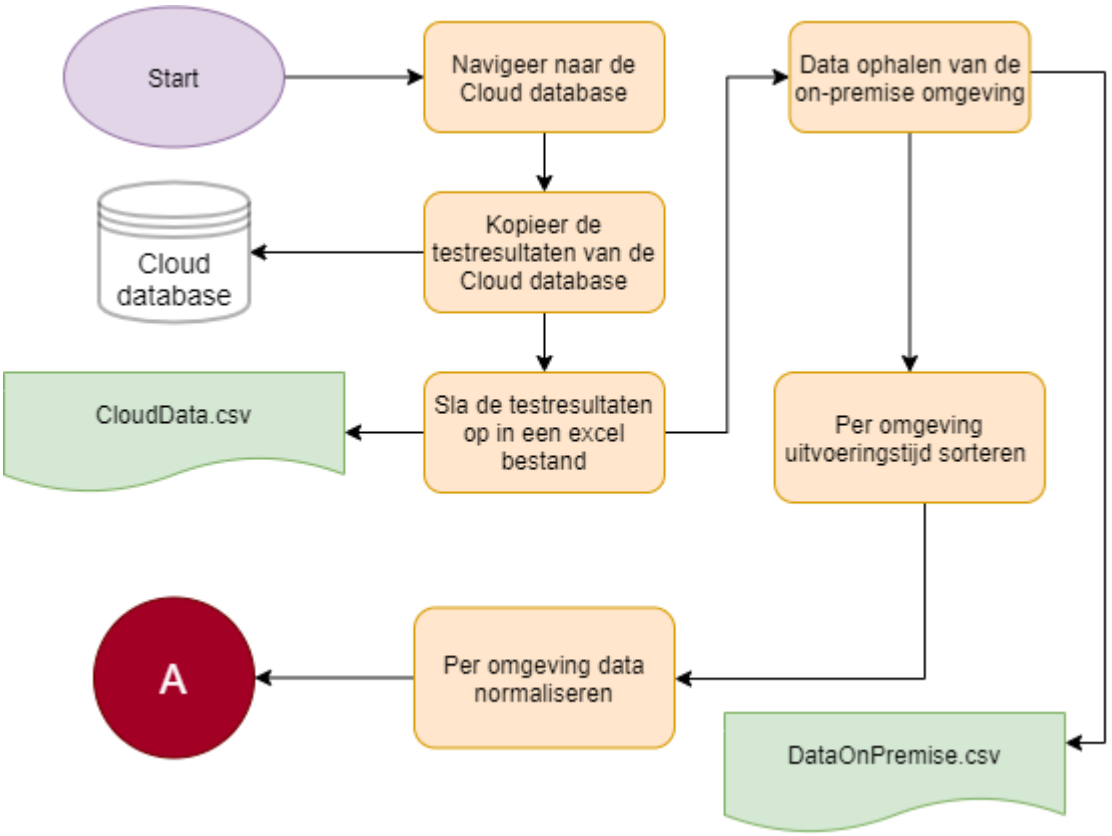


Figuur 40: Flowchart aanpak testplan

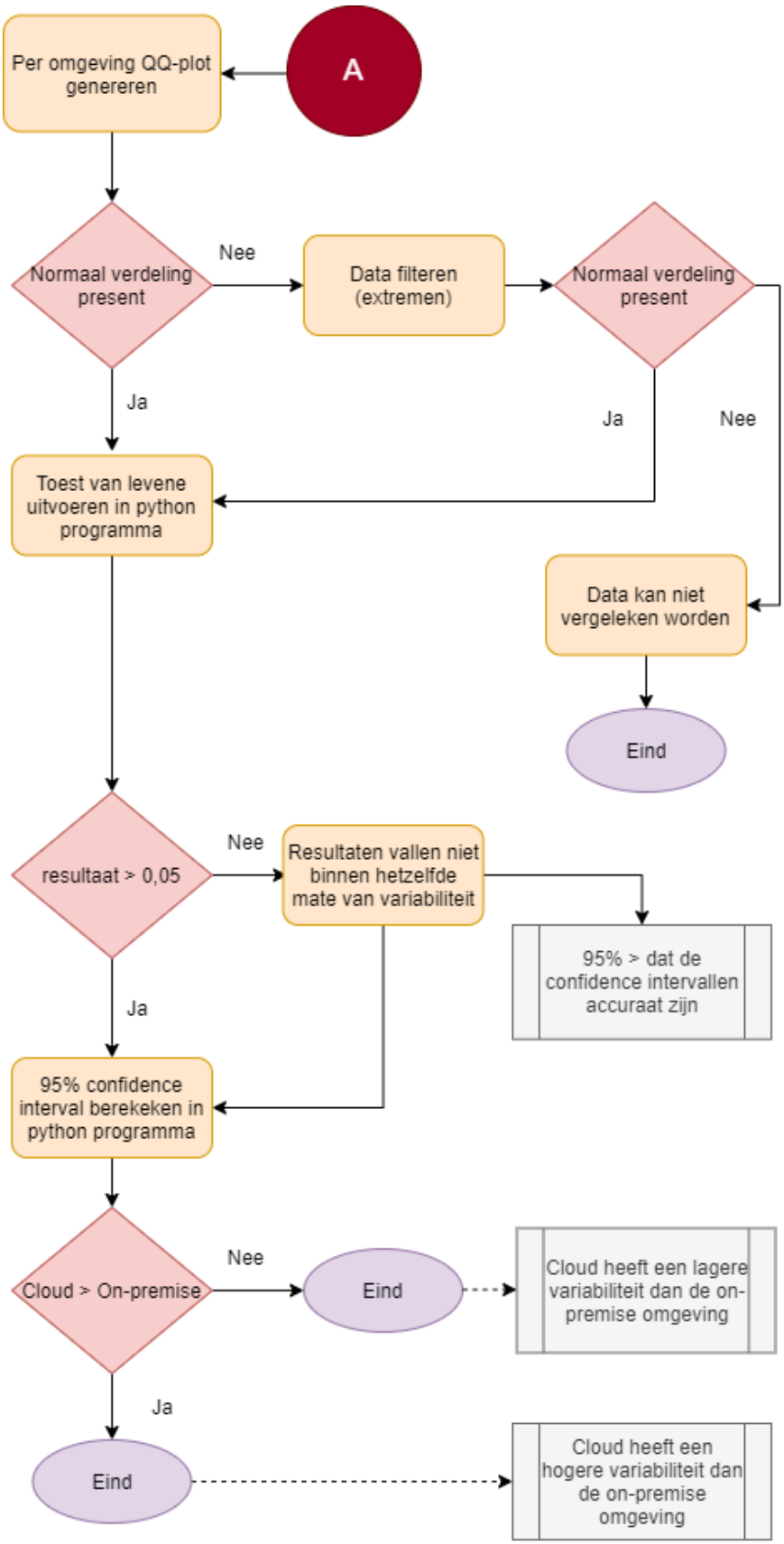
Bijlage G: Flowchart resultaten



Figuur 41: Legenda flowcharts



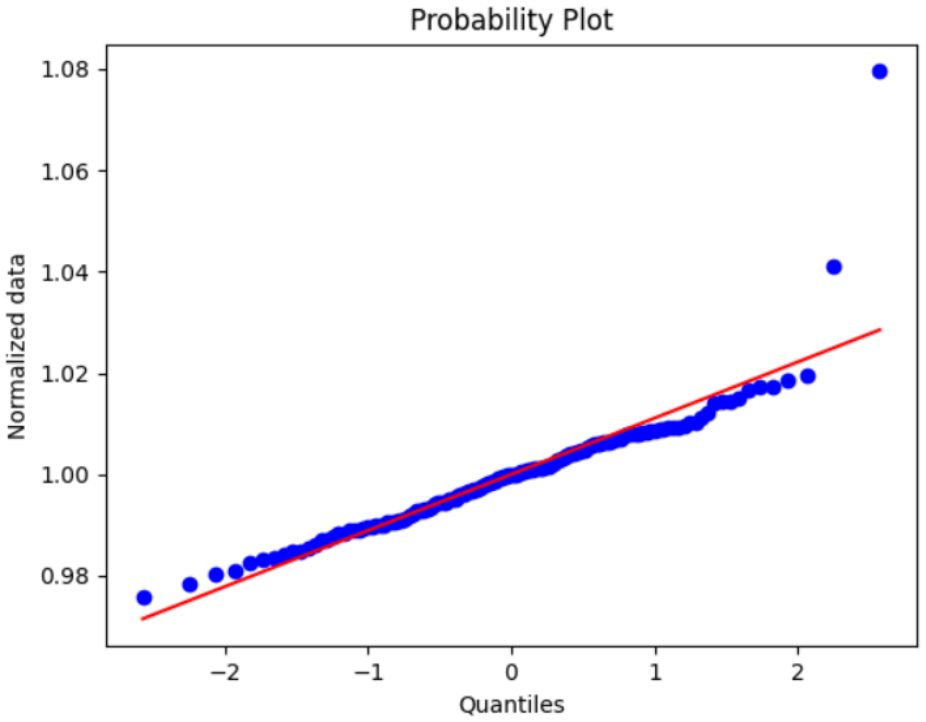
Figuur 42: Flowchart resultaten testplan



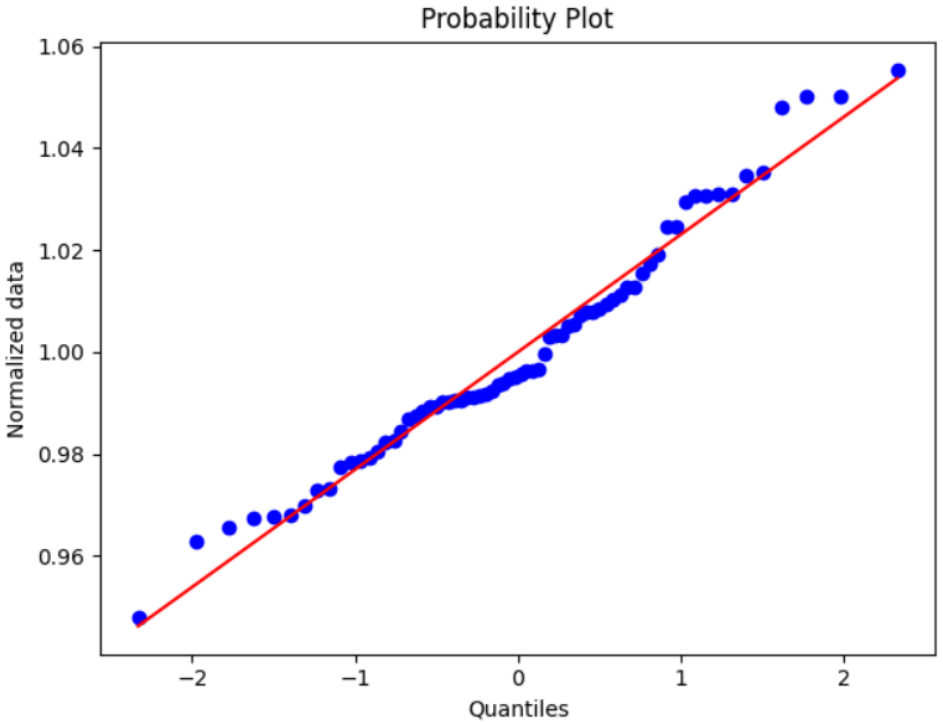
Figuur 43: Flowchart resultaten testplan

Bijlage H: QQ-plots

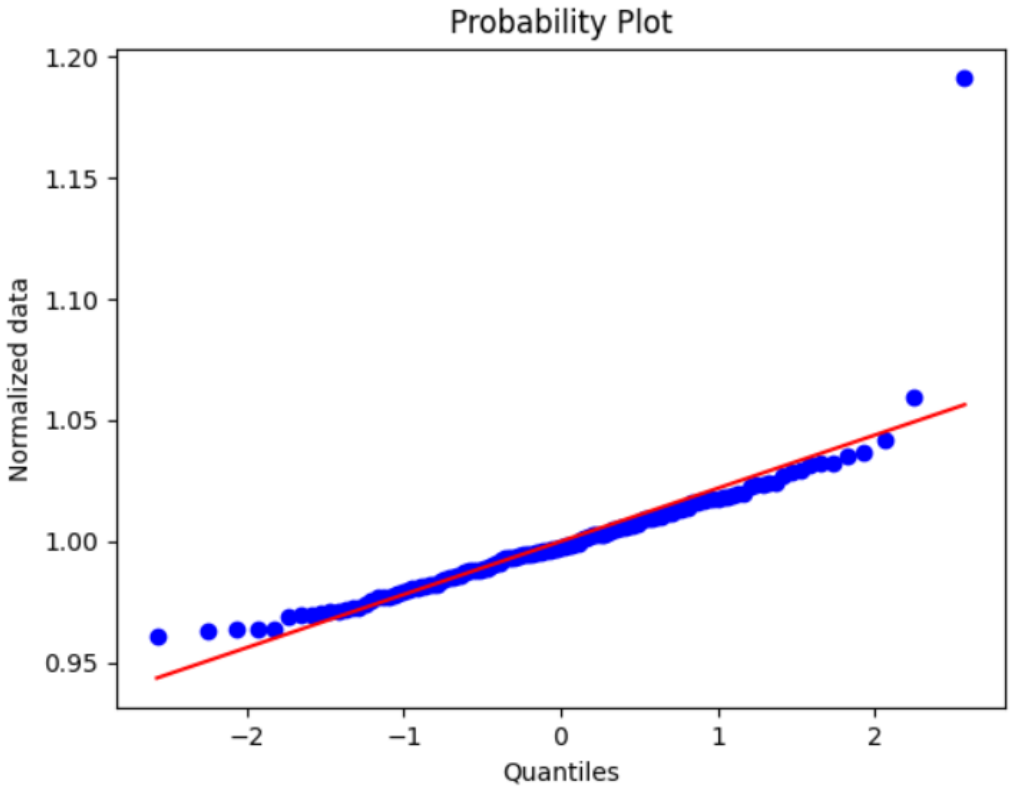
In bijlage H: QQ-plots zijn de QQ-plots opgenomen, afkomstig van de "*findAndScroll*" test. Hieronder is het resultaat weergegeven van de on-premise omgeving en van de zes cloud instances.



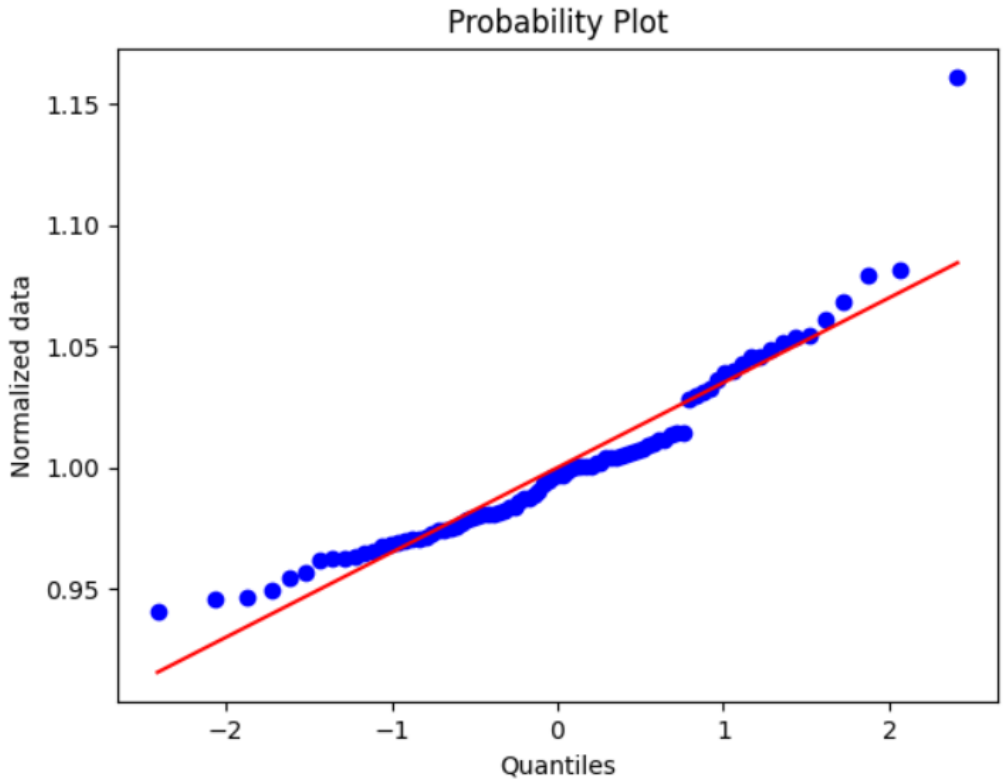
Figuur 44: QQ-plot afkomstig van de test "findAndScroll" uit de on-premise omgeving



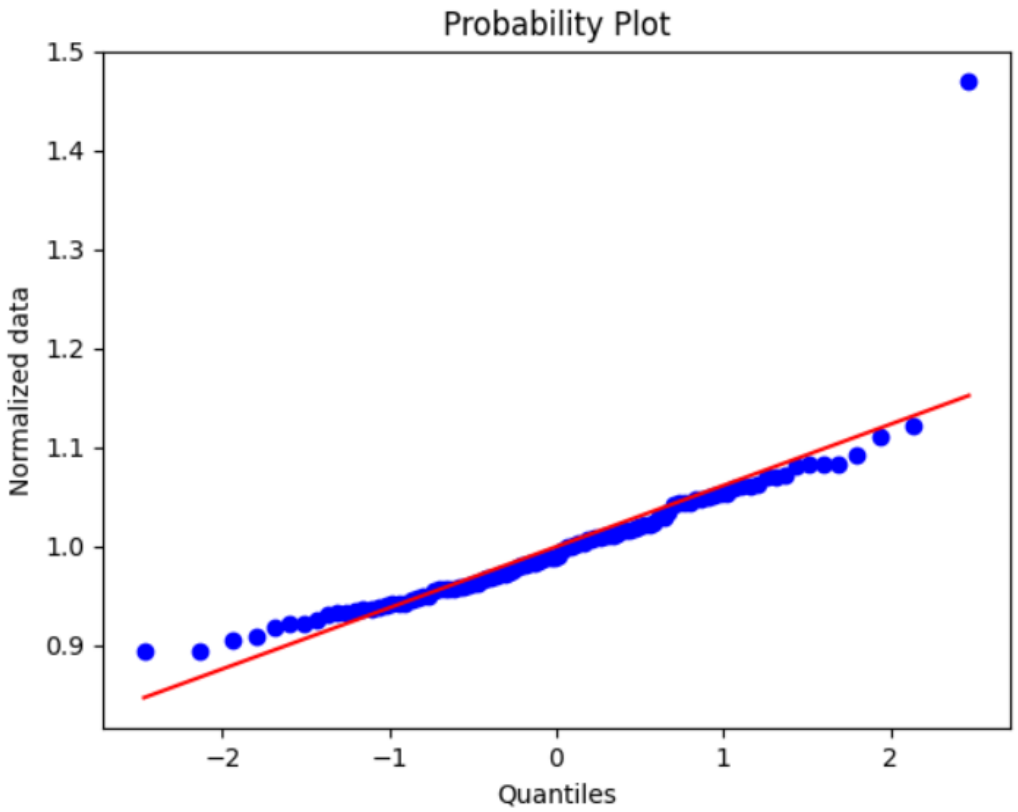
Figuur 45: QQ-plot afkomstig van de test "findAndScroll" uit de standard Azure instance



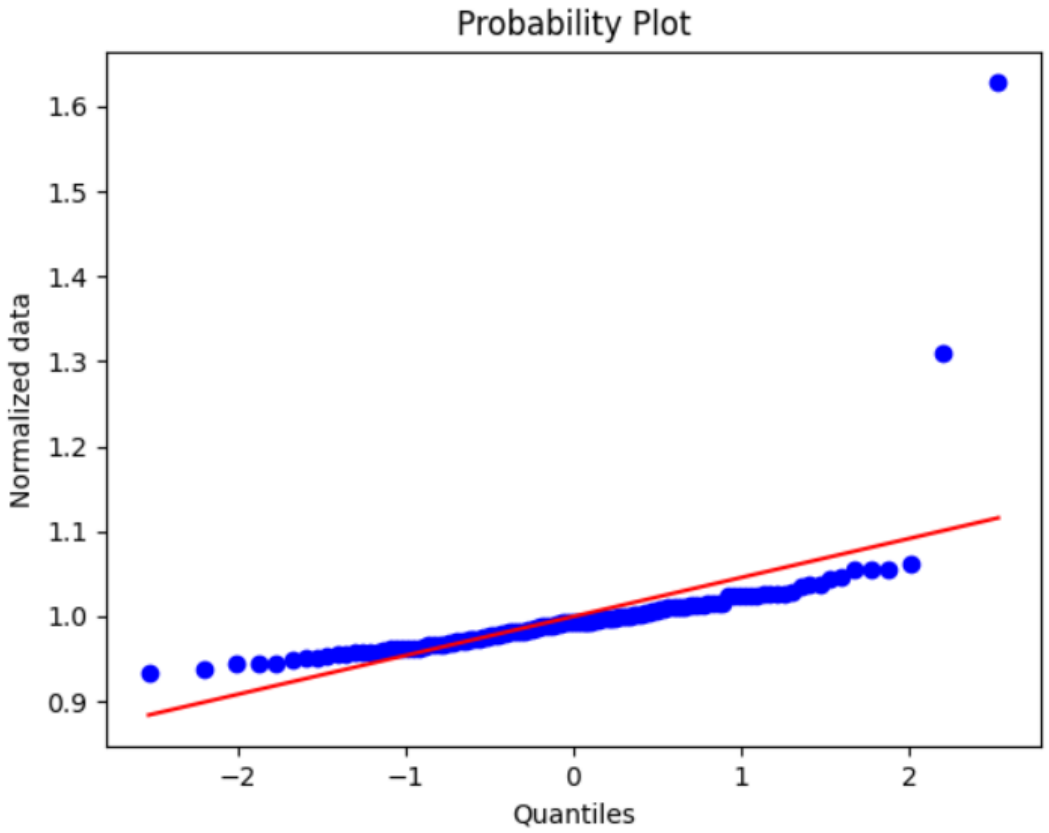
Figuur 46: QQ-plot afkomstig van de test "findAndScrol"l uit de compute Azure instance



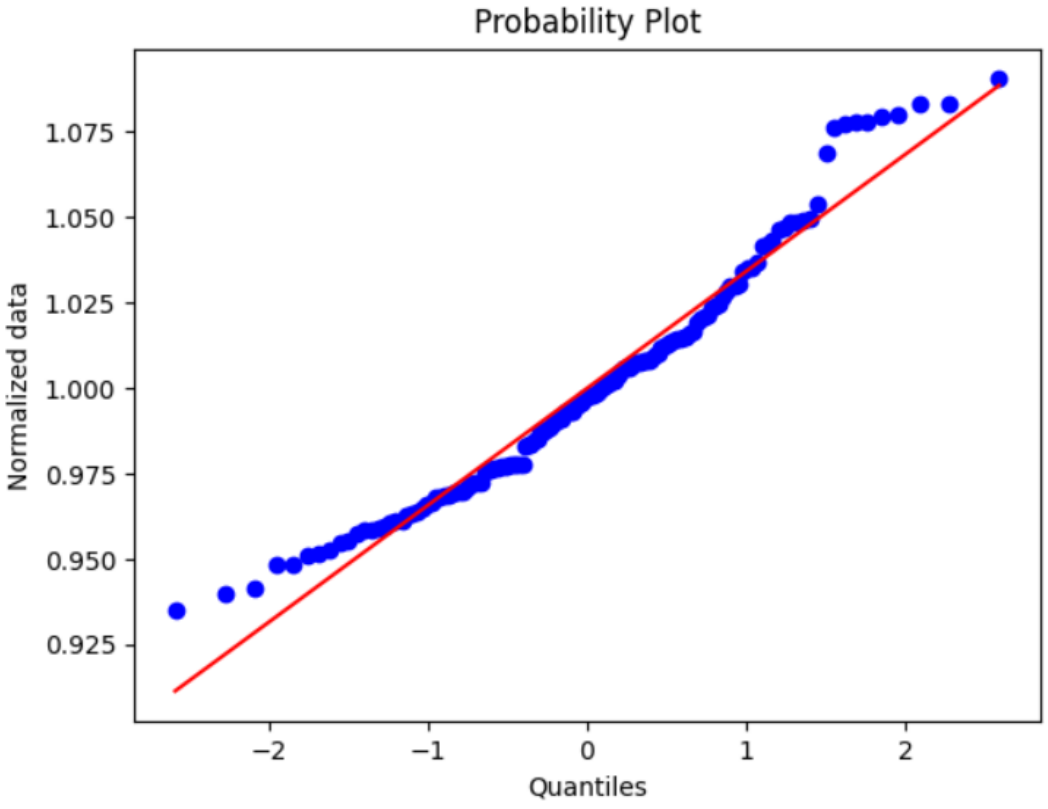
Figuur 47: QQ-plot afkomstig van de test "findAndScroll" uit de standaard AWS instance



Figuur 48: QQ-plot afkomstig van de test "findAndScroll" uit de compute AWS instance



Figuur 49: QQ-plot afkomstig van de test "findAndScroll" uit de standaard GCE instance



Figuur 50: QQ-plot afkomstig van de test "findAndScroll" uit de compute GCE instance

Bijlage I: Onderzoek A

Performance Testing in the cloud. How Bad is it Really?

Een onderzoek uit 2018 (Laaber et al., 2018) analyseerde performance testing in de cloud. Hierin is het onderzoek uitgevoerd op basis van twee deelvragen. Die worden hieronder besproken in combinatie met de onderzoeksopzet, conclusies, weerlegging en impact op huidig onderzoek.

A. Hoe variabel zijn benchmark resultaten in verschillende cloud omgevingen?

B. Hoe groot moet een regressie zijn om detecteerbaar te worden in een cloud omgeving? Welke statistische methoden bieden een zekere manier voor het detecteren van regressie?

Onderzoeksopzet

Het doel van het onderzoek is om vast te leggen hoe betrouwbaar performance testing is in de cloud. Om dit doel te bereiken worden vier open-source programma's uitgevoerd in de cloud, waarbij twee ontwikkeld in Java en twee in Go. Deze programma's worden ten uitvoer gebracht op dezelfde en verschillende cloud servers. Daarnaast wordt dit herhaald voor verschillende cloud providers, namelijk AWS, Microsoft Azure en Google Compute Engine (GCE). De open-source programma's maken gebruik van benchmarking, waardoor de open-source programma's in staat de performance van zichzelf te meten. De resultaten worden vergeleken op basis van variabiliteit en detecteerbaarheid van de regressie.

Cloud providers verschaffen meerdere type servers aan de gebruiker die elk zijn geoptimaliseerd voor verschillende doeleinden. Deze type worden in het cloud domein aangeduid als families en brengen elk andere voor- en nadelen. AWS biedt onder andere families die geoptimaliseerd zijn voor opslag, geheugen en snelheid.[7] Binnen deze families bestaan meerdere instances, zoals te zien op figuur 51. Deze instances bieden andere configuratie mogelijkheden aan binnen een familie. Tijdens het onderzoek wordt benchmarking uitgevoerd op verschillende families, met als doel de meest optimale te selecteren voor performance testing.

Een IBM Bluemix server is gebruikt als referentie. Deze server bevat geen virtualisatie software en wordt niet gebruikt door andere consumenten, waardoor deze vergelijkbaar is met een lokale server. De resultaten afkomstig van de cloud providers worden vergeleken met de IBM Bluemix server.

De regressie wordt tijdens het onderzoek gemeten door middel van de Wilcoxon rank-sum methode. Deze statistische methode wordt gebruikt om te om onderzoeken of twee verschillende steekproeven, afkomstig zijn van populaties met eenzelfde verdeling.[22]

In figuur 51 zijn de families voor en bijbehorende instance types (hier aangeduid als generatie) weergegeven. Elke familie biedt verschillende instances aan die geoptimaliseerd zijn voor een specifiek doel, zoals hevig CPU of GPU gebruik.

Family	Generation	Smallest	Largest
General Purpose (GP)	t2	t2.micro	t2.2xlarge
	m4	m4.large	m4.16xlarge
	m3	m3.medium	m3.2xlarge
Compute Optimized	c4	c4.large	c4.8xlarge
	c3	c3.large	c3.8xlarge
Memory Optimized	r3	r3.large	r3.8xlarge
	r4	r4.large	r4.16xlarge
	x1	x1.16xlarge	x1.32xlarge
Storage Optimized	i2	i2.xlarge	i2.8xlarge
	d2	d2.2xlarge	d2.8xlarge
Accelerated Computing	g2	g2.2xlarge	g2.8xlarge
	p2	p2.xlarge	P2.16xlarge

Figuur 51: Verschillende type families en instances (hier aangeduid als generatie) [8]

Conclusies

Verschil in variabiliteit per open-source programma

De variabiliteit van performance testresultaten kan beïnvloed op twee manieren. Ten eerste resulteert een korte executietijd van tientallen nanoseconden in onnauwkeurige metingen. Ten tweede kan de benchmark kwaliteit van het open-source programma invloed hebben op de variabiliteit.

IO-intensieve handelingen

Onderzoek (Leitner & Cito, 2016) concludeert dat handelingen die hevig gebruik maken van IO-operaties veel performance instabiliteit hebben, mede dankzij andere gebruikers die gebruik maken van de onderliggende hardware.

Prestatie families gelijk

Verschillende families van dezelfde cloud provider presteren nagenoeg identiek. Dit kan resulteren uit het feit dat de open-source programmatuur niet gebruik maakte van de volledige onderliggende computerkracht.

IBM Bluemix vs. cloud providers

De in het onderzoek opgesteld hypothese dat de IBM Bluemix machine substantieel beter presteert dan de cloud providers kan niet worden ondersteund. Wanneer de cloud servers

hoge variabiliteit tonen, is dat ook het geval bij de IBM Bluemix machine. Daarnaast presteert AWS substantieel beter dan Microsoft Azure en GCE en bracht het regelmatig betere resultaten voort, vergeleken met de IBM Bluemix machine.

Regressie metingen

Tijdens het onderzoek is getoetst of er met behulp van de Wilcoxon rank-sum methode performance regressie gemeten kan worden. Hieruit blijkt dat deze methode niet geschikt is voor het meten van performance regressie. Na de uitvoer van verschillende testen bleek dat een strategie waarbij medians vergeleken worden voldoende is om regressie van 1% tot 10% aan te tonen.

Weerlegging

Publieke en private cloud

Het besproken onderzoek analyseerde performance testing in relatie tot de publieke cloud. Hierdoor zijn de conclusies niet toepasbaar op de private cloud.

Verschillende ontwikkeltalen

De performance regressietesten zijn uitgevoerd door open-source programma's ontwikkeld in Java en Go. De huidige test runner bij Fonto is gecreëerd in Javascript, waardoor de resultaten in de cloud, vergeleken met het onderzoek, afwijkend kunnen zijn.

Gedateerd onderzoek

Sinds het onderzoek gepubliceerd is kunnen er veranderingen zijn geïmplementeerd in de cloud omgevingen die delen van het onderzoek dateren.

Ander systeem

De complexiteit van de test runner ligt vele malen hoger in vergelijking met de gebruikte open-source programma's. Hierdoor kunnen afwijkende resultaten waargenomen worden.

Bijlage J: Onderzoek B

Patterns in the Chaos

Een onderzoek uit 2016 (Leitner & Cito, 2016) toets meerdere hypotheses die gerelateerd zijn aan de voorspelbaarheid van performance in de cloud. Hierin worden onder andere de performance variatie en factoren die invloed hebben op de performance besproken. Daarnaast worden verschillende instances met elkaar vergeleken. De opgestelde hypotheses in het onderzoek komen voort uit gerelateerde literatuur en worden door middel van experimenten gevalideerd. Het doel van genoemd onderzoek is '... to identify the fundamental rules, patterns and mechanisms underlying performance variations of IaaS-based public cloud systems'.

Conclusies

De conclusies afkomstig uit het onderzoek (Leitner & Cito, 2016) die van toepassing zijn voor het huidige onderzoek zijn hieronder beschreven.

Substantieel verschil tussen cloud providers

De performance stabiliteit verschilt per cloud provider. Hierbij bleken AWS en Microsoft Azure stabiel te presenteren, waarbij GCE en IBM Softlayer resultaten leverden met een hogere hoeveelheid ruis.

De impact van de regio en dag op de instance performance

De stabiliteit van de instances wordt beïnvloed door de regio waarin deze gealloceerd is. Daarnaast heeft de specifieke dag van de week of tijdsdeel geen impact op de performance.

De keuze tussen instances is niet triviaal

Het onderzoek concludeert dat de verschillen tussen instances nihil zijn en dat duurdere instances niet altijd beter presteren. Ditzelfde geldt voor instances die geoptimaliseerd zijn voor onder andere intensief CPU of GPU gebruik. Deze presteren nagenoeg hetzelfde als de standaard instances.

Weerlegging

Aangezien het onderzoek (Laaber et al., 2018) een soortgelijke aanpak gebruikt als het onderzoek beschreven in bijlage I: Onderzoek A, zijn die weerleggingen ook toepasbaar op het onderzoek van Leitner & Cito uit 2016.

Bijlage K: Hoe werkt C4

Het C4 model beschrijft de software architectuur in vier niveaus, waarbij elk van deze niveaus de software architectuur in meer detail beschrijft. Daarnaast maakt C4 gebruik van verschillende abstracties om verschillende onderdelen aan te duiden binnen de niveaus. Deze abstracties worden hieronder toegelicht.

Begrippen

Software System

Een software system is de hoogste abstractie binnen het C4 model en beschrijft een softwaresysteem die van waarde is voor de gebruikers.[45]

Container

Binnen een software systeem worden één of meerdere containers gedefinieerd. Een container binnen C4 beschrijft een applicatie of data storage. Een container kan onder andere een mobiele applicatie, database, Python script of een server-side web applicatie zijn. Een container wordt in de meeste gevallen ten uitvoer gebracht in zijn eigen omgeving zoals op een server of als Docker container.[45]

Component

Binnen een container worden meerdere componenten gedefinieerd. Een component vertegenwoordigt een groep van functionaliteiten, die werkzaam zijn binnen een container. Hierbij, in tegenstelling tot een container, kan een component niet op zichzelf ten uitvoer worden gebracht. Ter voorbeeld, een selectie van Java of C# klassen die geïmplementeerd worden achter een interface wordt gedefinieerd als een component.[45]

Code

Binnen een component wordt de code gedefinieerd. Hierbij worden onder andere de klassen, functies en interfaces weergeven. Deze onderdelen kunnen automatisch gegenereerd worden door de gebruikte integrated development environment (IDE) of kunnen gemodelleerd worden door middel van UML.[45] [46]

Person

Naast bovengenoemde abstracties bevat het C4 model ook persons. Een person geeft een menselijke entiteit aan die gebruik maakt van één of meerdere softwaresystemen.[45]

Figuur 53: Notatie binnen C4 [45]

Diagrammen

De bovengenoemde abstracties die hierboven toegelicht zijn, worden toegepast in het proces van de C4 ontwerpmethodiek. Het proces dat wordt doorlopen bij C4 is hieronder toegelicht. Daarnaast is het proces van C4 visueel weergegeven in Bijlage L: Proces C4 visueel weergegeven.

Niveau 1: System Context diagram

Allereerst wordt het System Context Diagram opgezet, waarbij de hoogste abstractie binnen het C4 model wordt weergegeven. Hierbij worden de verschillende softwaresystemen, personen en afhankelijke softwaresystemen in kaart gebracht. Het doel van een System Context diagram is om de samenhang tussen gebruikers en softwaresystemen weer te geven, zonder hierbij technische details te definiëren.[45]

Niveau 2: Container diagram

In een container diagram wordt een van de softwaresystemen, die gedefinieerd zijn in het system context diagram, tot op meer detail uitgewerkt. Hierbij wordt het desbetreffende softwaresysteem opgesplitst in één of meerdere containers die ondere andere een mobiele applicatie, database of web applicatie kunnen zijn. Het doel van een container diagram is om de afhankelijkheden tussen containers te tonen, om weer te geven met welk van de containers de gebruiker interacteert, om de communicatie methode tussen containers weer te geven en om de communicatie tussen containers en andere software systemen die gedefinieerd staan in het system context diagram te tonen.[45]

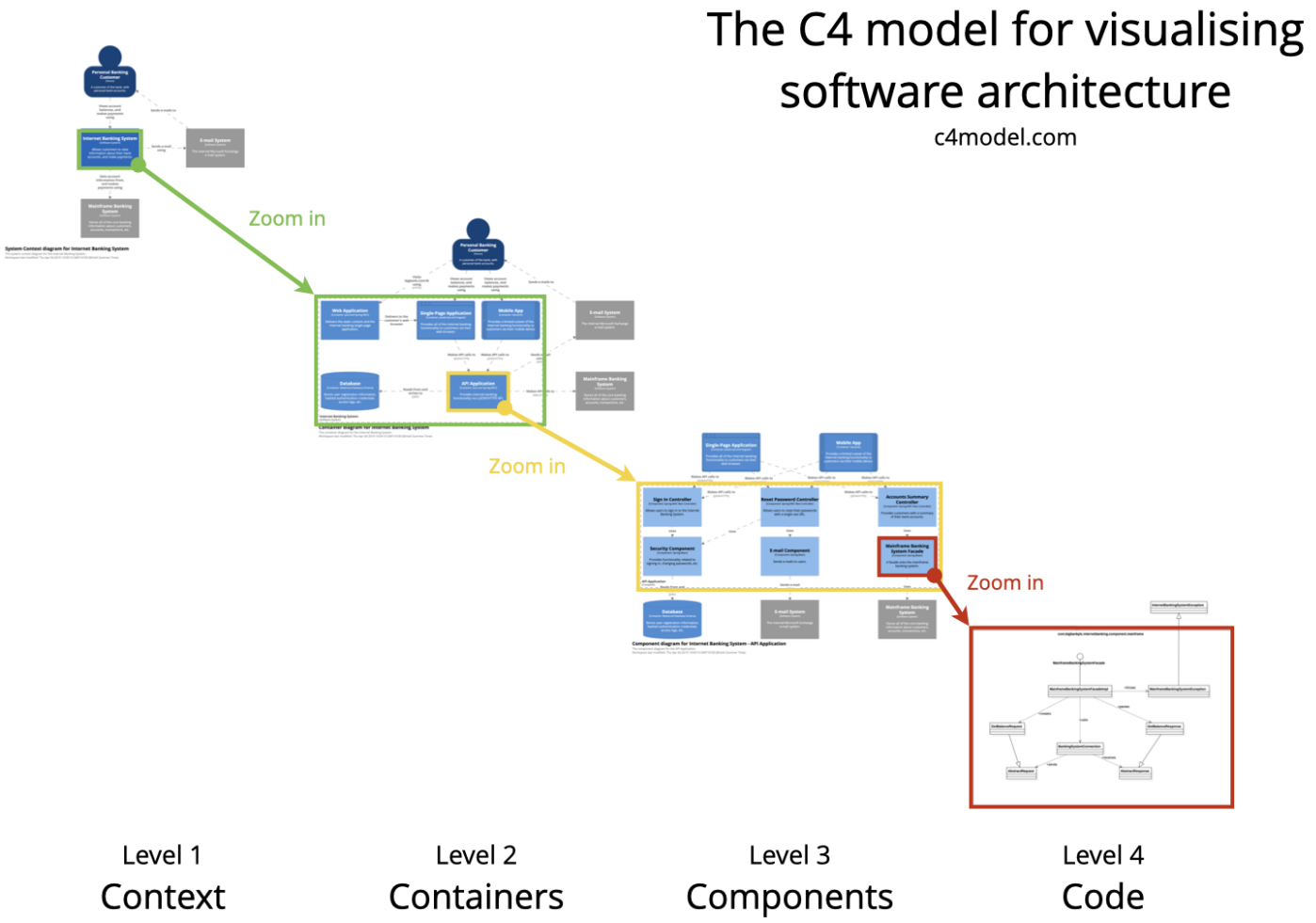
Niveau 3: Component diagram

Binnen een component diagram wordt een van de containers, die gedefinieerd zijn in het container diagram, tot op meer detail uitgewerkt. De container wordt hierbij onderverdeeld in componenten, die elk een groep van functionaliteiten representeren. Het doel van een component diagram is om weer te geven hoe een container is opgebouwd en welke technologieën daarbinnen geutiliseerd worden.[45]

Niveau 4: Code diagram

Tot slot het laagste abstractie niveau, genaamd code. In een code diagram worden de klassen, interfaces en functionaliteiten weergegeven van één component. De code diagrammen kunnen, zoals eerder vermeld, gegenereerd worden door middel van de IDE of opgesteld worden in UML. Daarnaast beveelt de website van de C4 ontwerpmethodiek aan om dit niveau niet te beschrijven, omdat het code diagram in de meeste gevallen te gedetailleerd wordt en daarnaast regelmatig veranderd moet worden. [45]

Bijlage L: Proces C4 visueel weergegeven



Figuur 52: Visualisatie van het C4 proces [45]