HO
GENT

Faculteit Bedrijf en Organisatie

Best practices om continuous integration en continuous delivery uit te rollen in een SAPUI5 webapplicatie op SAP Cloud Platform

Renaat Haleydt

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

Promotor:
Harm De Weirdt
Co-promotor:
Pieter-Jan Deraedt

Instelling: Amista

Academiejaar: 2018-2019

Tweede examenperiode

Faculteit Bedrijf en Organisatie

Best practices om continuous integration en continuous delivery uit te rollen in een SAPUI5 webapplicatie op SAP Cloud Platform

Renaat Haleydt

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

Promotor:
Harm De Weirdt
Co-promotor:
Pieter-Jan Deraedt

Instelling: Amista

Academiejaar: 2018-2019

Tweede examenperiode

Woord vooraf

Samenvatting

Inhoudsopgave

Lijst van tabellen

1. Inleiding

IT-bedrijven willen hun klanten steeds sneller te hulp kunnen schieten en betere software leveren. Vandaag de dag worstelen ze nog te vaak met deadlines die verstrijken en krijgen ze te maken met problemen tijdens de oplevering van software. Gelukkig is er doorheen de jaren al veel progressie gemaakt door het toepassen van principes zoals Agile en DevOps. Agile leidt tot een betere samenwerking tussen business en IT. DevOps gaat dan weer een stapje verder en een mindset om de samenwerking tussen alle afdelingen binnen een IT-bedrijf vlotter te laten verlopen. Een Continuous Integration en Continuous Delivery (CI/CD) pipeline opzetten is een vast onderdeel van DevOps en zou de werking van een bedrijf ten goede komen. Wanneer code, die naar de 'Source Control Repository' gepusht wordt, automatisch getest wordt of deze voldoet aan alle kwaliteitseisen en er feedback gegeven wordt over de kwaliteit van de toegevoegde code. Dan spreekt men over Continuous Integration. Er kan van een Continuous Delivery omgeving gesproken worden als de code, die naar de 'Source Control Repository' is gepusht, automatisch getest wordt of deze voldoet om naar Quality te brengen. Een CI/CD pipeline zorgt voor de automatisatie van testen, builds en deployment en heeft als grote voordelen het sneller dorvoeren van wijzigingen en het minimaliseren van 'downtime' van de applicatie.

Uit onderzoek - uitgevoerd door DZone - blijkt dat DevOps steeds meer ingeburgerd raakt in de bedrijfscultuur (Baker, 2019). In 2019 werkte 48% van de ondervraagden binnen een Operations team mee aan een Continuous Delivery pipeline, een stijging van 7% in vergelijking met 2018. De stijging is te danken aan het management, dat meer gelooft in de mindset van DevOps. Van de 527 ondervraagde technologiespecialisten heeft 54% van hen de steun van het management als het gaat over DevOps. Dit wil zeggen dat we op goede weg zijn, maar er nog werk aan de winkel is. Als we verder naar het onderzoek kijken, blijkt dat er een duidelijk verschil is tussen de implementatie van DevOps en een CI/CD pipeline. Bij 31% is er een Continuous Integration pipeline aanwezig in elk project,

Figuur 1.1: Deel van DZone's survey (Baker, 2019)

terwijl 33% zegt een CI pipeline te gebruiken in enkele projecten. Slechts 14% van de ondervraagden bevestigt dat ze een Continuous Delivery pipeline hebben voor elk project, dat is zelfs een daling ten opzichte van 2018. 28% gebruikt voor een aantal projecten een Continuous Delivery pipeline.

Hieruit kan men besluiten dat CI meer ingeburgerd is dan CD. In 58% van de gevallen vloeien Continuous Integration processen niet door naar Continuous Delivery. De belangrijkste redenen voor deze 'gap' zijn de moeilijkheid van de configuration setup, de moeilijkheden van user acceptance testing (de laatste fase binnen software testing waarbij echte gebruikers de software testen) en het automatiseren van die testen.
Volgens 47% van de 527 technologiespecialisten was dit te wijten aan de bedrijfscultuur die niet klaar is om Continuous Delivery toe te passen, in 2018 was dit nog 45%.

Zoals u kan zien op Figuur1.1 is er nog heel wat werk aan de winkel, maar is er toch al verbetering vastgesteld ten opzichte van vorige jaren. Dit bewijst ook dat een CI/CD pipeline en DevOps in het algemeen niet zo makkelijk op te zetten zijn. Technologisch is het vandaag niet zo bijster moeilijk om een pipeline op te zetten. Maar het veranderen van de mindset binnen een bedrijf, daar knelt vaak het schoentje.
Deze scriptie gaat echter niet al te diep in op het toepassen van de DevOps cultuur binnen een bedrijf, maar zal vooral het technische luik onder de loep nemen. Het bedrijf Amista zou graag een gepersonaliseerde vergelijking en handleiding hebben hoe een CI/CD pipeline op te starten.

Amista is een groeiend bedrijf binnen IT consultancy. Het is een dochterbedrijf van Boutique dat zich toespitst op SAP. Amista bestaat nog maar 5 jaar en ze hebben al 60 personen in dienst. Ze mogen Infrabel, Alcopa, Danone, AG Real Estate en nog andere grote namen tot hun cliënteel rekenen, zoals ook op hun site te lezen valt[1]. Ze zijn in twee landen actief, Frankrijk en België, maar werken ook samen met mensen uit India. Amista heeft zich verdiept in Sales, Marketing en Service Management binnen bedrijven. Ze helpen hun klanten op gebied van Innovation, Integration en Digital Learning. De missie van Amista is om samen met hun klanten, innovatieve en kwalitatieve oplossingen aan te

[1] https://www.amista.be

bieden met behulp van het volledige gamma dat SAP te bieden heeft. Amista wil graag de wensen van hun klanten zo goed mogelijk proberen te vervullen, daarom proberen ze zelf innovatief uit de hoek te komen. Ze spelen met het idee om een CI/CD pipeline op te zetten voor software development om zo hun klanten een nog betere service te bieden.

Om een Continuous Integration en Continuous Delivery pipeline op te stellen zijn er vandaag veel tools beschikbaar. Deze thesis zal verschillende build schedulers vergelijken op basis van de samenwerking met SAPUI5, SAP Cloud Platform, Git en Bitbucket. Er wordt vergeleken of de build scheduler past binnen een CI/CD pipeline, automated tests via Karma kan uitvoeren en bovendien wordt er gekeken naar gebruiksgemak en security. Aan de hand van de 'winnende' build scheduler wordt er een handleiding beschreven om een pipeline op te zetten binnen een SAPUI5 webapplicatie dat verbonden is met SAP HANA en gehost wordt op SAP Cloud Platform.

1.1 Probleemstelling

Amista heeft enkele klanten waar ze SAPUI5 webapplicaties voor maken, dit wordt vaak gecombineerd met een SAP HANA database dat gebruik maakt van Node.js en wordt allemaal gehost op SAP Cloud Platform. Elk bedrijf wil het beste voor zijn klanten. Voor een software consultancy bedrijf, zoals Amista, staat de service en het project dat de klant voor ogen heeft centraal en zullen ze er alles aan doen om de noden en wensen van hun klanten te vervullen.
Bij de oplevering van een project loopt het vaak mis, de deadline wordt vaak overschreden wat stress en problemen met zich meebrengt. Het gebeurt telkens weer opnieuw dat de klant een wijziging wil doorvoeren, wat de scope en planning in de war stuurt. Deze scenario's kunnen vermeden worden door een Continuous Integration en Continuous Delivery pipeline te gebruiken. Dankzij een CI/CD pipeline kan het development team sneller wijzigingen doorvoeren en krijgen ze sneller feedback van de klant. Aan de hand van een voorbeeld wordt bovenstaande verduidelijkt.

Amista heeft Appel als klant. Deze klant had in het verleden wat moeilijkheden met het opleveren van nieuwe projecten en wijzigen van bestaande projecten. Daarom besloten ze om een policy toe te passen waarbij de opleveringsdata gebundeld werden en om de twee weken een release-dag te voorzien. Dit maakt het niet mogelijk om een wijziging door te voeren tussen twee datums van oplevering. Er moet telkens gewacht worden tot de volgende opleverdatum om een verandering door te voeren.
Appel is een internationale speler binnen zijn industrie en heeft vestigingen over heel de wereld. In één van de landen is het verplicht dat grote bedrijven de belastingen online invullen. Wanneer dit niet gebeurt hangt het bedrijf grote boetes boven het hoofd. De overheid voorziet een speciaal certificaat voor het invullen van de belastingen. De tool voor de belastingaangifte werd gehost op SAP Cloud Platform, waar Amista verantwoordelijk voor is. Een tijdje geleden werd het certificaat vernieuwd op SAP Cloud Platform tijdens een release-dag, maar de build loopt mis. Er falen verschillende processen en de applicatie loopt vast. Ze zoeken het probleem en vinden dat er een parameter in de backend moet vernieuwd worden. Hier was Amista echter niet verantwoordelijk voor en wist bijgevolg

ook niet van het bestaan van deze parameter. Gezien de policy van Appel -dat er om de twee weken een release van de wijzigingen mag doorgevoerd worden- is het bedrijf niet voorbereid op onverwachte problemen. Uiteindelijk hebben ze met veel moeite iemand kunnen bereiken die de parameter in de backend kon aanpassen en kon zorgen voor een nieuwe release. Zo kon, gepaard met heel veel stress, vermeden worden dat Appel een boete kreeg.
Met een Continuous Integration en Continuous Delivery pipeline zou dit euvel in no-time opgelost kunnen worden. Met voldoende en goede testen zou de pipeline aangegeven hebben dat er een probleem was en zal de wijziging nooit productie halen.
(Bovenstaand voorbeeld bevat fictieve namen).

Om een CI/CD pipeline op te zetten binnen de hierboven beschreven omgeving is er nog niet veel informatie terug te vinden. Het probleem ligt hem in de combinatie van specifieke tools die gebruikt worden. Om te weten welke build scheduler het beste bij deze omgeving zou passen moeten er specifieke zaken onderzocht en vergeleken worden. Op internet is niet veel informatie terug te vinden welke build scheduler het best gebruikt wordt met een SAPUI5 applicatie, gehost op SAP Cloud Platform. Daarom wil Amista heel graag een gepersonaliseerde vergelijking en handleiding om een Continuous Integration en Continuous Delivery pipeline op te zetten, zodat ze hun klanten beter kunnen helpen.

1.2 Onderzoeksvraag

- Wat zijn de voor- en nadelen van een CI/CD pipeline in het algemeen en specifiek voor Amista?
- Is het mogelijk om een Continuous Integration en Continuous Delivery pipeline te implementeren voor de ontwikkeling van een SAPUI5 applicatie, gecombineerd met een SAP HANA database op SAP Cloud Platform?
- Welke tools moeten we gebruiken om een CI/CD pipeline te implementeren als we vergelijken op de samenwerking met SAPUI5, SAP Cloud Platform, Git en Bitbucket. Welke build scheduler past binnen een CI/CD pipeline, kan automated tests via Karma uitvoeren en is bovendien het veiligst en makkelijst in gebruik?
- Hoe kunnen we deze implementatie tot een succes brengen?

1.3 Onderzoeksdoelstelling

Het hoofddoel van deze thesis is een proof-of-concept uit te zetten over hoe een CI/CD pipeline te gebruiken in de dagelijkse werking van Amista. Deze thesis gaat daarnaast ook op zoek naar de beste tools om de pipeline op te bouwen rekening houdend met de samenwerking tussen SAPUI5, SAP Cloud Platform, Git en Bitbucket. Er wordt vergeleken of de build scheduler past binnen een CI/CD pipeline, automated tests via Karma kan uitvoeren en bovendien wordt er gekeken naar gebruiksgemak en security.

1.4 Opzet van deze bachelorproef

De rest van deze bachelorproef is als volgt opgebouwd:

In Hoofdstuk 2 wordt er dieper ingegaan op de domeinen: Continuous Integration, Continuous Delivery, Continuous Deployment, DevOps en Automated Testing omdat deze toch een dominantie rol spelen binnen deze thesis.

De waarom-vraag is minstens even belangrijk als de hoe-vraag. Concreet: wat zijn de voor- en nadelen van een CI/CD pipeline te integreren in het algemeen en specifiek voor Amista? Hoe ziet zo'n CI/CD pipeline eruit? Een antwoord op deze vragen kan je in Hoofdstuk 3 terug vinden.

SAP en de tools die in deze thesis worden gebruikt worden aan de hand van de gevonden literatuur beschreven in hoofdstuk 4.

In Hoofdstuk 5 wordt de methodologie toegelicht en worden de gebruikte onderzoekstechnieken besproken om een antwoord te kunnen formuleren op de onderzoeksvragen. Dit door in grote lijnen een vergelijking te maken tussen de build schedulers die op de markt aanwezig zijn. Ook de set-up van de omgeving wordt uitvoerig besproken.

In Hoofdstuk 6 wordt er een gepersonaliseerde handleiding beschreven om met de beste build scheduler een CI/CD pipeline te implementeren.

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

2. Continuous Integration, Continuous Deliver

Bedrijven zijn constant op zoek naar betere en snellere resultaten, maar in de wereld van IT is het vandaag soms nog lang wachten voor een wijziging effectief doorgevoerd wordt. Vaak wordt een enorme hoeveelheid software op het einde van een sprint opgeleverd, stukken code van verschillende developers moet op het laatste nippertje nog aan elkaar geplakt worden om toch een werkende applicatie te bezorgen, wat soms problemen met zich meebrengt. Met een vrij recente software development methode - Continuous Integration en Continuous Delivery genaamd - wil men hier verandering in brengen. Er wordt vaak gesproken van een CI/CD pipeline als het gaat over Continuous Integration en Continuous Delivery, maar er is nog een derde speler dat men kan invoeren: Continuous Deployment. Samen vormen zij de 3 musketiers om software projecten een grotere slaagkans te geven. Om een duidelijker beeld te scheppen van een CI/CD pipeline zal dit hoofdstuk eerst uitleg verschaffen over DevOps, omdat dit de allesomvattende term is waar de 3 musketiers deel van uitmaken. Er wordt ook dieper ingegaan op Continuous Integration, Continuous Delivery, Continuous Deployment en tot slot komt Automated Testing aan bod.

2.1 DevOps

DevOps is een samentrekking van development en operations en is een welbekend begrip binnen de informatica wereld. Het heeft als doel om de 'state of mind' binnen een bedrijf te veranderen zodat alle lagen/departementen vlotter samenwerken. Het is een praktische 'gids' dat bedrijven kunnen hanteren om de communicatie tussen developers en systeembeheerders vlotter te laten verlopen. Deze twee verschillende lagen in een bedrijf willen namelijk hetzelfde: zo snel mogelijk, kwaliteitsvolle software opleveren. DevOps is gebaseerd op Agile development, maar gaat verder dan dat. Het gaat dieper

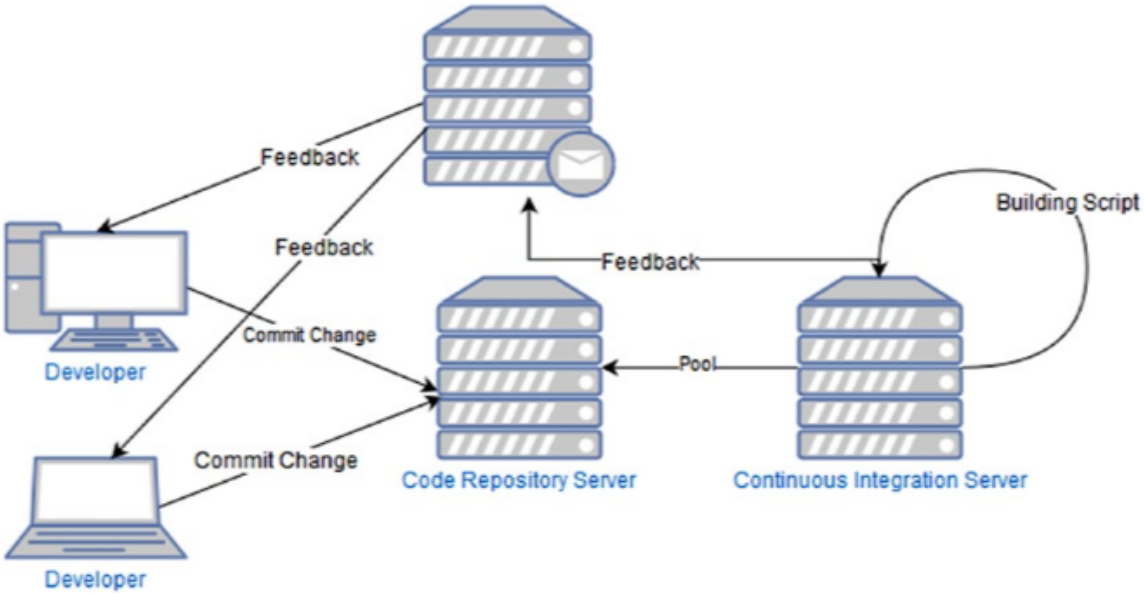
in op automatisatie, integratie, samenwerking en communicatie. Continuous Integration, Delivery en Deployment zijn kenmerkend voor DevOps, omdat het mee inzet op snellere oplevering van kwaliteitsvolle software. (Riti, 2018)

2.2 Continuous Integration

Dit is de eerste laag van de pipeline die zelf uit verschillende stappen bestaat. Eerst moet de geschreven code verstuurd worden naar een 'repository management server' aan de hand van een commit. De 'source code' wordt automatisch gebuild en bekeken of ze slaagt voor de automatisch gestarte testen. De focus bij deze stap ligt bij de teamleden (Fowler, 2006), van hen wordt verwacht dat ze - op regelmatige basis - hun code pushen (integreren met de master applicatie). Een goede samenwerking tussen de verschillende leden van het development team is broodnodig om tot het gewenste eindresultaat te komen. Het doel van een Continuous Integration is om de integratie feilloos te laten verlopen wanneer men software ontwikkelt en geen functionaliteiten verliest na een merge (Riti, 2018).

Bovenstaande uitleg is makkelijker te begrijpen aan de hand van een voorbeeld. Op Figuur 2.1 is een grafische voorstelling van onderstaande uitleg terug te vinden. De developer maakt een wijziging en commit de code naar de repository die te vinden is op het source-control systeem, zie 3.2.1 voor verdere uitleg. De Continuous Integration server (CI server) krijgt bericht dat er code is toegevoegd, haalt de laatst toegevoegde code op en laat de geschreven unit testen runnen. Wanneer alle testen slagen zal de CI server de code compilen en feedback bezorgen aan de developer. In dit voorbeeld is er gebruik gemaakt van een externe mail server om die feedback te verzenden. Deze stappen gebeuren telkens er nieuwe code naar de repository gestuurd wordt.

De uitleg die hierboven te lezen valt, is een best-practice hoe het zou moeten gebeuren. Er zijn echter enkele zaken die wat extra uitleg kunnen gebruiken. De frequentie en hoeveelheid code zijn belangrijke zaken waar de developer rekening mee moet houden bij een CI pipeline. Volgens Fowler (2006) is het de plicht van een developer om minstens 1 maal per dag een commit uit te voeren, telkens wanneer hij een kleine opdracht afgerond heeft. Zo blijft de hoeveelheid code klein en is het makkelijk om te zoeken wanneer er zich een probleem voordoet. Eens de wijziging gecommit is naar de version control repository moet de CI pipeline zijn werk doen. Een belangrijke stap is het bezorgen van feedback. Dit moet bestaan uit het resultaat van de build en de beschrijving van het probleem alsook de plaats waar het probleem zich voordoet, door bijvoorbeeld aan te geven welke test niet slaagt. De key factor hier is de tijd. Als de developer pas daags nadien feedback krijgt over de fout die hij gepusht heeft, wordt het al moeilijker om de fout te vinden en ze op te lossen.



Figuur 2.1: Voorbeeld van een Continuous Integration set-up (Riti, 2018)

2.3 Continuous Delivery

Eens het team met succes de Continuous Integration toepast kan men overschakelen naar de volgende stap: Continuous Delivery. Het is een manier dat ervoor zorgt dat de code die van de Continuous Integration stap komt, gebuild en voorbereid wordt voor een release. Er is echter wel nog een menselijke hand nodig om de build van deze stap te deployen en voor de buitenwereld beschikbaar te stellen (Fowler, 2013).
In Figuur 2.2 wordt de opbouw van een Continuous Delivery chain weergegeven, het is een grafische weergave van de stappen waaruit een Continuous Delivery pipeline moet bestaan. Dit is gebaseerd op de Continuous Integration chain, maar hier zijn extra stappen toegevoegd.

De eerste stap is het developen van de code die men wenst op te leveren. Net zoals bij Continuous Integration commit de developer de code naar de version control repository. De build scheduler haalt de laatst toegevoegde code op en test deze code. Enkel bij het slagen van alle testen wordt de code gebuild door de build scheduler en nog verder door de testmolen gehaald. Hier worden de integratie testen uitgevoerd en wordt gekeken of de code voldoet aan de kwaliteitseisen van een release. Eens de build slaagt voor alle testen wordt deze door de build scheduler klaar gemaakt voor release. Het vereist enkel nog menselijke goedkeuring om deze release te lanceren.
In dit proces is feedback ook uiterst belangrijk. Wanneer developers foute code pushen, moet de persoon die de foute code geschreven heeft zo snel mogelijk verwittigd worden. Op deze manier kan het euvel snel opgelost worden en kan er niet verder gebouwd worden op foutieve code. Mail kan een vorm zijn van feedback, maar er bestaan nog andere leuke vormen naast mailing.
Het bedrijf Dynatrace heeft bijvoorbeeld een licht ontworpen dat je in de kamer van het team kan hangen. Dit Internet of Things (IoT) gadget, DevOps UFO genaamd, is via WiFi

Figuur 2.2: Continuous Delivery chain (Riti, 2018)

verbonden met de pipeline omgeving. Het geeft rechtstreekse feedback over de staat van de CI/CD pipeline en is een vorm van monitoring. Als de commit door de pipeline geraakt zal de UFO groen kleuren, wanneer de build faalt kleurt de UFO rood en weet iedereen dat er een fout naar de CI server gebracht is. Zo weet de persoon die de foute code gepusht heeft dat er iets fout is en kan hij hier nog sneller op inspelen.

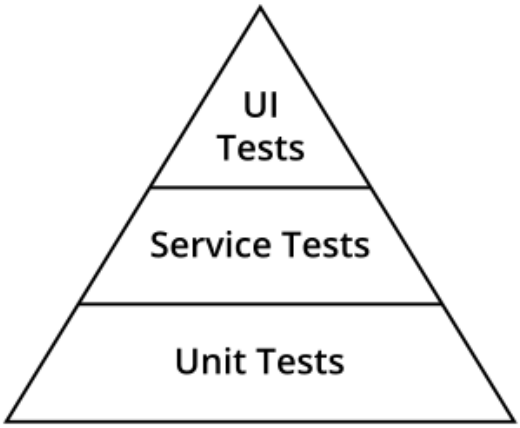
2.4 Continuous Deployment

De gelijkenis met Continuous Deployment is treffend, maar er is wel degelijk een verschil. Hier gaat men automatisch de veranderde code naar productie brengen. De veranderingen gaan door de volledige pipeline en eens ze slagen voor alle testen wordt - zonder menselijke interactie - de code naar productie gebracht (Claps, Svensson & Aurum, 2015). Dit wordt soms ook wel de 'train to production' genoemd, omdat elke code dat gepusht wordt naar de source-control automatisch tot bij de klant geraakt.

2.5 Automated Testing

Zoals hierboven reeds aangegeven is er geen Continuous Integration pipeline zonder automated testing. Het grootste doel van deze fase is - zoals de naam het zegt - testen van de code. Als de CI/CD pipeline voorzien wordt van voldoende en goede testen, creëert dit een veiligheidsgevoel. Op deze manier voldoet de software aan bepaalde criteria, die in de testen verwerkt worden. Het is dus een heel belangrijk onderdeel van de pipeline waar veel aandacht aan besteed moet worden. Om te voldoen aan de criteria van CI/CD moeten de testen automatisch gerund worden. Dit was tot voor kort echter vaak niet het geval. Er waren testers aangesteld om telkens opnieuw software te testen, door op de verschillende knoppen te drukken in de user interface (UI). Heel vaak slopen er fouten in omdat dit heel repetitief en saai werk was. Daar komt de automatisatie van pas (Vocke, 2018).

Mike Cohn kwam met het idee om testen op te delen in drie grote categorieën: Unit Tests, Service Tests en UI tests zoals u kan zien in figuur 2.3. Twee zaken zijn belangrijk om te onthouden: testen moeten uit verschillende lagen van detail bestaan en er moeten minder testen geschreven worden wanneer het team minder gedetailleerde code schrijft (high-level testing). Omdat bij high-level testen de requirements goed begrepen zijn door het test team.



Figuur 2.3: Test piramide (Vocke, 2018)

De grootste laag binnen de test omgeving zijn de Unit tests en staat het dichts bij de software code. Deze testen zijn snel om te schrijven en te runnen en kunnen gedetailleerde feedback geven wanneer een test faalt.
De laag erboven zijn de service tests - ook wel Integration tests genoemd - en focussen vooral op de functionaliteit van de applicatie. Ze testen de API calls en de integratie van de individuele functies.
De kleinste laag zijn de UI tests, ook wel end-to-end testen genoemd. Deze testen de user interface door bijvoorbeeld op knoppen te drukken, formulieren in te vullen of te navigeren doorheen de applicatie. Het nadeel van deze testen zijn dat ze fragiel, duur en tijdrovend zijn om te bouwen en traag om te runnen zijn. Om de snelheid te garanderen en de hoeveelheid build times klein te houden is het belangrijk de vorm van de piramide te behouden. Het gevaar dreigt om de test omgeving als een ijsjes hoorn te vormen met meer UI tests dan Unit tests (Fowler, 2012).

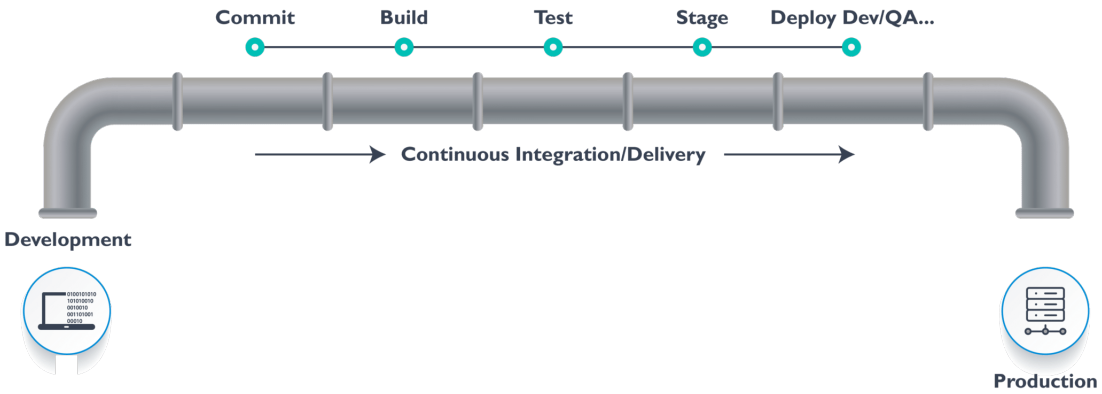
Vandaag zijn deze categorieën iets te simplistisch voorgesteld, vanwege de toenemende complexiteit van de software. Maar het is wel nog altijd een uitstekende referentie om de opbouw van testen uit te leggen. De Automated Test Quadrant is een voorbeeld waarop een test omgeving gebaseerd kan worden. Het is ook model, net zoals de test piramide en is bedoeld als gids. Er bestaan verschillende modellen waarop je je kan baseren.
De Automated Test Quadrant baseert deelt de testen op in vier categorieën op basis van hun rol en complexiteit.
De eerste kwadrant focust op de logica aan de business-zijde en zijn high-level (bevatten weinig details). De tweede testen veel details die te maken hebben met de business logica die de klant wenst.
De derde kwadrant zijn testen die geschreven zijn met de toekomst in het achterhoofd. Ze testen of de applicatie makkelijk uitbreidbaar is. De vierde soort test de technische zaken zoals het gebruik van de API enzovoort (Smart, 2017).

3. CI/CD pipeline

Dit hoofdstuk zal een Continuous Integration en Continuous Delivery pipeline in detail bekijken. Binnen een CI/CD pipeline zijn er verschillende lagen die elkaar aanvullen en opvolgen. Deze worden hier kort besproken, omdat dit een essentieel onderdeel is van de thesis. De voordelen, alsook de nadelen, van zo een pipeline worden ook aangehaald. Om de algemene voor- en nadelen te vinden wordt er teruggegrepen naar de literatuur. Er is namelijk al enorm veel geschreven over een CI/CD pipeline en wat de voor- en nadelen kunnen zijn. De toegevoegde waarden en de schaduwzijden van een CI/CD pipeline voor Amista worden aangeduid door Pieter-Jan Deraedt, CTO bij Amista.

3.1 Continuous Integration en Continuous Delivery pipeline

De pipeline stelt het proces van de verschillende workflows voor die reeds in deze thesis besproken zijn. Het is een samentrekking tussen het proces van Continuous Integration en Continuous Delivery waarbij alles automatisch wordt aangestuurd. In onderstaande figuur 3.1 is te zien hoe een pipeline eruit ziet waar alle basislagen aan bod komen en integreren met elkaar. Alles begint bij de developer die een verandering in de code aanbrengt en deze commit naar de version control repository waar de build scheduler toegang toe heeft. De build scheduler merkt op zijn beurt de commit en begint aan zijn taak: het builden van de code. Voor deze stap zorgt de build scheduler ervoor dat de unit testen uitgevoerd zijn alvorens de build begint. Eens de testen en de build slagen is het de taak om de automatische build tests uit te voeren en de code uitvoerig te testen of deze klaar is voor productie. Dit kan op een realistische testomgeving gebeuren of de code wordt door iemand anders nagekeken of deze voldoet aan alle kwaliteitseisen. Eens de code door de testen raakt wordt deze klaar gemaakt voor productie door deze naar de stage fase te brengen.



Figuur 3.1: Continuous Integration en Continuous Delivery pipeline (Tuli, 2018)

Eens de code zich in de stage fase bevindt kan deze manueel worden gedeployed om naar de klant te brengen.

3.2 Lagen binnen een CI/CD pipeline en beschikbare tools

De CI/CD pipeline bestaat uit verschillende lagen die naadloos met elkaar moeten samenwerken om het beste resultaat te bekomen. De verschillende taken en beschikbare tools van elke laag worden besproken.

3.2.1 Source Control System

Dit systeem houdt alle veranderingen bij aan de code en zal de veranderingen ook beheren zodat ze niet overlappen. Het laat toe dat meerdere developers tegelijk aan - soms dezelfde - code werken. Een source control systeem houdt dan de veranderingen bij wat elke developer gedaan heeft. Een best practice is dat er één stabiele versie van de software bestaat, meestal master genoemd.
Een gangbare praktijk binnen dit systeem is het maken van branches. Hier wordt een kopie gemaakt van de stabiele master, waardoor men wat kan knoeien in de branch zonder de master te beschadigen. Als men overtuigd is van de kwaliteiten dat men op een branch gemaakt heeft kan men de branch 'mergen' in de master. Deze techniek wordt ook wel 'revision control' of 'version control' genoemd (Skelton, 2014) en (Riti, 2018). Voorbeelden van zo een source control system zijn Git, CVS (Concurrent Version System), Subversion en Mercurial.

3.2.2 Hosted Version Control Services

Wordt ook wel Code Repository Server genoemd. Hier wordt de software van het source control system opgeslagen. Dit kan op een interne server opgeslagen worden, of op een externe die door bedrijven aangeboden wordt. Dit maakt het makkelijker voor developers om samen te werken en dezelfde bron te gebruiken. Deze stap is ook crusiaal om een goede

Continuous Integration te voorzien. Voorbeelden van Repository Management Services zijn: GitHub, BitBucket, GitLab en Coding zijn veruit de populairste op de markt.

3.2.3 Build Scheduler

Een Build Scheduler kan ook wel een Continuous Integration Server genoemd worden. Dit zorgt ervoor dat - telkens wanneer er code gecommit wordt - de stappen in de pipeline worden uitgevoerd. De taken van de build scheduler zijn:

- De nieuwe code ophalen van de Repository Server en deze samenvoegen met de oude code
- De testen uitvoeren
- Het builden van de software
- Feedback geven aan de developer over voorgaande stappen

Deze taken kunnen ook door een script volbracht worden, maar het is belangrijk dat deze taken automatisch gebeuren telkens er code gecommit wordt naar de repository manager (Riti, 2018).

In het hoofdstuk 5 worden de meest populaire build schedulers van vandaag met elkaar vergeleken. Naast Jenkins, Circle CI, Bamboo en Travis CI zijn er nog andere build schedulers op de markt die gebruikt kunnen worden: Team City, Gitlab CI, CodeShip en nog enkele tientallen.

3.2.4 Artifact Repository Manager

Als laatste zijn de Artifact Repository Managers aan de beurt. Deze repository manager houdt alles bij wat nodig is om de applicatie te deployen, zoals:

- Gecomprimeerde applicatie code
- Code in verband met de infrastructuur
- Images van virtuele machines
- Configuratie data

Deze tool houdt alle geschiedenis bij van de bovenstaande files. Zoals eerder vermeld kan men vanaf hier de software (automatisch) deployen. Dit is de allerlaatste fase in de CI/CD pipeline (Skelton, 2014). Sonatype Nexus en Archiva zijn dan weer voorbeelden van tools die gebruikt worden als artifact repository manager, deze houden bij wijze van spreken de code bij die klaar is om te deployen.

3.3 Voordelen

- Maak de applicatie die de klant wil. Het grootste probleem bij het falen van projecten ligt nog altijd bij het te grote verschil tussen de gemaakte applicatie en hetgeen de

klant voor ogen had. Elke persoon die met IT bezig is weet dat de klant heel vaak niet weet wat hij wil. Dit is zeer gevaarlijk wanneer er grote projecten moeten afgeleverd worden en de klant al 10 keer van idee veranderd is tegen de datum van oplevering. Mede dankzij een combinatie van Agile en CI/CD is het makkelijker om een minimale hoeveelheid aan code te schrijven, dit aan de klant te tonen, feedback te krijgen en erop verder te bouwen. CI/CD komt hier echt tot zijn recht vanwege de makkelijkheid om voort te bouwen op geschreven software en tevens de kwaliteit van voorgaande geschreven code niet te verliezen. Dit is de ideale combinatie om samen met de klant tot een gewild en bruikbaar product te komen (Humble, 2012).

- Wanneer men de principes van CI/CD volgt en minstens één keer per dag code commit naar de main repository verkleint dit de kans op grote bugs. Bij het pushen van kleine stukjes code en de onmiddellijke feedback die volgt door de build scheduler, weet het team of de developer dat er iets mis is met het kleine stukje code die hij probeerde te pushen richting repository. Het is op deze manier veel makkelijker om de fout eruit te halen dan wanneer de developer moet zoeken in code die hij weken geleden geschreven heeft. (Fowler, 2006).
- Bij het niet krijgen van feedback over de kwaliteit kan het gebeuren dat de developer voortbouwt op foutieve code. Door het onmiddellijk krijgen van feedback kan er veel tijd bespaard worden.
- Wanneer de developer een fout heeft ontdekt vlak na het pushen van de code zal hij die fout, hoogst waarschijnlijk, niet meer opnieuw maken. Dit zal veel tijd besparen tijdens verdere implementatie.
- In de meeste omgevingen met een CI/CD pipeline is het niet nodig om een Quality Assurance team (QA team dat waakt over de kwaliteit van de software) te hebben om de testen uit te voeren, wat de kosten doet dalen.
- Het risico op downtime van een applicatie verkleind door de vele testen die automatisch gerund worden alvorens de aanpassing naar buiten wordt gebracht. Dit is wel onder de voorwaarde dat de testen in een soortgelijke omgeving als de echte applicatie worden getest en dat de testen aan alle kwaliteitseisen voldoen om de applicatie werkende te houden.
- De stress die komt kijken bij een release kan achterwege gelaten worden. Door continu kleine aanpassingen door te voeren, die aan alle kwaliteitseisen voldoen eens ze door alle testen raken, zal er minder stress bij komen kijken. Dit is een heel groot voordeel voor iedereen die iets met de applicatie te maken heeft. Wanneer een team minder stress ervaart zal het ook minder fouten maken en de productiviteit ten goede komen.
- Bij een grote release is er soms ook een team van wacht die problemen, wanneer ze opduiken, kunnen oplossen. Bij een CI/CD pipeline is het niet nodig om schrik te hebben voor problemen omdat de software uitvoerig getest geweest is alvorens live te gaan.

3.4 Nadelen

Het vergt echter wat inspanningen om een Continuous Integration en Continuous Delivery pipeline op te zetten. De inspanningen moeten door heel het bedrijf geleverd worden.

- Er zal heel wat werk kruipen in het opzetten van een CI/CD pipeline. Dit gaat gepaard met werkuren, die geld kosten.
- De developers moeten zich ervan bewust zijn dat het enorm belangrijk is om op regelmatige basis en met kleine werkende aanpassingen de code naar de repository te sturen. Dit kan een aanpassing vergen om op deze manier te werk te gaan.
- Er kruipt heel wat tijd in het schrijven van goede testen die de applicatie op alle vlakken voldoende test.
- De geschreven testen moeten ook zeer goed onderhouden worden. Elke nieuwe functionaliteit moet uitvoerig getest worden zonder de vorige functionaliteiten uit het oog te verliezen.
- Wanneer het team nog nooit met een CI/CD tool heeft gewerkt vraagt dit ook een aanpassing dat vaak gepaard gaat met een training.
- Wanneer gekozen wordt om de code op een lokale server te runnen moet deze server ook onderhouden worden om veilig te blijven functioneren.
- Er komen nieuwe technologieën kijken bij het opzetten van een CI/CD pipeline. Sommige tools moeten betaald worden alvorens te kunnen gebruiken.

4. SAP

SAP is een Duitse onderneming opgericht in 1972 dat softwareoplossingen aanbiedt voor grote ondernemingen. Met meer dan 413.000 klanten verspreid over 180 landen mag SAP zich marktleider noemen op gebied van bedrijfssoftware. SAP heeft zich gespecialiseerd in ERP (Enterprise Resource Planning), software dat alle processen van het bedrijf automatiseert[1]. Een ERP systeem beheert meerdere functies en bedrijfsprocessen van één bedrijf op basis van een centrale database. Het heeft als doel de gegevens van de organisatie optimaal te gebruiken in de gehele organisatie en een betere beheersing van de bedrijfsprocessen voorzien. De oplossingen die SAP aanbiedt zijn vooral bedoeld voor de grotere bedrijven. Ze bieden software aan voor elke mogelijke industrie die er vandaag de dag bestaat, maar pakken uit met hun specialiteiten in de cloud business software. Hieronder worden kort de programma's beschreven die gebruikt worden in de voorbeeldapplicatie. Het geeft een algemeen beeld van de omgeving waar Amista graag een CI/CD pipeline wil voor integreren.

SAP Cloud Platform

SAP Cloud Platform is een Platform as a Service (PaaS), dat aangeboden wordt door SAP. het is een online platform dat - door hardware en software samen te brengen - applicaties overal toegankelijk maakt en samenbrengt tot één platform dat online ter beschikking is[2]. SAP Cloud Platform wordt zowel voor development als deployment gebruikt, maar reikt ook de hand aan verschillende technologieën: Internet of Things, Big Data, Artificiële Intelligentie enzovoort. Het is een platform dat zowel on-premise - waarbij software enkel lokaal op een computer beschikbaar is - als cloud technologieën samen kan brengen. Je

[1] https://www.sap.com/products/enterprise-management-erp.html
[2] https://cloudplatform.sap.com/index.html

kan er de technologieën ook uitbreiden en zelf ontwikkelen. Het haalt zijn kracht uit de perfecte integratie met andere SAP software die je ook nog eens kan uitbreiden.

Het verschil tussen een Platform as a Service (PaaS), een Infrastructure as a Service (IaaS) en Software as a Service (SaaS) is te vinden in de hoeveelheid men zelf voorziet in de opzet van een applicatie. SAP Cloud Platform biedt de mogelijkheid tot online development, testen en deployen van applicaties. Ook de database draait online in de cloud. Het is echter ook mogelijk om de on-premise applicaties te hosten, dit maakt het een perfect voorbeeld van een PaaS. Bij een SaaS omgeving draait alles in de cloud, het is niet mogelijk om ook maar iets on-premise uit te voeren. Alles wordt geregeld door de leverancier: security, updates, ...
Bij een IaaS voorziet de leverancier enkel de hardware (in de cloud) aan de klant. Een perfect voorbeeld hiervan is de server die gebruikt wordt in hoofdstuk 5. Digital Ocean voorziet enkel de Ubuntu server, wat er gebeurd met de server is volledig autonoom te beslissen.

SAPUI5

SAPUI5 is een framework dat uitgevonden is door SAP en bevat verschillende libraries die bovenop JavaScript gebouwd zijn. Via het SAP Cloud Platform kunnen er front-end applicaties gemaakt en deployed worden die geschreven zijn in SAPUI5. Het is een framework dat bedoeld is om HTML5 applicaties te bouwen die bijna automatisch responsive zijn zonder veel bijkomende code toe te voegen. Het maakt gebruik van dezelfde lay-out zodat het voor de eindklanten één mooi geheel vormt. Het biedt de developers een resem aan UI controls aan, zodat er een consistenter en beter UX design gehanteerd wordt (SAPSE, 2019).
Er is de voorbije jaren meer aandacht gegaan naar Responsive en UX design. SAP heeft, doormiddel van Fiori, daar een antwoord op geboden. Fiori is een term die OData (als dataservice) gebruikt en eenzelfde lay-out doorheen alle applicaties voorziet. Deze applicaties zijn tevens ook mobile friendly. SAP heeft ook enkele templates uitgewerkt waarbij veel voorkomende zaken makkelijk te maken zijn.Voorbeelden zijn een List Report, Object Page, ... [3].

SAP HANA

Een In-Memory Data Platform staat als titel op de site van SAP te lezen. Het is een platform dat gebruik maakt van het RAM-geheugen van de computer, wat enorme snelheden met zich meebrengt, maar ook een enorm kostenplaatje. Dit even terzijde wordt SAP HANA aangeprezen als een platform om ingewikkelde, real-time en analytische berekeningen uit te voeren op data. Het is een Relationeel Database Management Systeem (RDMBMS) dat geïntegreerd kan worden in SAP Cloud Platform, waarbij het mogelijk is om zowel on-premise als in de cloud te werken, of een combinatie van beiden[4].

[3] https://www.sap.com/products/fiori.html
[4] https://www.sap.com/products/hana.html

5. Methodologie

Deze thesis gaat aan de hand van een vergelijkende studie op zoek naar de beste build scheduler voor de specifieke set-up die ze bij Amista hanteren. De omgeving waar de build-scheduler overweg mee moet kunnen ziet er als volgt uit: een SAPUI5 webapplicatie met een SAP HANA database draaiend via Node.js en alles gehost op SAP Cloud Platform. Er worden aan de hand van criteria enkele build-schedulers gekozen die worden opgezet in bovenstaande omgeving, de stappen die hierbij komen kijken worden zorgvuldig uitgeschreven en in een handleiding gegoten. Zo is deze proof-of-concept reproduceerbaar en heeft Amista een mooi voorbeeld om een CI/CD pipeline op te zetten.

5.1 Vergelijkende studie van de build schedulers

We beginnen met een klein stukje theorie over de build scheduler, om nadien tot de vergelijking over te gaan. Hier worden ook de verschillende criteria die Amista aangaf als belangrijk en minder belangrijk opgedeeld in twee delen: de functionele en de niet-functionele requirements. Binnen beide categorieën wordt er nog een opsplitsing gemaakt tussen: must-haves, should-haves en nice-to-haves. Met deze informatie kunnen we al een eerste vergelijking maken tussen de build-schedulers die vandaag op de markt te vinden zijn. Als resultaat krijgen we een long list waar de tools naast elkaar worden gezet en vergeleken kunnen worden aan de hand van criteria. Uit deze eerste vergelijking nemen we de beste tools eruit om te testen in een realistische omgeving. Dit zal in hoofdstuk 6 gebeuren.

5.1.1 Build Scheduler

In Hoofdstuk 2 wordt er kort even aangehaald wat een build scheduler is en doet. Om hier toch een beter beeld van te krijgen, kaarten we nog een stukje theorie aan, om nadien aan de technische vergelijking te beginnen. De belangrijkste taak van een build scheduler is het uitvoeren van de builds. Met de principes van Continuous Integration in het achterhoofd weten we dat deze build uitgevoerd wordt na elke commit in de version control repository. Maar in praktijk zijn er twee verschillende categorieën: polling builds en scheduler builds. Bij polling builds wordt er, meestal op een zeer korte tijd zoals elke minuut bijvoorbeeld, gekeken of er veranderingen zijn aangebracht in de repository. Wanneer de build scheduler een verandering opmerkt zal hij automatisch de build uitvoeren.
Bij scheduler builds wordt er, meestal op een dagelijkse basis, naar de veranderingen in de repository gekeken. Bij veranderingen worden de builds automatisch uitgevoerd. Deze manier is niet helemaal in regel met de principes van CI/CD omdat er op een dagelijkse basis feedback wordt voorzien in plaats van onmiddellijk na de commit, zoals bij polling builds.

Iedere developer weet dat testen een cruciale rol hebben binnen software. Het automatiseren van de tests gebeurd door een testing tool, zoals Karma vaak gebruik wordt binnen SAPUI5 development. Deze tool zorgt ervoor dat de QUnit (Unit tests binnen SAPUI5) en OPA tests (Integration tests binnen SAPUI5) geautomatiseerd worden. Continuous Integration en Continuous Delivery draait allemaal rond het uitvoeren van de testen en het feedback geven over de resultaten ervan. Het is dan ook de taak van de build scheduler om voor een uitstekende samenwerking te zorgen met de gebruikte testing tool.

Tevens moet de build scheduler naadloos samen werken met de gebruikte version control tool. Hij moet namelijk de commits kunnen ophalen via deze version control tool.
Een build scheduler zorgt er ook voor dat het mogelijk is om de veranderingen aan de code te testen op een test server, indien deze aanwezig is. De build scheduler moet het mogelijk maken om de code te deployen naar de test server waar een realistische opstelling nagemaakt is om zo de code te testen. Feedback bezorgen aan de developer is het eigenlijke doel van de build scheduler. Hoe beter en sneller de feedback, hoe sneller het probleem opgelost kan worden. De developer verliest op deze manier weinig tijd en weet exact waar de code fout gelopen is.

5.1.2 Requirements

Nu we weten hoe een build scheduler werkt is het belangrijk om te weten naar welke criteria er moet gekeken worden om de beste tool eruit te kiezen. De functionele en niet-functionele requirements worden opgelijst om een beter beeld te krijgen hoe we een vergelijking moeten toepassen.

Functionele requirements

Een functionele requirement is een functie wat het systeem moet doen. De punten die in deze sectie worden aangehaald komen uit hoofdstuk 3. Om een duidelijk beeld te krijgen wat de build scheduler moet doen worden de functionele requirements hier nog eens kort besproken.

- De build scheduler moet de aanpassingen aan de code sneller naar de klant brengen
- De downtime van een applicatie moet dalen
- Er moet vermeden worden dat een applicatie stopt met werken wanneer een fout in de code wordt gedeployed

Niet-functionele requirements

Een niet-functionele requirement legt uit hoe het systeem een bepaalde functie moet uitvoeren. Voor Amista is het belangrijk dat de build scheduler voldoet aan hoge security eisen. Vaak wordt er met hele grote projecten gewerkt die gevoelige data en broncode beschikken. Om optimale veiligheid te garanderen wordt gewerkt met een build scheduler die op een on-premise systeem zal draaien. Een andere mogelijkheid, waar vandaag de dag veel in wordt geïnvesteerd, is een cloud build scheduler. Dit zorgt ervoor dat de code buiten de onderneming gaat, wat toch een zeker risico met zich meebrengt.

5.1.3 Criteria

Must-haves

De must-haves van de build scheduler zullen vooral te maken hebben met de omgeving waarin ze moeten werken. Zoals in de literatuurstudie al beschreven staat, moet een build scheduler gebruikt worden met een SAPUI5 applicatie, samen met een SAP HANA database gehost op SAP Cloud Platform.

In deze thesis wordt dezelfde source control manager gebruikt als bij Amista. Het is niet van toepassing om zelf een source control system en repository manager te kiezen, er moet gebruik gemaakt worden van Git en Bitbucket. Het is vanzelfsprekend dat de build scheduler moet kunnen integreren met Git en Bitbucket.

Om een betere vergelijking te kunnen maken heeft Amista een Ubuntu server voorzien voor de uitwerking van de proof-of-concept, zie hoofdstuk 6. Deze dient ook als simulatie voor de realistische on-premise set-up. Het is dus noodzakelijk dat de build scheduler op een Unix-based systeem kan draaien.

De build scheduler moet ook deel kunnen uitmaken van een Continuous Delivery pipeline en automated tests kunnen uitvoeren. Een belangrijk onderdeel van de automated tests binnen SAPUI5 zijn OPA en QUnit tests, deze moeten zeker uitgevoerd kunnen worden door de build scheduler. Binnen SAPUI5 wordt er vaak gebruik gemaakt van Karma om bovenstaande tests te automatiseren, daarom is het een must-have dat de build-scheduler

de werking van Karma ondersteunt.

Het is voor Amista heel belangrijk dat de build scheduler makkelijk in gebruik is. Ondanks dat deze technologieën vaak nieuw zijn voor de developers zou het leerproces niet lang mogen duren. Ze redeneren dan ook op volgende manier: ze spenderen liever wat meer geld aan een build scheduler waar de developers snel mee weg zijn, dan een goedkopere waar de softwareontwikkelaars een hele tijd over doen om het proces onder de knie te krijgen. Het geld dat ze spenderen aan de duurdere, maar makkelijkere build scheduler is kleiner dan de lonen die ze moeten uitbetalen aan de programmeurs om de tool onder de knie te krijgen.

De tools die we gaan vergelijken moeten ook getest worden op de mogelijkheid tot een audit. Dit omdat Amista ook vaak voor voedingsbedrijven werkt en dit wettelijk verplicht is. Veiligheid is ook een belangrijk punt voor Amista, omdat ze heel vaak met zeer gevoelige data van hun klanten werken. Daarom is het belangrijk dat de tools goed omgaan met security. Hoe valt dit te testen? Als de tool meerdere malen per maand een nieuwe versie van de software uitbrengt zijn ze begaan met de veiligheid.

Als we bovenstaande punten even kort samenvatten moet de build scheduler aan volgende vereisten voldoen:

- Hij moet bruikbaar zijn met een SAPUI5 applicatie, een SAP HANA database gehost op SAP Cloud Platform
- Hij moet kunnen integreren met Git & Bitbucket
- De build scheduler moet op een Unix-based server kunnen draaien
- De tool moet deel uitmaken van een CI/CD pipeline en automated tests kunnen uitvoeren, specifiek de Qunit en OPA tests uit de SAPUI5 applicatie door het gebruik van Karma te ondersteunen.
- Gebruiksgemak moet centraal staan
- Het moet mogelijk zijn om auditing toe te passen
- Security is belangrijk, zijn er maandelijks meerdere releases ter beschikking?

Should-Haves

De build scheduler zou moeten beschikken over gratis gebruik van de software om te experimenteren. Het is de bedoeling dat we in het volgende hoofdstuk een proof-of-concept kunnen uitwerken om bepaalde build schedulers te installeren. Daarom is het nodig om een gratis versie te hebben die de opstelling mogelijk maakt.
Amista communiceert vaak via Skype For Business, ze zouden het leuk vinden om via dit kanaal ook feedback te krijgen over de staat van de build.

Nice-to-Haves

De build scheduler moet niet per se op een cloud draaien, het belangrijkste is dat de data lokaal gehouden kan worden door de build scheduler op een on-premise installatie op te zetten. Maar Amista wil graag deze optie openhouden voor de toekomst. Daarom is het

mooi meegenomen als de gekozen build scheduler aan deze vereiste voldoet, maar het is geen breekpunt.
Bij Amista denken ze aan de toekomst. Het zou leuk zijn als de tool kan integreren met Docker om de builds te bouwen. Omdat dit toch een handige, opkomende tool is binnen de informatica.
Het is geen noodzaak om de build scheduler informatie via Slack of WhatsApp te versturen, maar een pluspunt.
De CTO van Amista zou het ook als een pluspunt zien als de build scheduler het mogelijk maakt om een rapport te genereren wat er de voorbije week/maand gebeurd is in de code. Wie heeft wat gedaan, wie maakt de meeste fouten, ...? Zo'n zaken zijn handig om de prestaties van de developers binnen een team te kennen en te versterken.

5.1.4 Long list

Nu de verschillende criteria gekend zijn kan de effectieve vergelijking tussen de build schedulers gebeuren. Eerst zal er een korte uitleg gegeven worden over welke programma's we gaan vergelijken en nadien vindt de vergelijking plaats in een overzichtelijke tabel.
Enkele woordjes uitleg bij de titels van de tabel:

- SAP CP & UI5 staan voor de integratie met SAP Cloud Platform en UI5
- Gebruik staat voor gebruiksgemak
- Audit checkt of het mogelijk is om auditing toe te passen
- Secure kijkt of er meerdere releases per maand worden uitgebracht. Zo ja is de build scheduler secure.

Jenkins

Jenkins is de meest gekende build scheduler binnen de IT-wereld. Het is een op zichzelf staande, open source automation server dat ontstaan is in 2011 na een afscheiding van het Hudson project. Jenkins is vooral bekend om de vele plug-ins die het mogelijk maken om met bijna alle talen en platformen te integreren. Jenkins is geschreven in Java en draait op een Java platform.

Circle CI

Circle CI is opgericht in 2011 in San Francisco en wordt door vele grote bedrijven gebruikt in hun Continuous Integration pipeline. Het grootste deel van Circle CI is geschreven in Clojure en de Frontend in ClojureScript.

Bamboo

Bamboo, een Java-based Continuous Integration en Continuous Delivery tool, werd opgericht in 2007 door het bedrijf Atlassian dat ook Bitbucket onder zijn hoede heeft.

		Jenkins	Circle Ci	Bamboo	Travis
Must-haves	**SAP CP & UI5**	Ja	Geen info	1 blogpost	Ja
	Git	Ja	Ja	Ja	Ja
	Bitbucket	Ja	Ja	Ja	Nee
	On-premise	Ja	Ja	Ja	Ja
	CI/CD pipeline	Ja	Ja	Ja	Ja
	Karma	Ja	Ja	Ja	Ja
	Gebruik	Makkelijk	Makkelijk	Matig	Makkelijk
	Audit	Ja	Basic	Ja	Ja
	Secure	Ja	Meestal	Nee	
Should-haves	**Prijs**	Gratis	$35 / user	$1100 / jaar voor 1 remote agent en ongelimiteerd aantal jobs	$2739 / jaar voor 5 jobs
	Trial	Gratis	20 dagen	30 dagen	Gratis voor open-source
	Skype for Business	Ja	Nee	Nee	Ja
	Container support	Ja	Ja	Ja	Ja
Nice-to-haves	**Slack**	Ja	Ja	Ja, via derde partij	Ja
	Rapport	Ja	Ja	Nee	Ja

Tabel 5.1: Long list van CI/CD tools

Travis CI

Travis CI is een integration tool dat geschreven is in Ruby en opgericht in 2011 in Duitsland. Het staat bekend om zijn samenwerking met GitHub.

5.2 Voorbeeldapplicatie dat Amista zal gebruiken

Hoe ziet de omgeving eruit waar Amista een CI/CD pipeline in wil opzetten? In de literatuurstudie worden de gebruikte tools uitgelegd, in dit hoofdstuk worden ze uitgewerkt en de onderliggende relaties besproken. Amista heeft een Ubuntu server ter beschikking gesteld om te experimenteren. In deze thesis wordt de server gebruikt als Continuous Integration server.

Om de leesbaarheid te bevorderen is er gekozen om de uitgewerkte handleiding in een apart bestand te schrijven. Er wordt namelijk vaak verwezen naar figuren. U kan de uitgewerkte handleiding terug vinden in het bestand proof-of-concept in de map 'bijlagen'.
Hieronder kan u de extra duiding terug vinden die bij de handleiding hoort.

5.2.1 Build scheduler server

Amista heeft een Ubuntu server ter beschikking gesteld dat wordt gehost op Digital Ocean, een Amerikaanse hosting bedrijf. Digital Ocean was de derde grootse speler op de markt in 2018 wanneer men spreekt over web-hosting computers. Ze zijn gespecialiseerd in cloud based oplossingen en hebben datacenters over heel de wereld.

Ubuntu server

Ubuntu is een Linux distributie, gebaseerd op het bekende Debian, en gekend vanwege zijn open-source eigenschappen. Deze versie van Linux wordt vooral gebruikt voor cloud computing, vandaar dat Digital Ocean ervoor kiest om met een Ubuntu 18.04 te werken. De versie 18.04 wordt ook wel 'Bionic Beaver' genoemd. Amista heeft voor de uitwerking van de proof-of-concept 6 een server voorzien met een x64-bit Operating System waar 1 CPU en 1GB RAM ter beschikking staat.

SSH staat voor secure shell en is een software protocol dat voor een veilige verbinding (tunnel) zorgt tussen de client en de server. Het wordt gebruikt voor het configureren van een server, het beheren van netwerken en operating systems. Alle gegevens dat tussen beiden worden uitgevoerd zijn geëncrypteerd waardoor het moeilijker wordt voor hackers om de data te bemachtigen.
Een server die gebruik maakt van SSH wordt ook wel een sshd server genoemd.

De default manier om in te loggen is via een account en een paswoord, maar het is ook mogelijk om het account en het paswoord te vervangen door een private en een publieke sleutel. Dit principe noemt key-based authentication en wordt vooral tijdens development en in scripts gebruikt of voor single sign-on. SSH genereert een private en een publieke sleutel op de client bij configuratie. De private key moet veilig bewaard worden op de client computer. De public key moet doorgegeven worden aan de remote server om de verbinding te kunnen maken.
Wanneer de client wil inloggen op de server voert hij een request uit. De server maakt via zijn public key een bericht en stuurt dit als response door naar de client. De client leest het bericht aan de hand van zijn private key en stuurt dan een aangepaste response terug naar de remote server. De server valideert deze response.
Bij een geldige private key zal er een goede response verstuurd worden, bij een ongeldige private key een foute response.
De 'id_rsa.pub' is de publieke key van de client die op de server moet komen om zo de ssh validatie te voorzien, dit wordt ook wel een ssh session genoemd. Eens de ssh session geconfigureerd is zal het niet nodig zijn om via een paswoord in te loggen op de remote server via deze client. Om veiligheidsredenen is het uiteraard beter om enkel via key-based authenticatie in te loggen en zo het paswoord uit te sluiten.

Om de remote server nog meer te beschermen tegen cyber aanvallen is het nodig om een firewall op te zetten. In de voorbeeldapplicatie maken we gebruik van de UFW Firewall, wat staat voor Uncomplicated Firewall. Dit is een gebruiksvriendelijke tool dat helpt om de IP Tables onder controle te houden en zo te zorgen dat maar bepaalde services toegelaten

worden tot onze server. In Linux maken ze gebruik van het protocol SSH via de service OpenSSH. Deze heeft tevens ook een profiel bij UFF wat het makkelijk maakt om de firewall op te zetten en enkel de SSH toe te laten die geconfigureerd is.

5.2.2 Database

Binnen SAP wordt een HANA database aangeraden om te gebruiken. Momenteel is versie 2.0 van SAP HANA op de markt en deze versie biedt tal van extra mogelijkheden ten opzichte van de vorige versie. SAP HANA wordt zeer goed ondersteund door de andere programma's binnen SAP.
Voor de database maken we gebruik van een Multi-Target Application Project, dit is een template dat SAP voorziet en is een goede uitvalsbasis om te gebruiken in de voorbeeldapplicatie.

Een realistische opstelling van een CI/CD voorbeeldapplicatie start bijna nooit vanaf nul, het bouwt meestal voort op bestaande, geschreven software. Daarom zal er in deze voorbeeldapplicatie manueel een basis gelegd worden, zo kan er aan de hand van een build scheduler voortgebouwd worden op deze geschreven software. De bescheiden database waar we naartoe willen, bestaat uit slechts één entiteit: een Artiest en heeft volgende properties: ID, Naam en JaarVanOorsprong.

De Artiest wordt via een HDB CDS Artifact, een Core Data Services document dat definities bevat om objecten te creëren in de database, gedefinieerd.

Node.js module

Om de models te gebruiken dat gecreëerd zijn, moeten we een tweede module toevoegen aan de HANA database: een Node.js module om de data als OData service te kunnen gebruiken. Deze module implementeert XSJS en XSODATA die op hun beurt zorgen voor de transformatie van het data model en de bereikbaarheid naar de buitenwereld.
CRUD (Create, Read, Update en Delete) operaties uitvoeren op de data beschikbaar stellen als OData is de taak van XSODATA.
XSJS zorgt dan weer voor de integratie met SAPUI5 en laat toe dat SAPUI5 applicaties de data kunnen lezen en bewerken.

5.2.3 UI5-applicatie

Hoe een SAPUI5 applicatie eruit ziet en hoe deze tot stand gekomen is kan men bekijken in de handleiding in het hoofdstuk UI5 applicatie.

6. Proof-of-concept van een CI/CD pipeline

Uit vorig hoofdstuk is gebleken dat Jenkins de tool is die aan het meeste must-haves, should-haves en nice-to-haves voldoet. In dit hoofdstuk wordt deze tool extra onder de loep genomen door hem op te stellen in een realistische omgeving zoals in hoofdstuk 5 besproken is. Door Jenkins in zo'n omgeving op te zetten krijgen we een meer realistisch beeld of deze build scheduler effectief doet wat er van hem verwacht wordt. Op het einde van dit hoofdstuk zal er een conclusie worden gemaakt of Jenkins aan te bevelen valt om in de specifieke omgeving op te zetten.
Maar eerst worden enkele tips uitgewerkt van SAP, hoe een CI/CD pipeline opgestart moet worden.

6.1 Continuous Delivery principles

Volgens Riti (2018) is het uitermate belangrijk om volgende stappen te volbrengen om tot een goede Continuous Delivery omgeving te komen.

- Er moeten goede branching strategieën bepaald worden om het team goed te laten samenwerken
- Een belangrijk onderdeel van Continuous Integration is natuurlijk testen, deze zijn in Continuous Delivery niet te vergeten
- Een stap verder is het automatisch uitvoeren van deze testen
- Na het slagen van de testen en het automatisch builden van de software moet de build klaar gemaakt worden voor release

6.2 CI/CD pipeline op SAP Cloud Platform

Het vergt enige vereisten om te voldoen aan de regels van Continuous Integration volgens Kramer (2018):

- Hou alles goed bij via een version control systeem
- Automatiseer de build
- Zorg ervoor dat tijdens de build er Unit testen lopen
- Het team moet op regelmatige basis commits uitvoeren
- Elke verandering moet gebuild worden
- Als er errors tevoorschijn komen tijdens de build, moeten die opgelost worden
- De build moet uitgetest worden op een kopie van de productieomgeving
- Automatiseer het deployment

Eens deze regels zijn toegepast, kunnen we spreken van een CI implementatie. Vaak wordt CI in combinatie gebracht met Continuous Delivery. Om dit in een vloeiende lijn te laten lopen, spreekt men van een CI/CD pipeline.

6.3 CI/CD pipeline volgens SAP

Een programmeur schrijft nieuwe code voor een verandering die de klant wil uitvoeren. Idealiter zou dit - voor het mergen naar de masterapplication - eens door een voter build moeten gaan, waar automatische tests aanwezig zijn die kijken of de code geen problemen zou geven als je die zou mergen met de master. Een laatste stap voor de code naar de master gemerged wordt, is het toepassen van code reviews door collega developers (het 4-ogen principe). Na het samenvoegen wordt automatisch de CI-build geactiveerd. De code gaat door de automatische tests. Eens de testen slagen worden de wijzigingen geïntegreerd op de master.

Dan komt de Continuous Delivery fase, waarbij de code nog eens door een testsysteem gaat. Deze fase gebeurt volledig automatisch, maar er kunnen ook manueel testen uitgevoerd worden. Eens de code door deze fase raakt, is ze klaar om te deployen. Bij Continuous Deployment worden de wijzigingen dus automatisch naar buiten gebracht (Kramer, 2018).

6.4 Automated Tests voor CI

Om te zorgen dat de automated tests aan de noden van Continuous Integration voldoen, moet er rekening gehouden worden met enkele criteria: snelheid, betrouwbaarheid, hoeveelheid en onderhoud. Bij Continuous Integration draait alles rond feedback, hierbij speelt snelheid een niet te onderschatten rol. Wanneer het runnen van de automated tests enige tijd vergt, zal de developer pas laat feedback terug krijgen waar de pipeline gefaald is. Daarom is het belangrijk rekening te houden met de test piramide bij het ontwerpen van de test omgeving. Er moet telkens een goede afweging gemaakt worden in welke categorie

elke test gestoken kan worden (Jones, 2019).

Betrouwbaarheid wordt tegenwoordig als 'normaal' beschouwd, maar dit is niet zo vanzelfsprekend. Het kan gebeuren dat de automated tests niet zo betrouwbaar zijn waardoor de developers met valse informatie moeten werken. Dit is niet bevorderlijk voor de verdere productie en het vertrouwen in een Continuous Integration en Continuous Delivery pipeline. Er zijn wel enkele tips om de automated tests betrouwbaar te maken: door de UI elements een degelijke identifier geven. Zo hoeven de testen niet de onbetrouwbare css selectors te gebruiken om aan de elementen te kunnen.

De test data van één bron van informatie voorzien is nog een belangrijke tip. Vooral in het bijzonder wanneer testen - die dezelfde data aanpassen en controleren - tegelijk runnen. Er moet ook rekening gehouden worden met de asynchrone acties bij Service en UI tests (zie de Test Pyramid in Figuur 2.3) door bepaalde situaties te vermijden. We hebben het over situaties waarbij de applicatie zich in de verkeerde staat bevindt, zodat de asynchrone test foute resultaten teruggeeft.

De hoeveelheid aan tests moet beperkt blijven om zo de snelheid van de execution times en het onderhoudsgemak te bewaren. Als men automated tests schrijft voor een CI/CD pipeline, moet men zaken testen die de integratie en het deployen van de applicatie kunnen verstoren. Ook kritieke functionaliteiten, nieuwe informatie, zaken die de voorbije builds fout liepen moeten getest worden. De testen groeperen per functionaliteit is een goede tip, zo moet er niet telkens elke functionaliteit opnieuw getest worden. Maar kan de build scheduler beslissen welke groep test moet runnen bij de nieuwe code. Angie Jones Jones (2019) geeft ook nog als tip om af en toe wat onnodige testen te verwijderen.

Voor het onderhoud van de testen moet je rekening houden met de staat van je applicatie. Deze verandert doorheen de tijd en je testen moeten deze verandering ook doorstaan.

Testing in SAPUI5

Binnen SAPUI5 wordt er gebruik gemaakt van QUnit tests en OPA5, One-Page Acceptance tests
QUnit is een JavaScript unit testing framework dat bekend staat voor zijn ou-of-the-box asynchronous capaciteiten. Dit is handig wanneer er UI functionaliteit, zoals animaties na het laden van bepaalde zaken, getest moeten worden. Omdat QUnit ook gebruik maakt van JQuery, net zoals SAPUI5, maakt dit het de perfecte tool om testen mee uit te voeren. OPA5 daarentegen bant elke vorm van asynchroon werk en kan makkelijk de elementen in de UI gebruiken om te testen. Dit maakt OPA5 de ideale tool om interactie met de gebruiker, integratie met SAPUI5 en navigatie te testen[1].

[1] https://sapui5.hana.ondemand.com/sdk

6.5 Jenkins

Uit het hoofdstuk5 hebben we in grote lijnen de vergelijking gemaakt tussen verschillende build schedulers. Jenkins kwas daar als absolute winnaar naar boven. Deze gaan we dus ook gebruiken om onze CI/CD pipeline uit te werken. In het vorige hoofdstuk hebben we de Ubuntu server helemaal klaar gezet voor het gebruik met een build scheduler. De opzet van Jenkins wordt in het hoofdstuk Continuous Integration uitgelegd in het extra document dat te vinden is bij deze thesis.

6.6 Conclusie

7. Conclusie

A. Onderzoeksvoorstel

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

A.1 Introductie

Betere, redundante projecten opleveren, dit is waar ieder bedrijf naar streeft. Amista is een consultancy bedrijf dat op zoek gaat, samen met hun klanten, naar oplossingen binnen de wereld van SAP. Bij Amista weten ze heel goed waar de noden van hun klanten liggen en zouden hier dan ook graag op inspelen. Klanten willen namelijk de gewenste veranderingen onmiddellijk te zien krijgen, de wachttijd op een oplevering die ze willen doorvoeren moet minimaal blijven. Meestal werkt een heel team aan de oplevering van een project, dit leidt tot verschillende versies van de code. Men weet niet precies op welk punt de code 'echt werkt'. Continuous Integration en Continuous Delivery kan een hulpmiddel zijn om bij elke push werkende software te hebben. Er wordt namelijk gecontroleerd of de code voldoet aan criteria om het als 'werkende' te beschouwen (de code wordt groen verklaard).

Vandaag de dag verwachten klanten dat het programma blijft werken eens een oplevering doorgevoerd wordt (dit wordt vaak 0 downtime genoemd in het vakjargon). Een manier om deze nood op te vangen is het implementeren van Continuous Deployment.

Deze studie biedt een weg aan bedrijven die denken om een CI/CD pipeline te integreren op SAP Cloud Platform om te slagen in hun opdracht. Het gaat specifiek over SAPUI5 webapplicaties die gebruik maken van SAP HANA, NodeJS en HTML5. De best practices om zo een systeem uit te rollen worden besproken, alsook wat de voor- en nadelen zijn van

bepaalde frameworks/tools.

Onderzoeksvragen:

- Wat zijn de voor- en nadelen van een CI/CD pipeline te integreren in het algemeen en specifiek voor Amista?
- Is het mogelijk om op een eenvoudige manier een Continuous Integration en Continuous Delivery pipeline te implementeren voor de ontwikkelingen van een SAPUI5 applicatie op SAP Cloud Platform?
- Hoe kunnen we deze implementatie tot een succes brengen?
- Welke tools moeten we gebruiken om een CI/CD pipeline op SAP Cloud Platform te implementeren als we vergelijken op snelheid, configureerbaarheid, documentatie en kostprijs?

A.2 State-of-the-art

SAP Cloud Platform

SAP Cloud Platform is een Platform-as-a-service (PaaS), een platform dat - door hardware en software samen te brengen - applicaties overal toegankelijk maakt en samenbrengt tot 1 platform[1]. SAP Cloud Platform is een development en deployment platform dat ook de hand reikt aan verschillende technologieën: Internet of Things, big data, Artificial Intelligence enzovoort. Het is een platform dat zowel on-premise als cloud technologieën samen kan brengen, die je kan uitbreiden en zelf kan ontwikkelen. Het haalt zijn kracht vooral uit de perfecte integratie van andere SAP software die je ook nog eens kan uitbreiden.

Continuous Integration Dit is een manier om software te maken waar de focus ligt bij de teamleden (Fowler, 2006). Zij worden verwacht hun geschreven code op regelmatige basis te integreren met de master applicatie. Op die manier gaat de code door een molen die een geautomatiseerde build zal uitvoeren en kijkt of de code slaagt voor Unit tests. Een best practice is om te testen in een kopie van de productieomgeving, zo heb je weinig risico op ongelukken. Deze techniek van implementeren brengt bepaalde voordelen met zich mee: er zijn minder problemen om de code te integreren op de master applicatie waardoor men sneller kan voldoen aan de eisen van de klant.

[1] https://cloudplatform.sap.com/index.html

Continuous Delivery De code wordt getest de testen slagen wordt de code afgeleverd in een formaat dat klaar is om te deployen. Bij deze stap is er een menselijke keuze nodig om de software naar de klant of gebruiker te brengen (Fowler, 2013).

Continuous Deployment
Dit is een manier om software uit te sturen en als klaar te beschouwen. Het is een mogelijkheid om alle soorten wijzigingen - inclusief nieuwe functies, configuratiewijzigingen, bugfixes en experimenten - op een veilige, snelle en automatische manier bij de klant of gebruiker te brengen. Deze manier van deployen is makkelijker om te voldoen aan de wijzigingen die moeten doorgevoerd worden. Er komt geen grote release bij te pas waar iedereen bang afwacht of de update stand houdt. Door telkens kleine veranderingen op een veilige manier door te voeren zorgt men ervoor dat de applicatie veel minder kans heeft op falen (Claps e.a., 2015). Het is aan te raden dat wanneer je Continuous Deployment wil implementeren in je project, je eerst Continuous Integration en Continuous Delivery op punt moet zetten. Ook is het belangrijk om een goede flow van versie control te hebben binnen het team en je moet gebruik maken van een automated deploy script die je aan de build hangt .

CI/CD integreren op SAP Cloud Platform
SAP Cloud Platform biedt de mogelijkheid om verschillende omgevingen op te stellen waarin je kan werken als developer. Het vergt enige vereisten om te voldoen aan de regels van Continuous Integration (Kramer, 2018):

- Hou alles goed bij via een version control systeem
- Automatiseer de build
- Zorg ervoor dat tijdens de build er Unit testen lopen
- Het team moet op regelmatige basis commits uitvoeren
- Elke verandering moet gebuild worden
- Als er errors tevoorschijn komen tijdens de build moeten die opgelost worden
- De build moet uitgetest worden op een kopie van de productieomgeving
- Automatiseer de deployment

Eens deze regels toegepast zijn kunnen we spreken van een CI implementatie. Vaak wordt CI in combinatie gebracht met Continuous Delivery. Om

dit in een vloeiende lijn te laten gaan spreekt men van een CI/CD pipeline.

CI/CD pipeline volgens SAP
SAP is een Duitse onderneming dat softwareoplossingen aanbiedt voor grote ondernemingen en heeft zicht gespecialiseerd in het maken van ERP pakketten. Dat is software dat alle processen van het bedrijf opneemt[2]. Een programmeur schrijft nieuwe code voor een verandering die de klant wil uitvoeren. Idealiter zou dit - voor het mergen naar de masterapplication - eens door een voter build moeten gaan, waar automatische test aanwezig zijn die kijken of de code geen problemen zou geven als je die zou mergen met de master. Een laaste stap voor de code naar de master gemerged wordt, is het toepassen van code reviews door collega developers (het 4-ogen principe). Na het samenvoegen wordt automatisch de CI-build geactiveerd. De code gaat door de automatische tests. Eens de testen slagen worden de wijzigingen geïntegreerd op de master.

Dan komt de Continuous Delivery fase, waarbij de code nog eens door een test systeem gaat. Deze fase gebeurt allemaal automatisch, maar er kunnen ook manueel testen uitgevoerd worden. Eens de code door deze fase raakt is ze klaar om te deployen. Bij Continuous Deployment worden de wijzigingen dus automatisch naar buiten gebracht (Kramer, 2018).

Tools die gebruikt kunnen worden om een CI/CD pipeline te implementeren
Er bestaan verschillende source code repositories waar je de versies van je code kan beheren. GitHub, Git, GitLab, Bitbucket zijn voorbeelden van zo een tools. Build schedulers zorgen ervoor dat de procedures worden samengesteld en dat de builds worden getriggerd. Voorbeelden hiervan zijn: Jenkins, Travis CI, GitLab-CI en Bamboo. Sonatype Nexus en Archiva zijn voorbeelden van tools die gebruikt worden als repository manager, deze houden bij wijze van spreken de code bij die klaar is om te deployen.

A.3 Methodologie

Eerst worden de voor- en nadelen van het integreren van een CI/CD pipeline in een SAPUI5 applicatie uitgeschreven. Deze studie zal de voor-

[2]https://www.sap.com/products/enterprise-management-erp.html

en nadelen van de verschillende tools onderzoeken op vlak van snelheid, configureerbaarheid met SAP en de tools die Amista gebruikt, documentatie die te vinden is online en de kostprijs. Er wordt een voorbeeldapplicatie gemaakt die zal helpen bij het uitschrijven van de best practices om een CI/CD pipeline uit te werken aan de hand van de tools die gekozen werden. De voorbeeldapplicatie wordt een omgeving waar het mogelijk is om kleine deeltjes code te wijzigen en die dan testen of ze klaar zijn om te deployen.

A.4 Verwachte resultaten

Uit onderzoek zal blijken dat de nadelen niet zullen opwegen tegen de vele voordelen die een CI/CD pipeline te bieden heeft. Amista maakt reeds gebruik van Git als versiebeheersysteem, dit zal ongetwijfeld doorwegen op de keuze. Ook het feit dat Git open source en dus helemaal gratis te gebruiken is heeft zijn voordelen. GitHub, GitLab en Bitbucket zijn allemaal een online repository management systeem om Git projecten te beheren. GitLab is open source, maar er bestaat een formule waar een onderneming moet betalen voor de diensten en server beheer. Als een onderneming gebruik wil maken van GitHub en Bitbucket zal ze geld op tafel moeten leggen. Wanneer we kijken naar de samenwerking met build schedulers, zien we dat de online repository management systemen hun eigen tool hebben. Zo heeft GitLab een eigen gemaakte Continuous Integration tool, GitLab CI genaamd. Deze tool is gratis te gebruiken tot 2000 minuten per maand[3]. Wanneer er gebruik gemaakt wordt van Bitbucket zal hoogst waarschijnlijk Bamboo gebruiken, deze twee tools komen van hetzelfde bedrijf en werken bijgevolg naadloos samen. Wanneer men GitHub gebruikt is het mogelijk om Travis CI te implementeren. Jenkins heeft dan weer plug-ins ter beschikking waarbij de samenwerking met de bovenstaande online repository management systemen verzekerd is.
Als we kijken naar de build schedulers zien we dat Jenkins een grote community heeft, het brengt geen kosten met zich mee (want het is open source) en biedt vele plug-ins aan om combinatie met andere tools makkelijk te laten verlopen. Sonatype Nexus biedt de mogelijkheid aan om te kiezen tussen de open source versie of de professionele versie die minstens $10/maand

[3]https://about.gitlab.com

kost[4]. Het verschil zit hem in de support die Sonatype biedt (OBrien, 2010). Apache Archiva is ook open source, maar daar is minder informatie over te vinden. De community is niet zo groot als bij Nexus.
In dit onderzoek wordt er ook een koppeling van de gevonden theorie aan een klein praktijkvoorbeeld gemaakt. Dit aan de hand van best practices om een Continuous Integration/Continuous Delivery pipeline op te zetten en een soort gids om die best practices toe te passen op de omgeving waar Amista mee werkt.

A.5 Verwachte conclusies

Als onderneming heeft het zeker voordelen om een CI/CD pipeline te integreren. Gaande van onmiddellijke feedback op geschreven code van developers, betere implementatie, 0 downtime. Het grootste nadeel zal de tijd van implementatie zijn, maar ook de kostprijs. De integratie en onderhoud van zo een infrastructuur brengt heel wat aanpassingen met zich mee die geld kosten. Er moeten ook veel test geschreven worden om een goede pipeline te bekomen. Het zal mogelijk zijn dat bedrijven, mits een goede handleiding van best practices bij de hand, een succesvolle CI/CD pipeline kunnen integreren. Git, Jenkins en Nexus zullen als winnaars uit de bus komen om zo de pipeline te maken. Rekening houdend met de kosten en de configureerbaarheid met de tools die Amista gebruikt en beschikbare documentatie.

[4]https://www.sonatype.com/nexus-product-pricing

Bibliografie

Baker, J. (2019). Key Research Findings. Verkregen 25 maart 2019, van https://dzone.com/guides/devops-implementing-cultural-change

Claps, G. G., Svensson, R. B. & Aurum, A. (2015). On the journey to continuous deployment: Technical and social challenges along the way. Verkregen 11 maart 2019, van https://www.sciencedirect.com/search/advanced?docId=10.1016/j.infsof.2014.07.009

Fowler, M. (2006). Continuous Integration. Verkregen 9 februari 2019, van http://www.dccia.ua.es/dccia/inf/asignaturas/MADS/2013-14/lecturas/10_Fowler_Continuous_Integration.pdf

Fowler, M. (2012, mei 1). TestPyramid. Verkregen 30 maart 2019, van https://martinfowler.com/bliki/TestPyramid.html

Fowler, M. (2013). Continuous Delivery. Verkregen 30 maart 2019, van https://martinfowler.com/bliki/ContinuousDelivery.html

Humble, J. (2012, oktober 26). Continuous Delivery. Verkregen 20 april 2019, van https://www.youtube.com/watch?v=skLJuksCRTw

Jones, A. (2019). Testing in CI. Verkregen 30 maart 2019, van https://dzone.com/guides/devops-implementing-cultural-change

Kramer, W. (2018). Continuous Integration (CI) Best Practices with SAP, CI/CD Practices. Verkregen 25 februari 2019, van https://developers.sap.com/tutorials/ci-best-practices-ci-cd.html

OBrien, T. (2010). Nexus Open Source or Professional: Which One is Right for You? Verkregen 8 december 2018, van https://blog.sonatype.com/2010/01/nexus-open-source-or-professional-which-one-is-right-for-you/

Riti, P. (2018, oktober 25). *Pro DevOps with Google Cloud Platform*. Apress, Berkeley, CA.

SAPSE. (2019). SAPUI5. Verkregen van https://developers.sap.com/topics/ui5.html

Skelton, M. (2014). The Continuous Delivery Toolchain. Verkregen 25 februari 2019, van https://dzone.com/guides/continuous-delivery-1

Smart, J. F. (2017, januari 18). Test Pyramid Heresy. Verkregen 18 mei 2019, van https://dzone.com/articles/a-test-pyramid-heresy

Tuli, S. (2018). Learn how to set up a CI/CD pipeline from scratch. Verkregen 5 mei 2019, van https://dzone.com/articles/learn-how-to-setup-a-cicd-pipeline-from-scratch

Vocke, H. (2018, februari 26). The Practical Test Pyramid. Verkregen 25 maart 2019, van https://martinfowler.com/articles/practical-test-pyramid.html