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 figuren

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 principes zoals Agile en DevOps toe te passen. Agile leidt tot een betere samenwerking tussen business en IT. DevOps gaat dan weer een stapje verder en is een mindset om de samenwerking tussen alle afdelingen binnen een IT-bedrijf vlotter te maken. 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. Een CI/CD pipeline zorgt voor de automatisatie van testen, builds en deployment en heeft als groot voordeel dat er sneller wijzigingen doorgevoerd kunnen worden, alsook de tijd dat een applicatie niet runt -'downtime' in IT genoemd- kleiner wordt.

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 dankzij het management dat gelooft in de mindset van DevOps. Van de 527 technologiespecialisten heeft bij 54% van hen de steun van het management, maar er is nog een hele weg af te leggen. Als we verder naar het onderzoek kijken, blijkt dat er een duidelijk verschil is tussen de implementatie van DevOps en de implementatie van een CI/CD pipeline. 31% heeft voor elk project een Continuous Integration pipeline en 33% heeft een CI pipeline voor enkele projecten. Slechts 14% van de ondervraagden bevestigt dat ze voor elk project aan Continuous Delivery doen, er is zelfs een daling ten opzichte van 2018. 28% heeft voor een aantal projecten zo 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

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

configuration setup, de moeilijkheden van user acceptance testing (de laatste fase binnen software testing waarbij echte gebruikers de software testen) en automated testing. 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 mogelijk om een pipeline op te stellen, maar de mindset binnen het bedrijf moet meewerken om dit tot een goed eind te brengen. Deze scriptie gaat echter niet al te diep in op hoe de DevOps cultuur binnen een bedrijf toe te passen is, maar zal vooral het technische luik onder de loep nemen. Het bedrijf Amista zou graag een gepersonaliseerde vergelijking en handleiding hebben hoe men technisch een CI/CD pipeline opstart.

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 anderen 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 de Sales, Marketing en Service Management takken van bedrijven. Ze helpen hun klanten op gebied van Innovation, Integration, HCM Services (implementatie van succesfactoren binnen de HR-afdeling) en Digital Learning. De missie van Amista is om samen met hun klanten, innovatieve en kwalitatieve oplossingen aan te 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 zo innovatief mogelijk te zijn. Ze spelen met het idee om een CI/CD pipeline op te zetten voor software development.

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 snelheid, geheugenverbruik, hoeveelheid beschikbare documentatie en confi-

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

gureerbaarheid met de huidige set-up die Amista gebruikt. 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 doen ze er alles aan om in te spelen op de noden en wensen van hun klanten. Voor projecten wordt er momenteel een datum afgesproken voor oplevering, hier komt vaak veel stress bij kijken en duiken er wel eens problemen op. Om dit te vermijden kan een Continuous Integration en Continuous Delivery pipeline helpen. Daarnaast is het ook niet makkelijk om met de huidige gang van zaken wijzigingen van klanten door te voeren. Dankzij een CI/CD pipeline kan het development team sneller wijzigingen doorvoeren en krijgen ze sneller feedback van de klant. Een voorbeeld kan dit iets duidelijker maken.

Amista heeft Appel als klant. Deze klant had in het verleden wat moeilijkheden met het opleveren van projecten en wijzigen van projecten. Daarom besloten ze om een policy toe te passen waarbij de opleveringsdata gebundeld werden en elke twee weken te releasen. Het was dus niet mogelijk om een wijziging door te voeren als de opleveringsdag verstreken was, dan moet men twee weken wachten. Appel is een internationale speler en heeft vestigingen over heel de wereld. In één van de landen is het verplicht dat grote bedrijven de belastingen online invullen. Als ze dit niet doen, krijgen ze grote boetes. De overheid voorziet hiervoor een speciaal certificaat. De tool voor de belastingaangifte stond op SAP Cloud Platform waar Amista verantwoordelijk voor is. Het certificaat wordt vernieuwd op SAP Cloud Platform op de opleveringsdatum, 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 voorschrijft 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 vernieuwd heeft en kon zorgen voor een nieuwe release. Ondertussen had Appel schrik voor de grote boete die hen boven het hoofd hing. Met een Continuous Integration en Continuous Delivery pipeline zou dit euvel in no-time opgelost kunnen worden. Met voldoende en goede testen zal de pipeline ook aangeven dat er een probleem is en zal de wijziging nooit productie halen. Het probleem had dus vermeden kunnen worden met 0 downtime. (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 worden. Op internet is niet veel informatie terug te vinden welke build scheduler nu juist het best past binnen deze omgeving, ook niet om een CI/CD pipeline op te zetten. 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 ontwikkelingen van een SAPUI5 applicatie op SAP Cloud Platform?
- Welke tools moeten we gebruiken om een CI/CD pipeline op SAP Cloud Platform te implementeren als we vergelijken op snelheid, geheugenverbruik, configureerbaarheid en hoeveelheid documentatie.
- Hoe kunnen we deze implementatie tot een succes brengen?

1.3 Onderzoeksdoelstelling

Deze thesis zou als hoofddoel een proof-of-concept zijn 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 snelheid, geheugenverbruik, configureerbaarheid met de tools die Amista gebruikt en de hoeveelheid documentatie er te vinden is.

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 en Continuous Deployment. Hier worden de verschillende lagen van een CI/CD pipeline besproken en welke tools de markt domineren per laag.

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? 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.

Er zijn heel veel build schedulers op de markt, deze worden in grote lijnen vergeleken in hoofdstuk 6.

De set-up van de omgeving wordt uitvoerig besproken in Hoofdstuk 7.

In Hoofdstuk 8 wordt er een gepersonaliseerde handleiding gegeven om met de beste build scheduler een CI/CD pipeline te implementeren.

Tenslotte wordt in Hoofdstuk 9 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

2.1 Algemeen

Bedrijven zijn constant op zoek naar betere en snellere resultaten. In het software development circuit is het vandaag soms nog lang wachten voor een wijziging effectief doorgevoerd wordt. Men levert nog vaak software op aan het einde van de sprint, wat soms voor problemen zorgt als men veel code tegelijk aflevert. Met een nieuwe software development methode - Continuous Integration en Continuous Delivery genaamd - wil men deze problemen zoveel mogelijk vermijden. 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.

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 gebruiken om de communicatie tussen developers en systeembeheerders beter te maken. 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, integra-

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

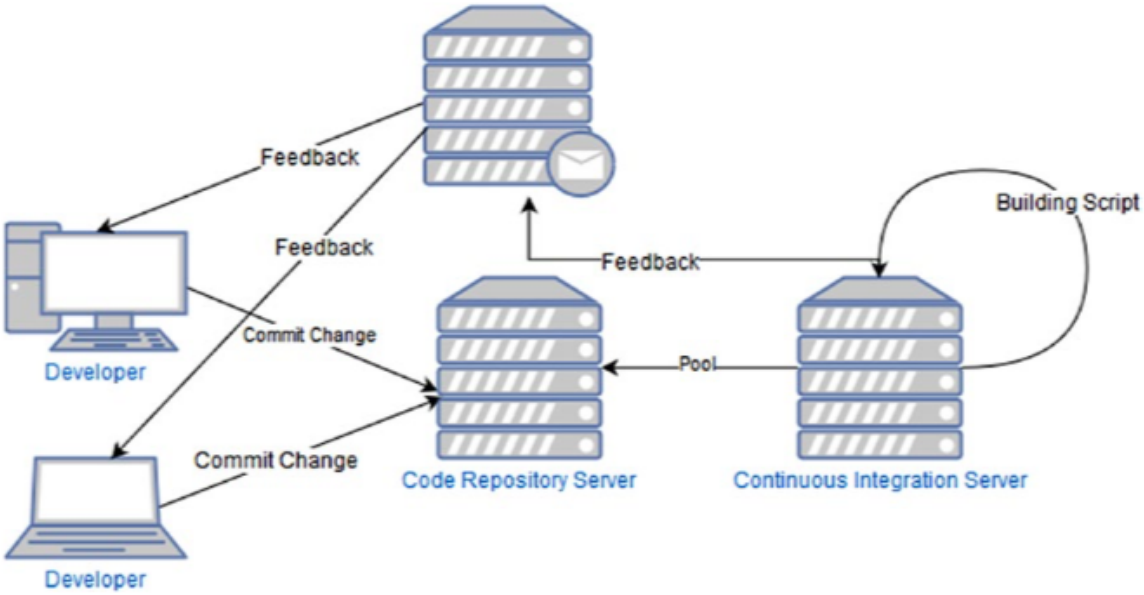
Continuous Integration

Dit is een eerste stap in de pipeline waarbij de geschreven code wordt gecommit naar een 'repository management server'. Hierdoor wordt de 'source code' automatisch opnieuw opgebouwd en slaagt voor de testen die automatisch zijn opgestart. 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 verliezen 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 dit voorbeeld terug te vinden. De developer maakt een wijziging en commit de code naar de repository die te vinden is op het source-control systeem. De Continuous Integration server krijgt het bericht dat er code is toegevoegd, haalt de laatst toegevoegde code op en laat de 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 elke keer er code naar de repository gestuurd wordt. Bovenstaand voorbeeld 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 te doen. Hij moet telkens een commit uitvoeren 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 de feedback. Dit moet namelijk het resultaat van de build en de feedback waar het probleem zich kan bevinden terug geven, door bijvoorbeeld aan te geven welke test niet slaagt bij het uitvoeren van de automated testing. 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.

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



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

grafische weergave van de stappen die een Continuous Delivery pipeline moet hebben. 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. Deze maakt ook de build klaar voor release en 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. 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 verbonden aan de pipeline omgeving. Het geeft feedback over de staat van de CI/CD pipeline en is een vorm van monitoring. Het geeft de developers en iedereen in de kamer onmiddelijke feedback wanneer er een commit gebeurd. 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 is. Zo weet de persoon die de foute code gepusht heeft dat er iets fout is en kan hij hier nog sneller op inspelen.

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

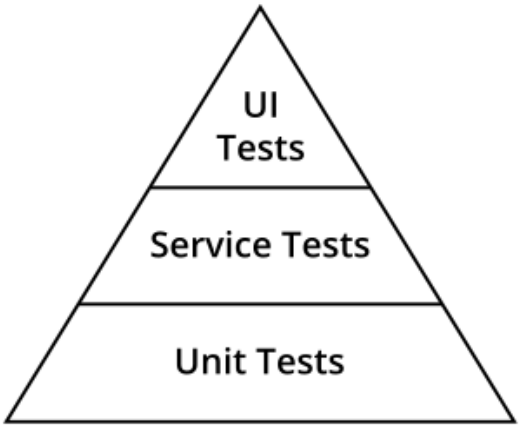
Figuur 2.2: Continuous Delivery chain (Riti, 2018)

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.

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, wordt er een veiligheidsgevoel gecreëerd. 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. 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. 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 gedetailleerd (high-level testing) gaat. Omdat bij high-level testen de requirements goed begrepen zijn door het test team. 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 service tests - ook wel Integration tests genoemd - focust vooral op de functionaliteit van de applicatie. Deze testen leggen de nadruk op de functionaliteit dat de applicatie moet bevatten. 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



Figuur 2.3: Test piramide (Vocke, 2018)

(Fowler, 2012).

2.2 Lagen binnen een CI/CD pipeline en beschikbare tools

Er bestaan verschillende repository management services die een source control systeem hebben. GitHub, GitLab en Bitbucket zijn enkele voorbeelden. 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 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.

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. Best practice is er maar 1 versie van de software die stabiel is, 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.

Repository Management 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 en is nodig om een goede Continuous Integration aan te bieden.

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 pipeline wordt uitgevoerd. De taken van de build scheduler zijn:

- 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).

Artifact Repository Manager

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

- packaged application code
- application assets
- infrastructure code
- virtual machine images
- configuration 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).

3. Voor- en nadelen CI/CD pipeline

Dit gaat over de algemene voordelen van zo een pipeline, maar ook specifiek van Amista. Om een antwoord te krijgen op de algemene voordelen wordt er als basis teruggegrepen naar de literatuur. Er is namelijk al enorm veel geschreven over een CI/CD pipeline en wat de voor- en nadelen kunnen zijn. Bijkomstig is er een interview afgenomen met de expert omtrent DevOps, Patrick Debois. Het interview en de literatuur bieden samen de perfecte combinatie om de voordelen en nadelen te bespreken. Op de vraag wat de voor- en nadelen voor Amista zullen zijn kan enkel iemand van Amista op antwoorden.

3.1 Voordelen

- Maak de applicatie die de klant wil
- Als men kleine stukjes code commit naar de centrale repository en de testen slagen niet, weet men dat het aan dit klein deeltje van de code ligt en kan men de fout veel sneller opsporen.
- In de meeste omgevingen met een CI/CD pipeline is het niet nodig om een QA team te hebben om de testen uit te voeren, wat de kosten doet dalen.
- Het risico op downtime van een applicatie wordt naar beneden gehaald. Door de vele testen die automatisch gerund worden alvorens de aanpassing aan de code naar buiten wordt gebracht, verkleint de kans dat een fout niet herkend wordt. Dit is wel onder de voorwaarde dat de testen in een soortgelijke omgeving 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 meestal ook een team van wacht om problemen, die zouden kunnen opduiken, op te lossen

3.2 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 wat werk kruipen in het opzetten van een CI/CD pipeline. Dit gaat gepaard met werkuren, die geld zullen 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 lagen zal testen.
- 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.

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 zicht 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 grof 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 online[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 kan er de

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

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.

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 gemaapkt 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 is bedoeld om dezelfde lay-out en hetzelfde gebruik voor de eindklant te garanderen. Het biedt aan de developers een resem aan UI controls aan, zodat er een consistenter en beter UX design gehanteerd wordt. (SAPSE, 2019)

SAP HANA

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 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[3].

[3]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

Amista heeft enkele zaken aangehaald die zeker aanwezig moeten zijn. Deze zijn onderverdeeld in 2 categorieën: functioneel en niet-functioneel. In elke categorie wordt dan weer onderscheid gemaakt tussen: zaken die zeker aanwezig moeten zijn, taken die de tool zou moeten kunnen en enkele nice-to-haves. Er wordt een lijst met build-schedulers gemaakt en hoe deze aan de criteria voldoen. Op het einde wordt een korte conclusie gemaakt wat voor Amista de beste build scheduler is om een CI/CD pipeline te integreren in hun software development?

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 binnen SAP uitgelegd, in dit hoofdstuk worden

de onderliggende relaties tussen de tools besproken. Amista heeft een Ubuntu server ter beschikking gesteld om te experimenteren. Voor deze thesis wordt de server gebruikt als Continuous Integration server, zo kunnen de build schedulers op de beste manier vergeleken worden zonder veel externe factoren.

5.3 Proof-Of-Concept

In dit hoofdstuk worden alle stappen die gebeurd zijn tijdens het opzetten van de voorbeeldapplicatie en de CI/CD server beschreven aan de hand van tekst en afbeeldingen. Het is perfect mogelijk om aan de hand van de info uit dit hoofdstuk de omgeving opnieuw te reproduceren.

6. Vergelijking tussen build schedulers

Hier worden de verschillende criteria die Amista aangaf als belangrijk en minder belangrijk opgedeeld in twee delen: de functionele requirements 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 hoofdstuk8 gebeuren.

6.1 Functionele requirements

Een functionele requirement is een functie wat het systeem moet doen. De punten die in deze sectie worden aangehaald komen uit hoofdstuk**??**. 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 omdat er een fout in de code zit die gedeployed wordt

6.2 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.

6.3 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 hoofdstuk8. 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. Omdat ze bij Amista met Bitbucket werken is het vanzelfsprekend dat de build scheduler ook kan integreren met deze Code Repository.

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

- Hij moet kunnen werken 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

6.4 Should-Haves

6.5 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 wel openhouden voor de toekomst. Daarom is het mooi meegenomen als de gekozen build scheduler aan deze vereiste voldoet, maar het is geen breekpunt.

Build scheduler	SAPUI5, HANA & Cloud Platform	Git & Bitbucket	Unix-based server	Cloud toepassing
Bamboo		Yes		
Circle CI				
Jenkins		Yes	Yes	
Travis				

6.6 Long list

Nu we de verschillende criteria gekend zijn kan de effectieve vergelijking gebeuren van de build schedulers.

7. Voorbeeldapplicatie

In dit hoofdstuk wordt er stap voor stap beschreven hoe de opstelling is opgebouwd om de vergelijking te HANA tussen de verschillende build schedulers.

7.1 Build scheduler server

Amista heeft een Ubuntu server ter beschikking gesteld dat wordt gehost op Digital Ocean. De server wordt gebruikt om de build schedulers op te draaien en zo te vergelijken. Eerst moeten er enkele belangrijke zaken ingesteld worden alvorens aan de slag te gaan, zoals security en dergelijke.

Digital Ocean

Ubuntu server

Installatie Ubuntu server

Eerst HANA we de Ubuntu server klaar voor gebruik om zo de nodige zaken te installeren. Via de root account wordt er via SSH ingelogd op de server.

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. Eens ingelogd op de sshd server moeten er enkele zaken aangepast worden aan de ssh configuratie in de file /etc/ssh/sshd_config. Omdat we hier via de root gebruiker werken moet de property PermitRootLogin op yes staan. Dit zorgt ervoor dat de root gebruiker kan inloggen. StrictMode moet ook op yes staan, zo kan er niemand inloggen als de authenticatie documenten leesbaar zijn voor iedereen. Dit voor het beveiligen van configuratie documenten.

De default manier om in te loggen via ssh 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 public key. 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 public key op de client wanneer deze stap wordt geconfigureerd. De private key moet veilig bewaard worden op de client computer. De public key moet doorgegeven worden aan de remote server. 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. In deze thesis gaat men ervan uit dat de client computer een ssh key heeft die gebruikt kan worden. 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. Voor deze thesis en om veiligheidsredenen is het beter om enkel via key-based authenticatie in te loggen en het paswoord uit te sluiten. Nu moet de sshd_config file opgeslagen worden (x̂ + y + enter) en de ssh daemon herstart worden door het commando sudo systemctl restart ssh in te geven.

Om de remote server nog meer te beschermen tegen cyber aanvallen is het nodig om een firewall op te zetten. In deze voorbeeldapplicatie HANA we gebruik van de UFW Firewall. Dis staat voor Uncomplicated Firewall en is een gebruiksvriendelijke tool dat helpt om de iptables onder controle te houden om zo te zorgen dat bepaalde services toegelaten worden tot onze server. In Linux HANA ze gebruik van het protocol SSH via de service OpenSSH, deze heeft ook een profiel bij UFW.

Nu alle stappen voor de configuratie van de server gedaan zijn is het zeer makkelijk om in te loggen op de server. Het is hetzelfde als de eerste keer, maar nu vraagt de server niet meer naar een paswoord, maar gebruikt hij de ssh-key. Het is voldoende om ssh root@188.166.61.128 te typen om in te loggen. Als je wil uitloggen is het nodig om in de command line van de server exit te typen.

7.2 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 en wordt daarom ook wel veel gebruikt. Voor de database HANA we gebruik van een Multi-Target Application Project, dit is een template die SAP ons geeft en is een goede uitvalsbasis om te gebruiken in de voorbeeldapplicatie. Zoals eerder al aangegeven is een Source Code Repository van groot belang voor een CI/CD pipeline en development in het algemeen. Eens de repository aangemaakt is heb je het webadres nodig om de clone te HANA op je lokale machine.

De volgende stap is het project aanmaken. In de Web IDE voor HANA development is het belangrijk om eerst enkele instellingen aan te passen. Zoals eerder vermeld HANA we het project aan de hand van het Multi-Target Application Project template. na de creatie van het project is het nodig om een build uit te voeren. (figuur InitialBuildHANAProject) Om het project aan het account op Bitbucket te linken klikt u op het project met de rechter muisknop, gaat u naar Git en dan Initialize Local Repository. (figuur InitializeLocalRepositoryHANA) Nu linken we de gemaakte repository aan het project door via rechter muisklik op het project op Git en dan Set Remote te klikken. Hier moet je de gekopieerde bitbucket link plakken in het veld voor URL. (figuur ConfigureGitRepositoryHANA) Na het ingeven van het juiste wachtwoord, is het nodig om op OK te klikken wanneer het Changes Fetched window opent. Na deze stap is het project gelinkt met de repository op Bitbucket. Het maken van het project heeft voor changes gezorgd in de repository, deze moeten eerst gecommit en gepusht worden.

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 gaan bestaat uit twee entiteiten: een Artiest en een Album. Een artiest kan meerdere albums hebben, maar een album kan maar tot één artiest behoren. De entiteit Artiest heeft volgende properties: ID, Naam, JaarVanOorsprong en Stad. Album heeft ID, Naam, Beschrijving, Genre, Jaar en Studio als properties. Als basis maken we de entiteit Artiest enkel met ID, Naam en JaarVanOorsprong. De rest zal later toegevoegd worden aan de hand van de pipeline. Eens deze gegevens ingevoerd zijn moeten we alle veranderingen toevoegen aan de commit om dan te pushen naar de repository.

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. het is nodig om op de data CRUD (Create, Read, Update en Delete) operaties uit te voeren en als OData naar buiten gaat, dit is de taak van XSODATA. XSJS zorgt dan weer voor de integratie met SAPUI5. Het laat toe dat SAPUI5 applicaties de data kan lezen en kan bewerken. Zoals u in Om een OData service te maken moeten volgende stappen gebeuren: in de js-module in de lib folder, moet een xsodata-file gemaakt worden in een nieuwe folder, xsodata genaamd. De link met de data moet in deze nieuwe file gemaakt worden. Later zal er ook de associatie moeten gemaakt worden tussen Artiest en Album. Deze stappen zijn te zien in De volgende stap is de xsjs service maken door in de lib folder een nieuwe folder, xsjs genaamd, te maken met de nieuwe hdb.xsjs file. Zie De nieuwe Node.js module moet gebuild worden door op de module met de rechtermuisklik te klikken, Run,

Run As en dan Node.js Application te kiezen.

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

Uit vorig hoofdstuk is gebleken dat SCHRAPPEN WAT NIET PAST: Jenkins & Travis de tools zijn die aan het meeste must-haves, should-haves en nice-to-haves voldoen. In dit hoofdstuk worden deze tools extra onder de loep genomen door ze op te stellen in een realistische omgeving zoals in hoofdstuk/refch:voorbeeldapplicatie besproken is. Door de tools te vergelijken in zo'n omgeving krijgen we een meer realistisch beeld welke build-scheduler het beste zal functioneren in deze omgeving. Op het einde van dit hoofdstuk zal er een conclusie worden gemaakt welke build-scheduler Amista moet kiezen en waarom. Maar beginnen doen we met enkele tips te geven van SAP hoe een CI/CD pipeline opgestart moet worden.

8.1 Continuous Delivery principles

8.2 CI/CD pipeline 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 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.

8.3 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 (SAPERP, 2019). 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).

8.4 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 onderhouden is ook een belangrijke tip. Dit kan door één bron van informatie te voorzien voor test data, vooral in het bijzonder wanneer testen - die dezelfde data aanpassen en controleren - tegelijk runnen. Rekening houden met de asynchrone acties bij Service en UI tests uit de Test Pyramid (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, de hoge waarde van tests 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 mee 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 teste moeten deze verandering ook doorstaan.

8.5 Conclusie

9. 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

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.

SAPERP. (2019). Proven, time-tested on-premise ERP. Verkregen van https://www.sap.com/products/enterprise-management-erp.html

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

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