



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

<b>1</b>	<b>Inleiding</b>	<b>13</b>
1.1	Probleemstelling	15
1.2	Onderzoeksvraag	16
1.3	Onderzoeksdoelstelling	16
1.4	Opzet van deze bachelorproef	16
<b>2</b>	<b>Continuous Integration, Continuous Delivery &amp; Continuous Deployment</b>	<b>19</b>
2.1	Algemeen	19
<b>3</b>	<b>CI/CD pipeline</b>	<b>25</b>
3.1	Continuous Integration en Continuous Delivery pipeline	25
3.2	Lagen binnen een CI/CD pipeline en beschikbare tools	26
3.2.1	Source Control System	26

3.2.2	Hosted Version Control Services .....	26
3.2.3	Build Scheduler .....	27
3.2.4	Artifact Repository Manager .....	27
3.3	<b>Voordelen</b>	27
3.4	<b>Nadelen</b>	28
<b>4</b>	<b>SAP .....</b>	<b>31</b>
<b>5</b>	<b>Methodologie .....</b>	<b>33</b>
5.1	<b>Vergelijkende studie van de build schedulers</b>	33
5.1.1	Build Scheduler .....	34
5.1.2	Requirements .....	34
5.1.3	Criteria .....	35
5.1.4	Long list .....	37
5.2	<b>Voorbeeldapplicatie dat Amista zal gebruiken</b>	41
5.2.1	Build scheduler server .....	41
5.2.2	Database .....	46
5.2.3	Automated testing binnen SAPUI5 .....	55
<b>6</b>	<b>Proof-of-concept van een CI/CD pipeline .....</b>	<b>61</b>
6.1	<b>Continuous Delivery principles</b>	61
6.2	<b>CI/CD pipeline op SAP Cloud Platform</b>	62
6.3	<b>CI/CD pipeline volgens SAP</b>	62
6.4	<b>Automated Tests voor CI</b>	62
6.5	<b>Short List</b>	63
6.6	<b>Conclusie</b>	66

<b>7</b>	<b>Conclusie .....</b>	<b>73</b>
<b>A</b>	<b>Onderzoeksvoorstel .....</b>	<b>75</b>
A.1	Introductie	75
A.2	State-of-the-art	76
A.3	Methodologie	78
A.4	Verwachte resultaten	79
A.5	Verwachte conclusies	80
	<b>Bibliografie .....</b>	<b>81</b>



## Lijst van tabellen

5.1	Long list CI/CD tools, nummer 1 .....	38
5.2	Long list CI/CD tools, nummer 2 .....	39
5.3	Long list CI/CD tools, nummer 3 .....	40



# 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 Figuur 1.1 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<sup>1</sup>. 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-

<sup>1</sup><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 verstrekken 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, configurerbaarheid 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, configurerbaarheid 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 onderzoeks vragen. 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 gegeven 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 onderzoeks vragen. Daarbij wordt ook een aanzet gegeven voor toekomstig onderzoek binnen dit domein.



## 2. Continuous Integration, Continuous Delivery

### 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-

## **Hoofdstuk 2. Continuous Integration, Continuous Delivery & Continuous Deployment**

20

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**

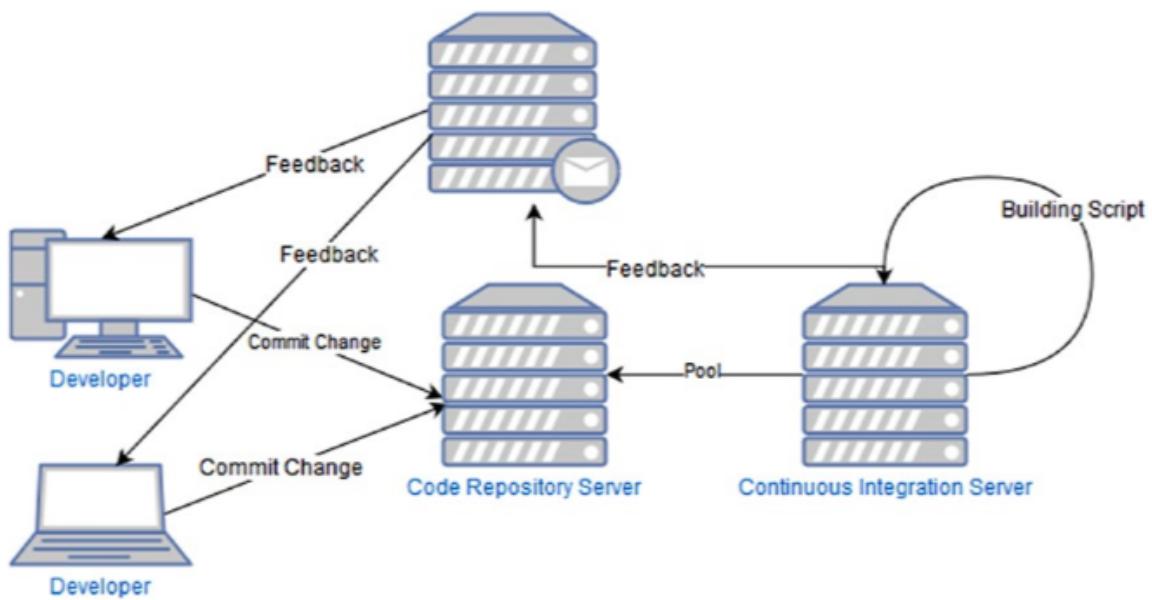
Dit is een eerste stap in de pipeline waarbij de geschreven code wordt gecommit naar een 'repository management server'. Hierdoor wordt de 'source code' automatisch opnieuw opgebouwd en slaagt voor de testen die automatisch zijn opgestart. De focus bij deze stap ligt bij de teamleden (Fowler, 2006), 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



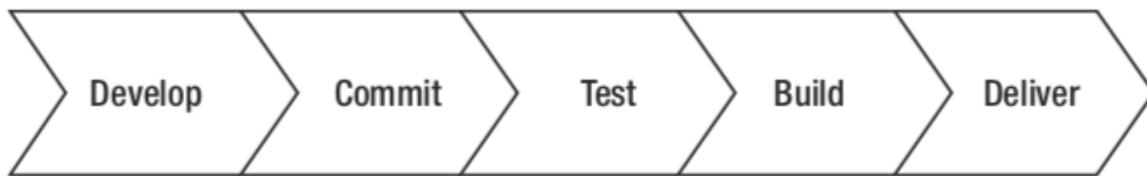
Figuur 2.1: Voorbeeld van een Continuous Integration set-up (Riti, 2018)

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 ontwikkelen 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 gebouwd 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 eeuvel 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 onmiddellijke 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



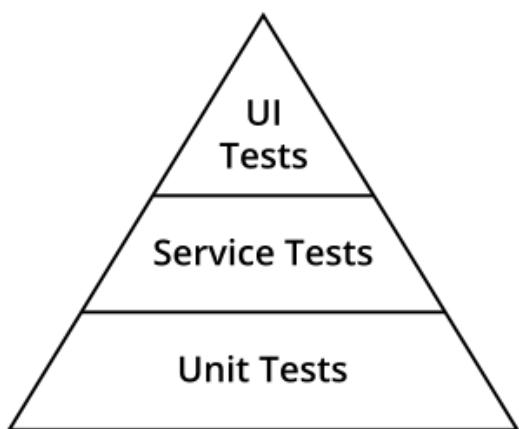
Figuur 2.2: Continuous Delivery chain (Riti, 2018)

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).

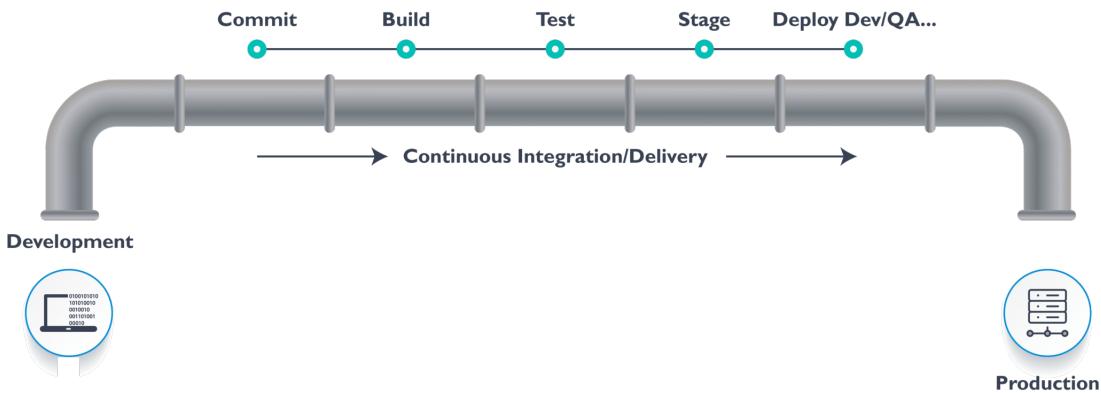


## 3. CI/CD pipeline

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 besproken. Er wordt een onderscheid gemaakt tussen de algemene kenmerken en die specifiek voor Amista. 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 waarden en de schaduwzijde van een CI/CD pipeline voor Amista werden 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. De pipeline stelt een proces voor dat verschillende fases doormaakt. Het is een samentrekking tussen het proces van Continuous Integration en Continuous Delivery waarbij alles automatisch wordt aangestuurd. In onderstaande figuur3.1 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 tijdens deze stap zorgt de build scheduler ervoor dat de unit testen uitgevoerd zijn alvorens de build begint. Eens de build slaagt is het de taak van de build scheduler 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



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

of deze voldoet aan de kwaliteitseisen. Eens de code door de testen raakt wordt de commit klaar gemaakt voor productie die deze in de stage fase te brengen. Eens de code zich in de stage fase bevindt kan deze manueel worden gedeployed om naar productie 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 lagen hebben elk hun taken om uit te voeren en hebben ook hun eigen tools waaruit men kan kiezen om te gebruiken.

### 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. 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.

### 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 cruciaal 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 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).

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 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). 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 slechte kenmerken van de software. 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. 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 van 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). Als men dagen aan een stuk programmeert zonder deftige feedback te krijgen op de geschreven code kan het gebeuren dat de geschreven code vernietigd moet worden, omdat deze is voortgebouwd op een fout. Om dit verlies tegen te gaan is een CI/CD pipeline de perfecte oplossing.
- 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 voor verdere implementatie.
- 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

### 3.4 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 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 kruipst 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.
- 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



## 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 bedrijfsssoftware. SAP heeft zich gespecialiseerd in ERP, Enterprise Resource Planning software dat alle processen van het bedrijf automatiseert<sup>1</sup>. 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<sup>2</sup>. 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

<sup>1</sup><https://www.sap.com/products/enterprise-management-erp.html>

<sup>2</sup><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 gemaapt 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 consistentere 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 aangepresent als een platform om ingewikkelde, real-time analytische berekeningen uit te voeren op data. Het is een Relationale Database Management Systeem (RDBMS) 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<sup>3</sup>.

---

<sup>3</sup><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 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 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, gekeken of er veranderingen zijn aangebracht in de repository. Wanneer de build scheduler een verandering merkt 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. Het is van groot belang om een build scheduler te kiezen die de gebruikte testing tool kan integreren.

Het is de taak van de build scheduler om naadloos samen te 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. De build scheduler heeft ook als belangrijke taak goede feedback te bezorgen aan de developers indien testen niet slagen. 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 opgeliist 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 hoofdstuk3. 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 hoofdstuk6. 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.

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 wel 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.

### 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.

#### 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 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 armen heeft.

#### 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.

	Must-have	SAP Cloud Platform & UI5	Git	Bitbucket	On premise	CI/CD pipeline	Automated tests	OPA tests
Jenkins	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Geen info
Circle CI	Geen info	Ja	Ja	Ja	Ja	Ja	Ja	Geen info
Bamboo	1 blogpost	Ja	Ja	Ja	Ja	Ja	Ja	Geen info
Travis CI	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Geen info

Tabel 5.1: Long list van vergelijking CI/CD tools, nummer 1

	Must-have			Should-haves		
	Gebruiksgemak	Auditing	Meerdere releases / maand	Prijs	Trial	
Jenkins	Makkelijk	Ja	Ja	Gratis	Gratis	
Circle CI	Makkelijk	Basic	Meestal	\$35 / user	20 dagen	
Bamboo	Middelmatig	Ja	Nee	\$1100 / jaar voor 1 remote agent en ongelimiteerd aantal jobs	30 dagen	
Travis CI	Makkelijk	Ja	Ja	\$2739 / jaar voor 5 jobs	Gratis voor open source	

Tabel 5.2: Long list van vergelijking CI/CD tools, nummer 2

	Should-haves			Nice-to-haves	
	Skype for Business	Container support	Slack	Rapport	
Jenkins	Ja	Ja	Ja	Ja	Ja
Circle CI	Nee	Ja	Ja	Ja	Ja
Bamboo	Nee	Ja	Ja, via derde partij	Nee	
Travis CI	Ja	Ja	Ja	Ja	Ja

Tabel 5.3: Long list van vergelijking CI/CD tools, nummer 3

## 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.

### 5.2.1 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 wanneer men spreekt over web-hosting computers. Ze zijn gespecialiseerd in cloud based oplossingen en hebben datacenters over heel de wereld verspreid. 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.

#### Ubuntu server

Ubuntu is een Linuxdistributie, gebaseerd op het bekende Debian en gekend is vanwege de open-source eigenschappen. Deze versie van Linux wordt vooral gebruikt voor cloud computing, vandaar dat Digital Ocean voor 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<sup>6</sup> een server voorzien met een x64-bit Operating System waar 1 CPU en 1GB RAM ter beschikking staat.

#### Installatie Ubuntu server

Eerst maken we de Ubuntu server klaar voor gebruik om zo de nodige zaken te installeren. In figuur 5.1 in de bijlagen wordt er getoond wat er moet gebeuren als er voor de eerste keer ingelogd wordt op de server. 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

```
[MacBook-Pro-van-Renaaat:~ Renaat$ ssh root@188.166.61.128
The authenticity of host '188.166.61.128 (188.166.61.128)' can't be established.
ECDSA key fingerprint is SHA256:rKRV6j2yQuxkI709abiHGxeGlYungYoPd90pU69E68Q.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '188.166.61.128' (ECDSA) to the list of known hosts.
[root@188.166.61.128 ~]# password:
You are required to change your password immediately (root enforced)
Welcome to Ubuntu 18.04.2 LTS (GNU/Linux 4.15.0-46-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

System information as of Sat Apr  6 08:01:31 UTC 2019

System load:  0.0          Processes:      83
Usage of /:   6.6% of 24.06GB  Users logged in:   0
Memory usage: 16%
Swap usage:   0%
IP address for eth0: 188.166.61.128

Get cloud support with Ubuntu Advantage Cloud Guest:
  http://www.ubuntu.com/business/services/cloud

10 packages can be updated.
0 updates are security updates.

*** System restart required ***

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/*copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

Changing password for root.
[(current) UNIX password:
[Enter new UNIX password:
[Retype new UNIX password:
root@ubuntu-s-1vcpu-1gb-ams3-01:~# ]
```

Figuur 5.1: Opzetten Server, stap 1

```
$OpenBSD: sshd_config,v 1.101 2017/03/14 07:19:07 djm Exp $  
  
# This is the sshd server system-wide configuration file. See  
# sshd_config(5) for more information.  
  
# This sshd was compiled with PATH=/usr/bin:/bin:/usr/sbin:/sbin  
  
# The strategy used for options in the default sshd_config shipped with  
# OpenSSH is to specify options with their default value where  
# possible, but leave them commented. Uncommented options override the  
# default value.  
  
#Port 22  
#AddressFamily any  
#ListenAddress 0.0.0.0  
#ListenAddress ::  
  
#HostKey /etc/ssh/ssh_host_rsa_key  
#HostKey /etc/ssh/ssh_host_ecdsa_key  
#HostKey /etc/ssh/ssh_host_ed25519_key  
  
# Ciphers and keying  
#RekeyLimit default none  
  
# Logging  
#SyslogFacility AUTH  
#LogLevel INFO  
  
# Authentication:  
  
#LoginGraceTime 2m  
PermitRootLogin yes  
StrictModes yes  
#MaxAuthTries 6  
#MaxSessions 10
```

Figuur 5.2: Opzetten Server, stap 2

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. Deze configuratie kan u zien in figuur 5.2 hier onder.

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. Zoals u in figuur 5.3 kan zien heeft de client die gebruikt werd tijdens het schrijven

```
[MacBook-Pro-van-Renaat:~ Renaat$ cd .ssh/
[MacBook-Pro-van-Renaat:~.ssh Renaat$ ls -l
total 24
-rw----- 1 Renaat  staff  1675 Oct  3  2017 id_rsa
-rw-r--r-- 1 Renaat  staff   403 Oct  3  2017 id_rsa.pub
-rw-r--r-- 1 Renaat  staff  2079 Apr  6 10:01 known_hosts
```

Figuur 5.3: Opzetten Server, stap 3

```
MacBook-Pro-van-Renaat:~ Renaat$ ssh-copy-id root@188.166.61.128
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/Users/Renaat/.ssh/id_rsa.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
root@188.166.61.128's password:

Number of key(s) added:      1

Now try logging into the machine, with: "ssh 'root@188.166.61.128'"
and check to make sure that only the key(s) you wanted were added.
```

Figuur 5.4: Opzetten Server, stap 4

van deze thesis enkel lees- en schrijfrechten voor de file 'id\_rsa', dit om het als secret te bewaren. 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. In figuur5.4 in de bijlagen is te zien hoe de ssh session wordt opgezet tussen de client en de remote server voor de root user. Voor deze thesis en om veiligheidsredenen is het beter om enkel via key-based authenticatie in te loggen en het paswoord uit te sluiten. In figuur5.5 is te zien welke aanpassingen in de sshd\_config file van de sshd server moeten gebeuren om het niet meer mogelijk te maken om in te loggen via een paswoord. PasswordAuthentication en ChallengeResponseAuthentication moeten naar no verandert worden. PublicKeyAuthentication moet naar yes verandert worden. Nu moet de 'sshd\_config' file opgeslagen worden ( $\hat{x} + y + \text{enter}$ ) 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 maken 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 maken ze gebruik van het protocol SSH via de service OpenSSH, deze heeft ook een profiel bij UFW. In Figuur5.6 in de bijlagen is te zien hoe de firewall de SSH service toelaat. Het is enkel mogelijk om de server te bereiken via deze service. Later worden er uiteraard meerdere services toegelaten.

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.

```

# Logging
#SyslogFacility AUTH
#LogLevel INFO

# Authentication:

#LoginGraceTime 2m
PermitRootLogin yes
StrictModes yes
#MaxAuthTries 6
#MaxSessions 10

PubkeyAuthentication yes

# Expect .ssh/authorized_keys2 to be disregarded by default in future.
#AuthorizedKeysFile      .ssh/authorized_keys .ssh/authorized_keys2

#AuthorizedPrincipalsFile none

#AuthorizedKeysCommand none
#AuthorizedKeysCommandUser nobody

# For this to work you will also need host keys in /etc/ssh/ssh_known_hosts
#HostbasedAuthentication no
# Change to yes if you don't trust ~/.ssh/known_hosts for
# HostbasedAuthentication
#IgnoreUserKnownHosts no
# Don't read the user's ~/.rhosts and ~/.shosts files
#IgnoreRhosts yes

# To disable tunneled clear text passwords, change to no here!
PasswordAuthentication no
#PermitEmptyPasswords no

# Change to yes to enable challenge-response passwords (beware issues with
# some PAM modules and threads)
ChallengeResponseAuthentication no

```

Figuur 5.5: Opzetten Server, stap 5

```

[root@ubuntu-s-1vcpu-1gb-ams3-01:/etc/ssh# ufw app list
Available applications:
  OpenSSH
[root@ubuntu-s-1vcpu-1gb-ams3-01:/etc/ssh# ufw allow OpenSSH
Skipping adding existing rule
Skipping adding existing rule (v6)
[root@ubuntu-s-1vcpu-1gb-ams3-01:/etc/ssh# ufw enable
[Command may disrupt existing ssh connections. Proceed with operation (y|n)? y
Firewall is active and enabled on system startup
[root@ubuntu-s-1vcpu-1gb-ams3-01:/etc/ssh# ufw status
Status: active

To                         Action      From
--                         -----      -----
OpenSSH                      ALLOW      Anywhere
OpenSSH (v6)                  ALLOW      Anywhere (v6)

```

Figuur 5.6: Opzetten Server, stap 6

The screenshot shows the 'Create a new repository' interface. The 'Repository name' field contains 'Bachelorproef\_Hana\_Database'. The 'Access level' section has a checked checkbox for 'This is a private repository'. The 'Include a README?' dropdown is set to 'Yes, with a template'. Under 'Version control system', 'Git' is selected. A 'Description' box contains the text: 'This is the repository in Bitbucket for the Hana 2.0 Database.' The 'Forking' setting allows only private forks. In 'Project management', neither 'Issue tracking' nor 'Wiki' is checked. The 'Language' dropdown says 'Select language...'. At the bottom are 'Create repository' and 'Cancel' buttons.

Figuur 5.7: Source Control, stap 1

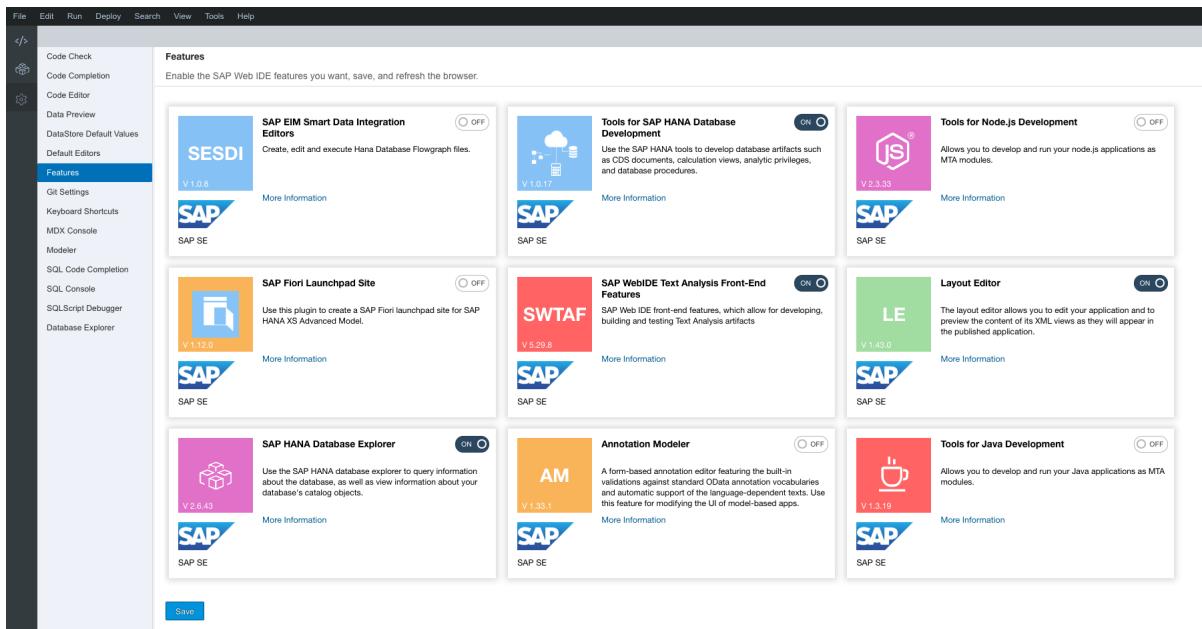
## 5.2.2 Database

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 en wordt daarom ook wel veel gebruikt. Voor de database maken 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.

### Source Control & Databank module

In figuur 5.7 kan u waarnemen hoe een repository gemaakt wordt in Bitbucket. Eens de repository aangemaakt is heb je het webadres nodig om de clone te maken op je lokale machine.

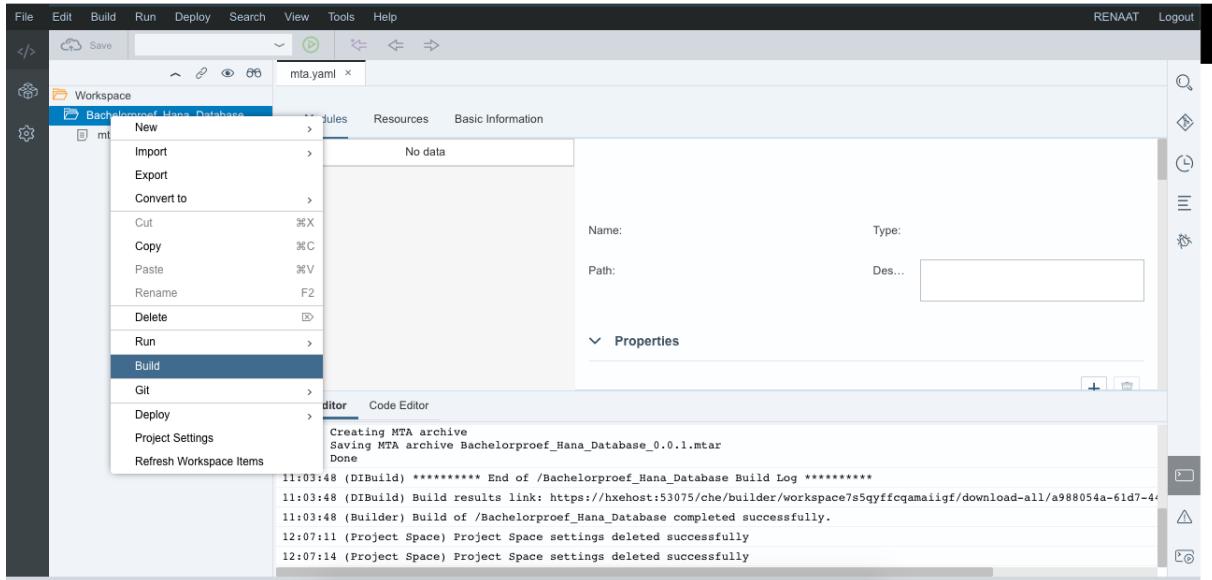
De volgende stap is het project aanmaken. In de Web IDE voor HANA development is het belangrijk om eerst enkele instellingen aan te passen. Alle nodige features die nodig zijn



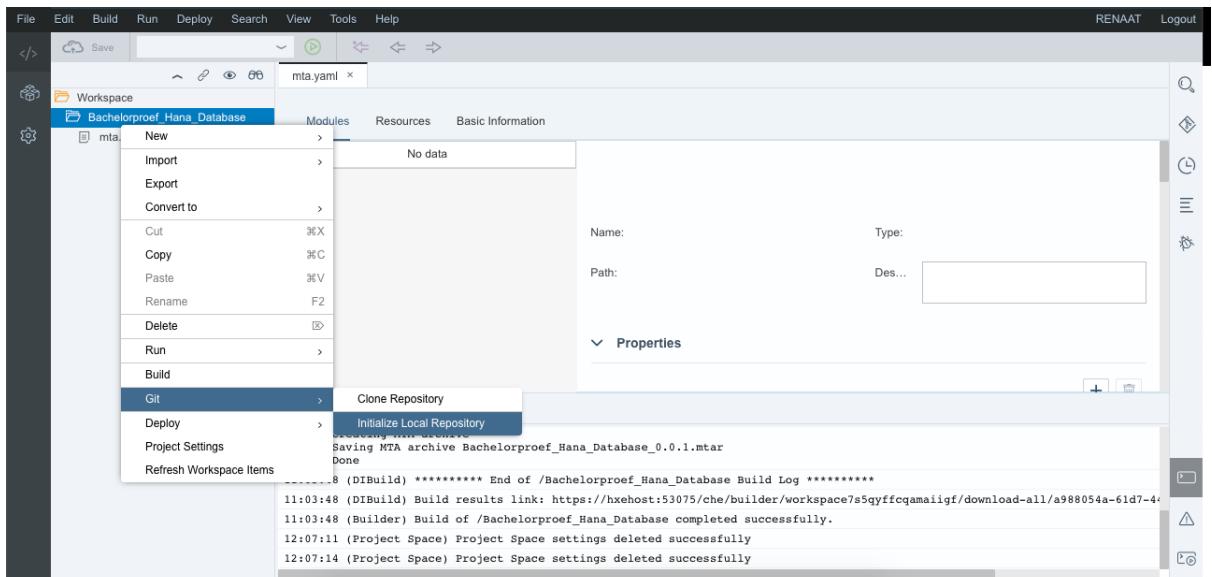
Figuur 5.8: Opzetten HANA, stap 1

kan u terug vinden in figuur5.8. Nu is het ook belangrijk om je Git account te connecteren met de Web IDE door het Git email adres en Git username in te voeren in de Git Settings. Zoals eerder vermeld maken 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 (figuur5.9). 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' (figuur5.10). 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. Na het ingeven van het juiste wachtwoord, is het nodig om op 'OK' te klikken wanneer het 'Changes Fetched' window opent. Deze stappen zijn te volgen in figuren 5.11 en 5.12. 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. In figuur5.13 is te zien wat er allemaal moet gebeuren om een commit en push te doen naar de repository.

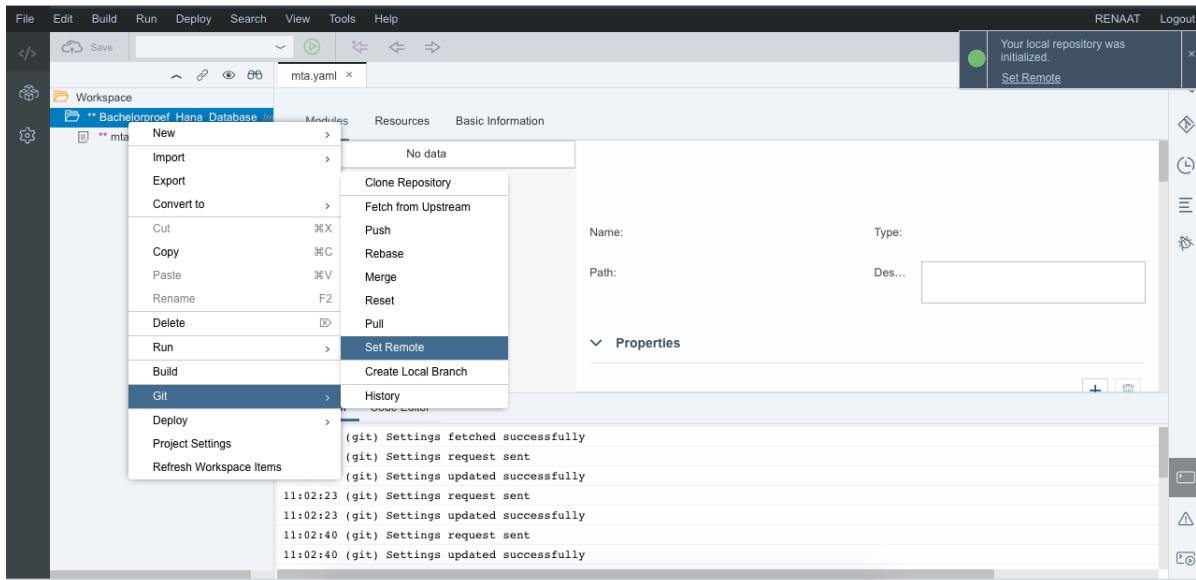
Een realistische opstelling van een CI/CD voorbeeldapplicatie start bijna nooit vanaf nul, het bouwt meestal voort op bestaande, geschreven software. Daarom zal er in deze voorbeeldapplicatie manueel een basis gelegd worden, zo kan er aan de hand van een build scheduler voortgebouwd worden op deze geschreven 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. Eerst maken we een HANA Database Module zoals in figuren 5.14, 5.15 en 5.16 te zien zijn. Daarna wordt een HDB CDS Artifact gemaakt. Dit is een Core Data Services document dat definities bevat om objecten te creëren in



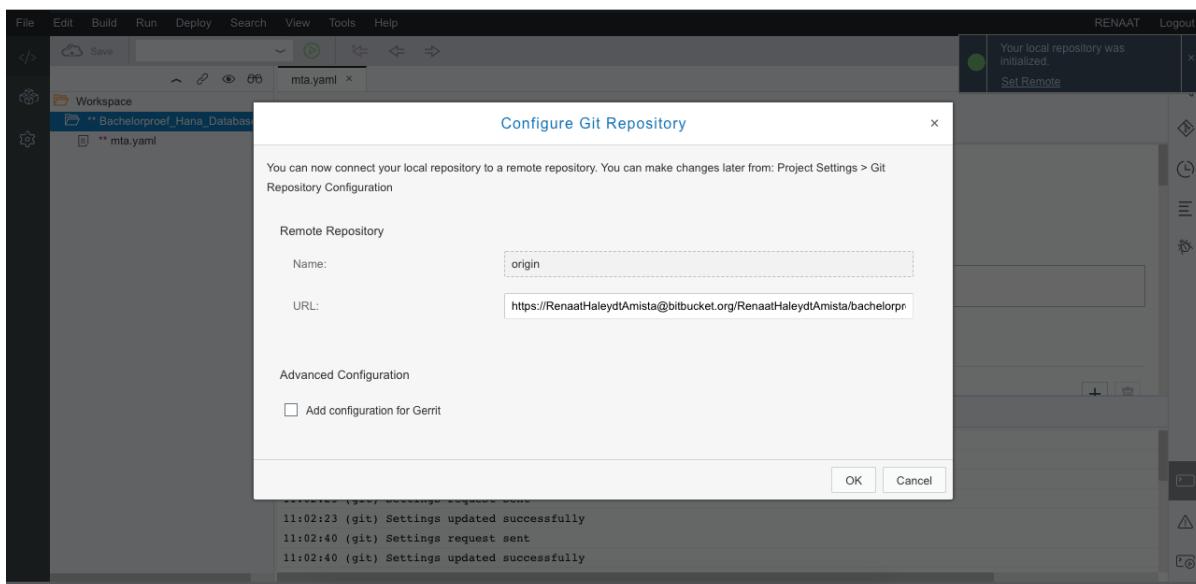
Figuur 5.9: Opzetten HANA, stap 2



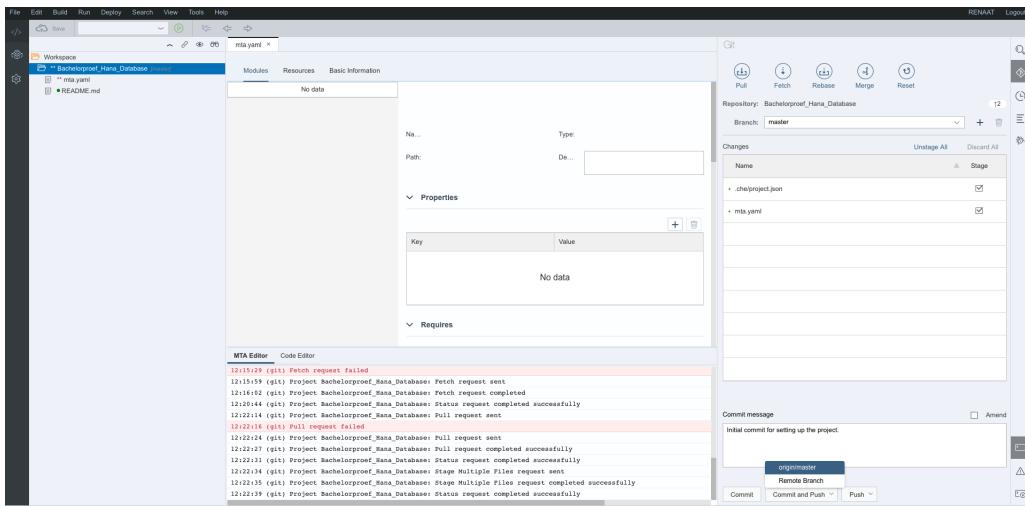
Figuur 5.10: Opzetten HANA, stap 3



Figuur 5.11: Opzetten HANA, stap 4



Figuur 5.12: Opzetten HANA, stap 5

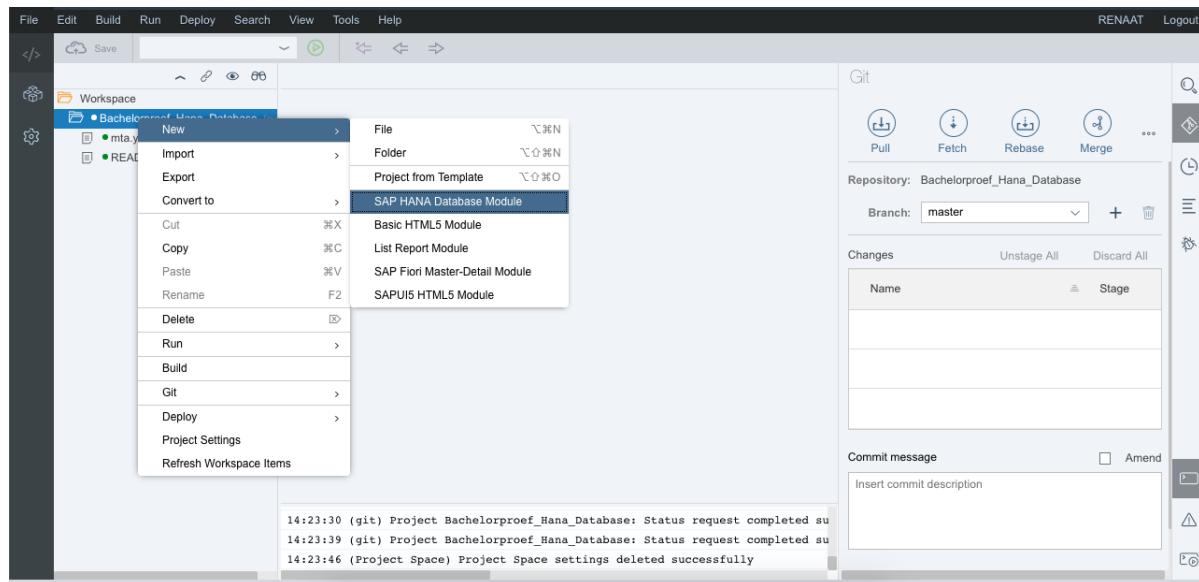


Figuur 5.13: Opzetten HANA, stap 6

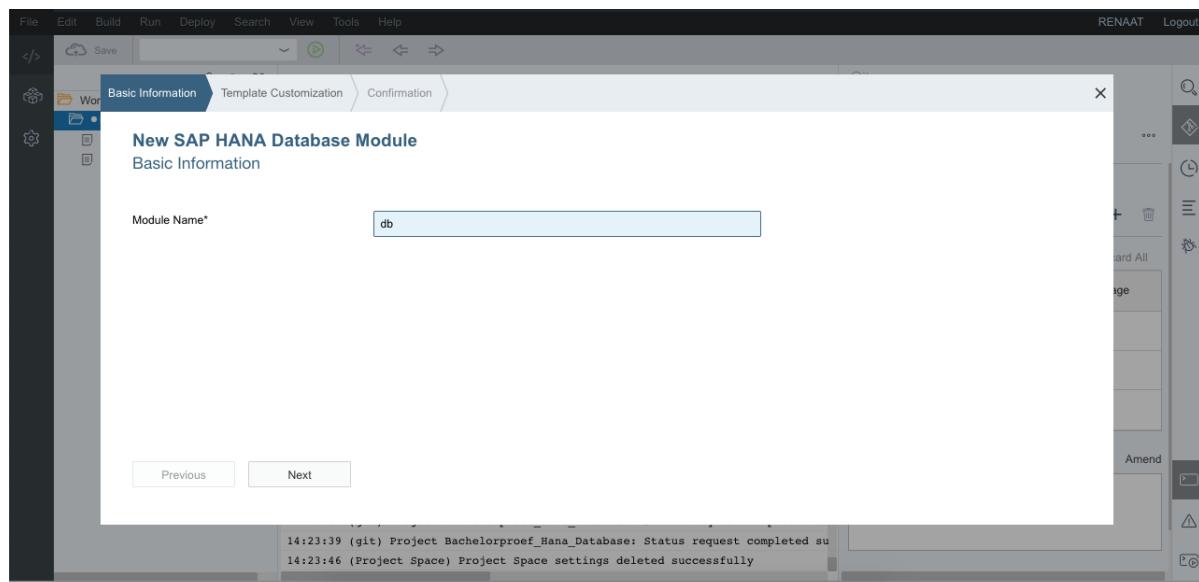
de database. Hoe de Artiest gedefinieerd wordt kan u zien in figuren 5.17, 5.18, 5.19 en 5.20. Nu alle gegevens ingevoerd zijn, moeten we de database effectief aanmaken. In de meest linkse tab is er een knop 'Database Explorer' genaamd. Als je daar op klikt zie je een lege kolom verschijnen. Je moet op het + -teken klikken om een database aan te maken. Kies uit de lijst de optie met jouw naam en geef een realistische naam bij het veld 'How to Show in Display'. Laat de rest van de setting zoals ze zijn. Dan is het de bedoeling om de gegevens die we hierboven hebben ingevoerd in de net aangemaakte database komen. Dit doe je door terug te gaan naar de 'Development' omgeving (in meest linkse tab), daar met de rechter muisknop op 'data.hdbcards' file te klikken en voor de optie 'Build Selected Files' te kiezen. De stappen kan je volgen in figuren 5.21 en 5.22. Eens deze gegevens ingevoerd zijn moeten we alle veranderingen toevoegen aan de commit om dan te pushen naar de repository.

### Node.js module

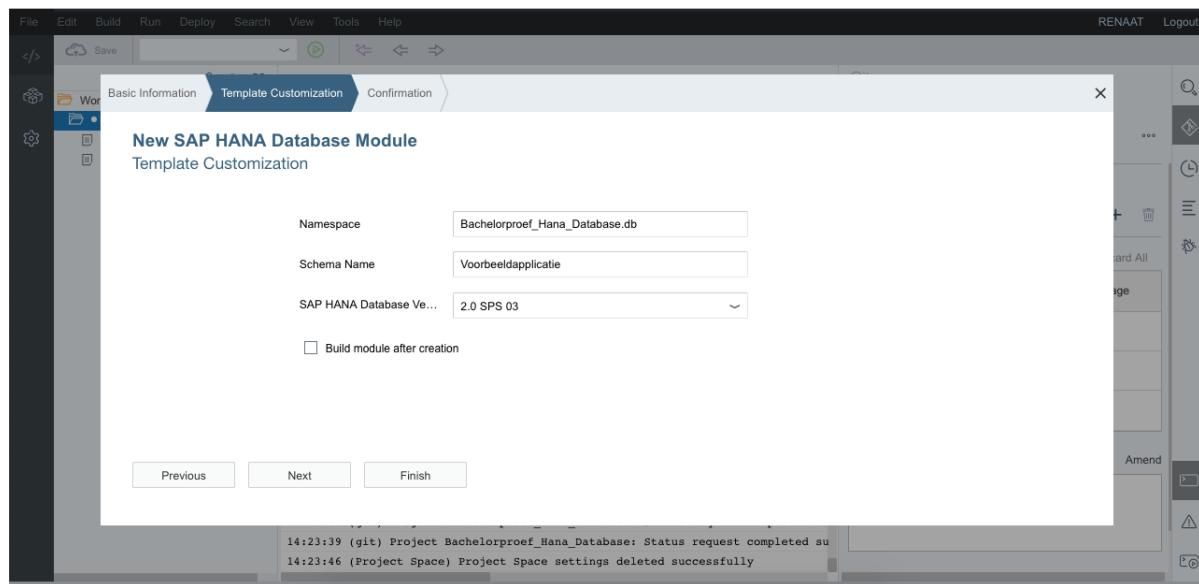
Om de models te gebruiken dat gecreëerd zijn, moeten we een tweede module toevoegen aan de HANA database: een Node.js module om de data als OData service te kunnen gebruiken. Deze module implementeert XSJS en XSODATA die op hun beurt zorgen voor de transformatie van het data model en de bereikbaarheid naar de buitenwereld. 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 figuur 5.23 ziet moeten de instellingen aangepast worden. De feature Tools For Node.js Development moet beschikbaar zijn om te gebruiken in het project. Dit moet gesaved worden en het project moet opnieuw geladen worden. De volgende stap is de module maken door op het project met de rechtermuisknop te klikken, new en dan Node.js Module kiezen. Geef het een goede naam (js is good practice) en zorg ervoor dat Enable XSJX support aangevinkt is alvorens op Finish te klikken, zoals te zien is in figuur 5.24, 5.25 en 5.26. Databeveiliging is zeer belangrijk. Binnen HANA wordt er gebruik gemaakt van UAA (User Account en Authentication) om de Node.js module te



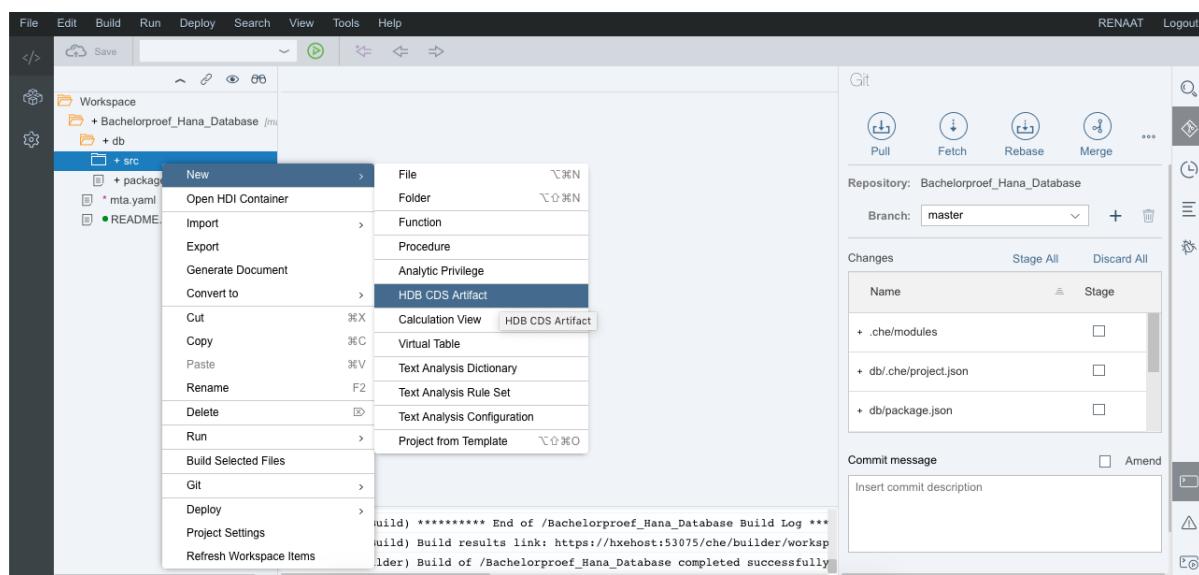
Figuur 5.14: Opzetten HANA, stap 7



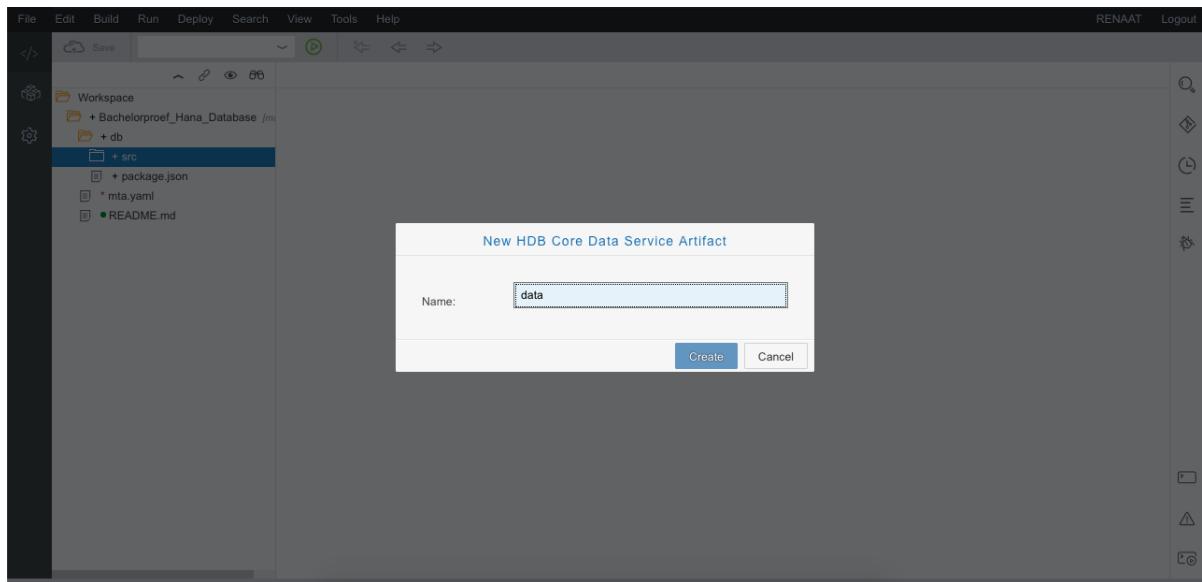
Figuur 5.15: Opzetten HANA, stap 8



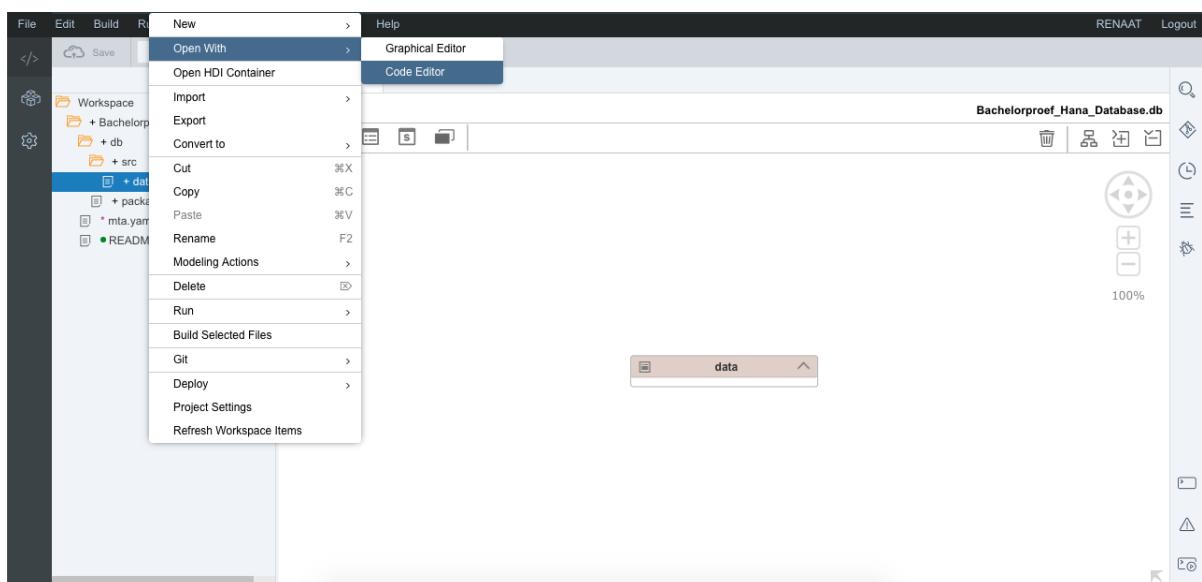
Figuur 5.16: Opzetten HANA, stap 9



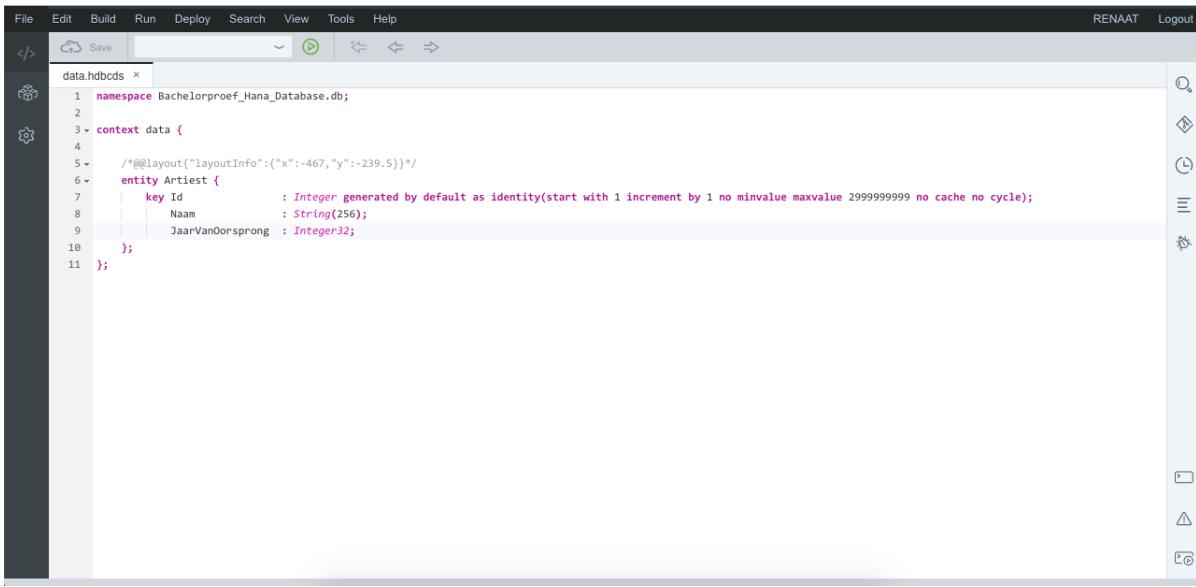
Figuur 5.17: Opzetten HANA, stap 10



Figuur 5.18: Opzetten HANA, stap 11



Figuur 5.19: Opzetten HANA, stap 12

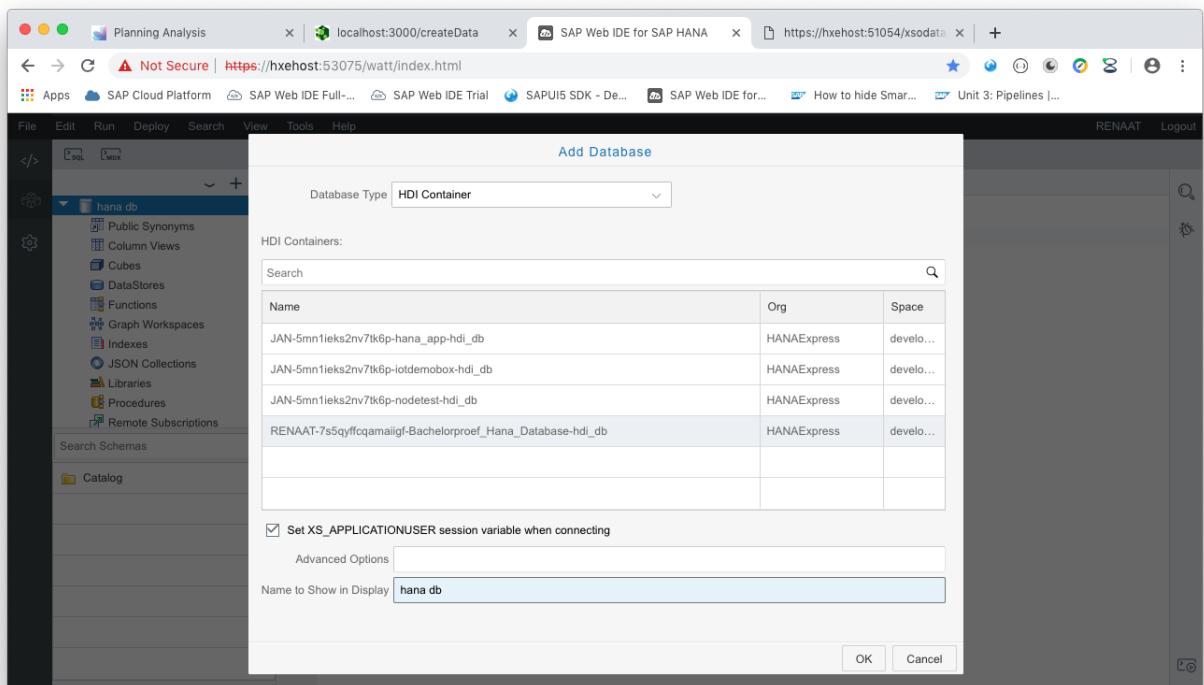


```

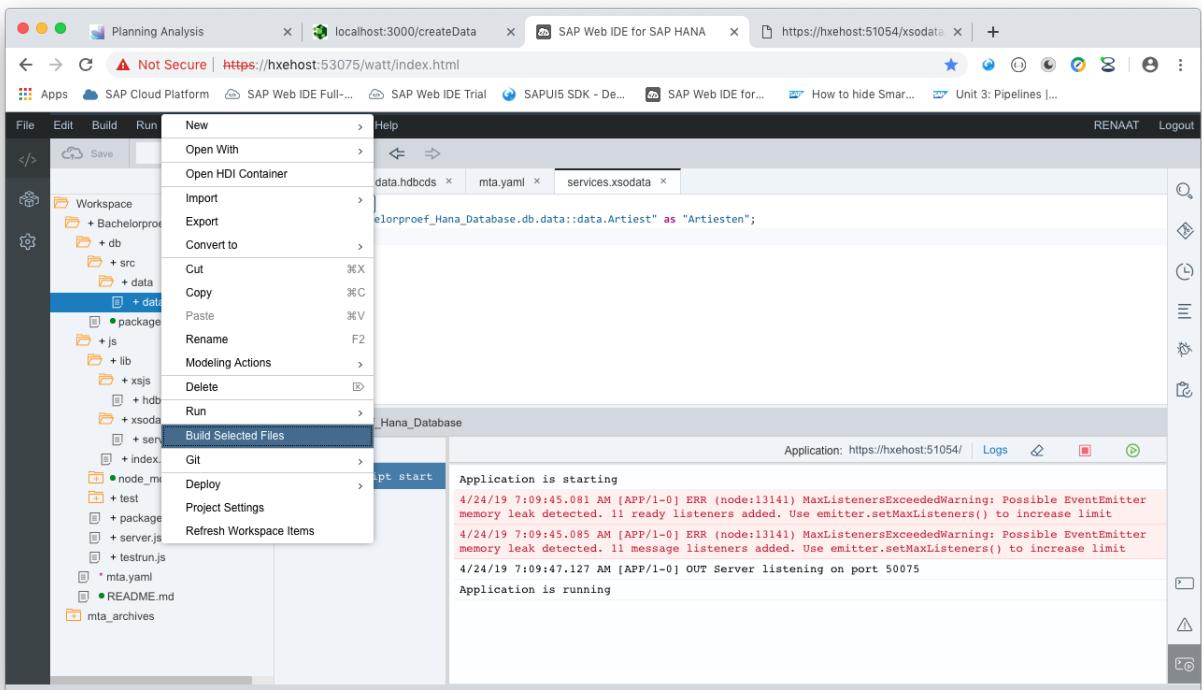
File Edit Build Run Deploy Search View Tools Help
RENAAT Logout
</> Save
data.hdcbs x
1 namespace Bachelorproef_Hana_Database.db;
2
3 context data {
4
5   @@layout{"layoutInfo":{"x":-467,"y":-239.5}}*
6   entity Artiest {
7     key Id           : Integer generated by default as identity(start with 1 increment by 1 no minvalue maxvalue 2999999999 no cache no cycle);
8     | Naam          : String(256);
9     | JaarVanOorsprong : Integer32;
10   };
11 };

```

Figuur 5.20: Opzetten HANA, stap 13



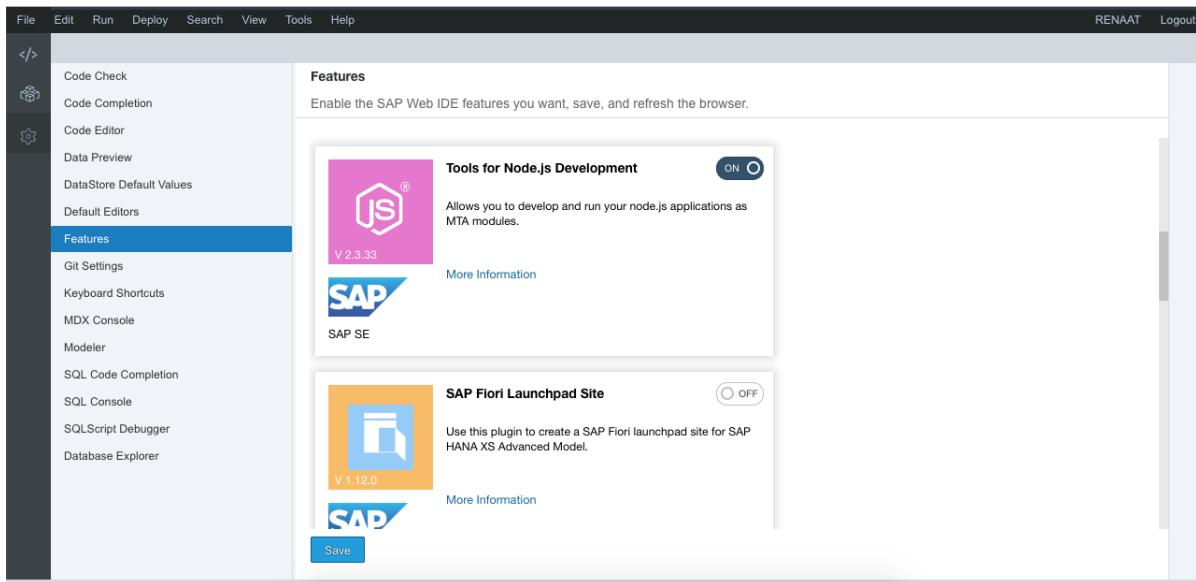
Figuur 5.21: Opzetten HANA, stap 14



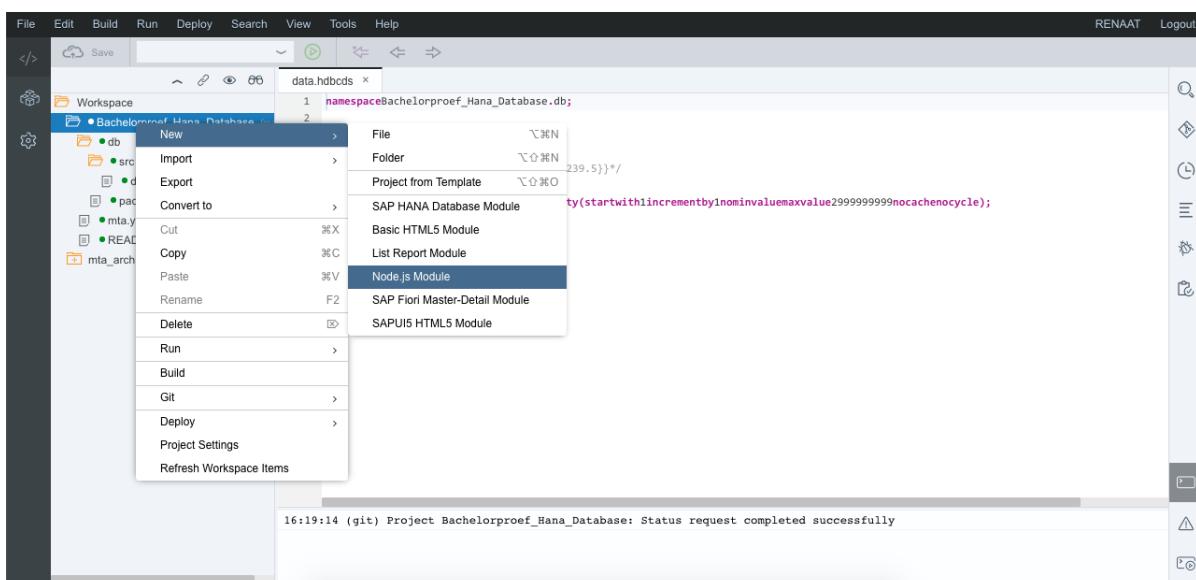
Figuur 5.22: Opzetten HANA, stap 15

beveiligen. Deze service moet samen met de database module en de HDI container, achter de db module, toegevoegd worden als resource zoals te zien valt in figuur 5.27. 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 figuren 5.28 en 5.29. 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 figuur 5.30. De inhoud van deze nieuwe file is te zien in figuur 5.31. 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 zoals te zien is in figuur 5.32.

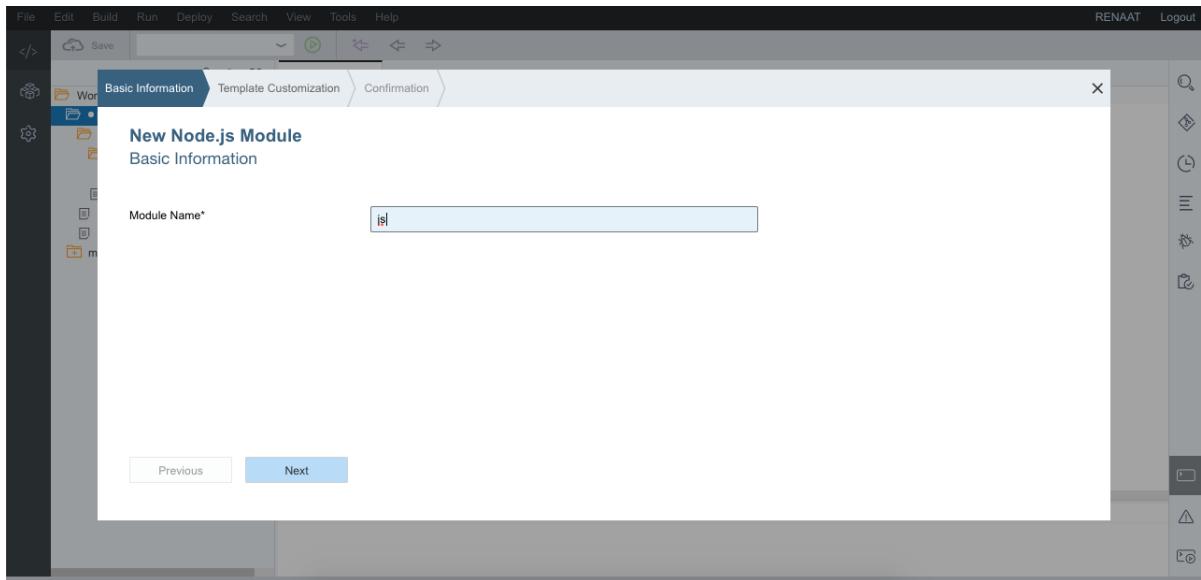
### 5.2.3 Automated testing binnen SAPUI5



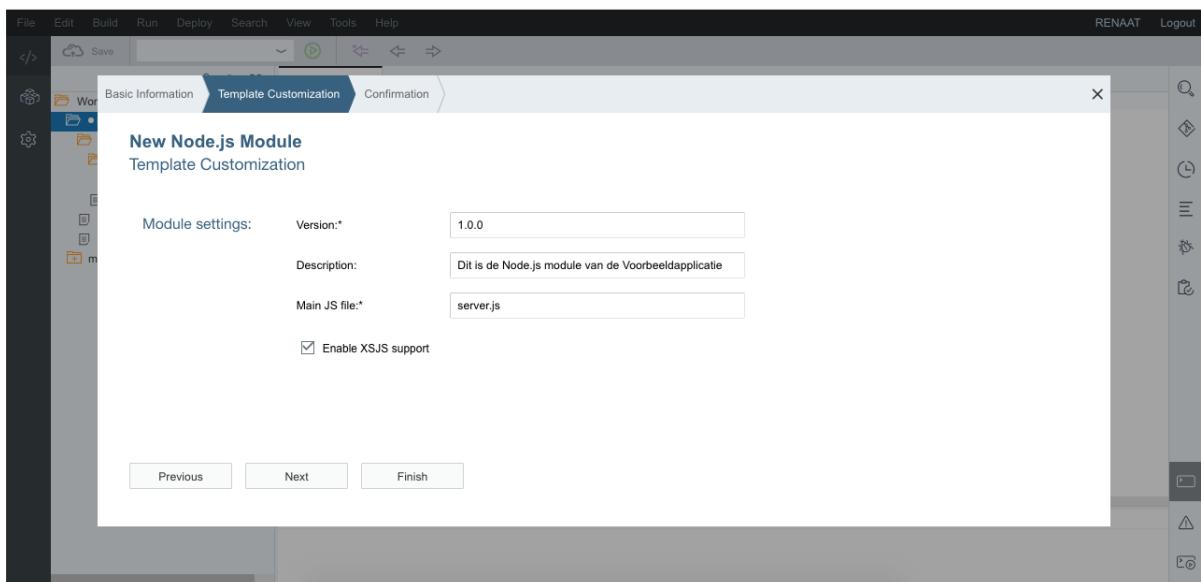
Figuur 5.23: Opzetten HANA, stap 16



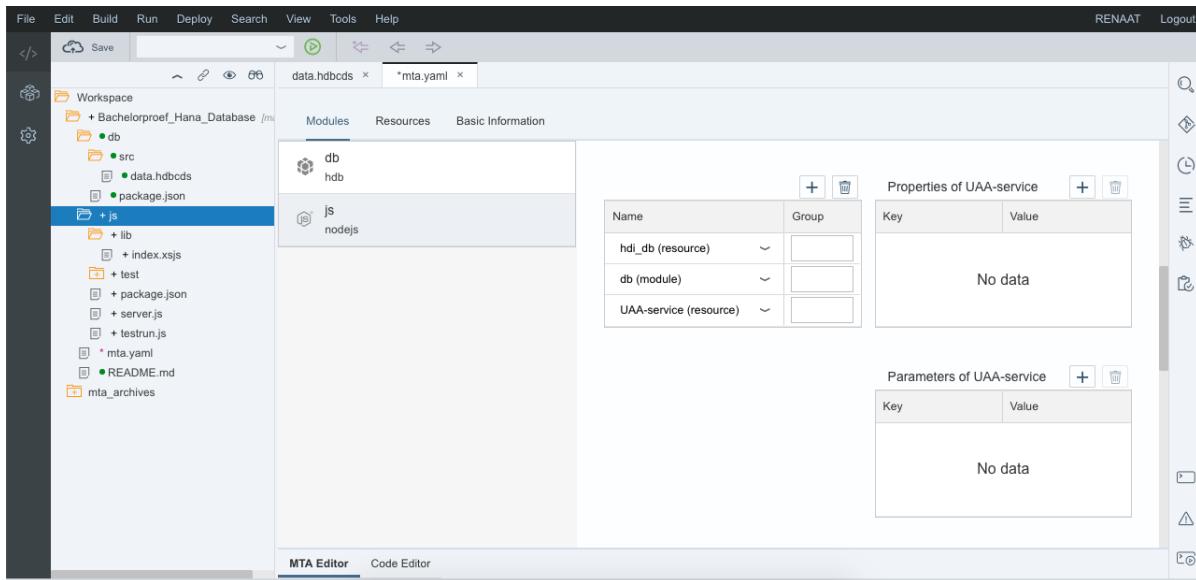
Figuur 5.24: Opzetten HANA, stap 17



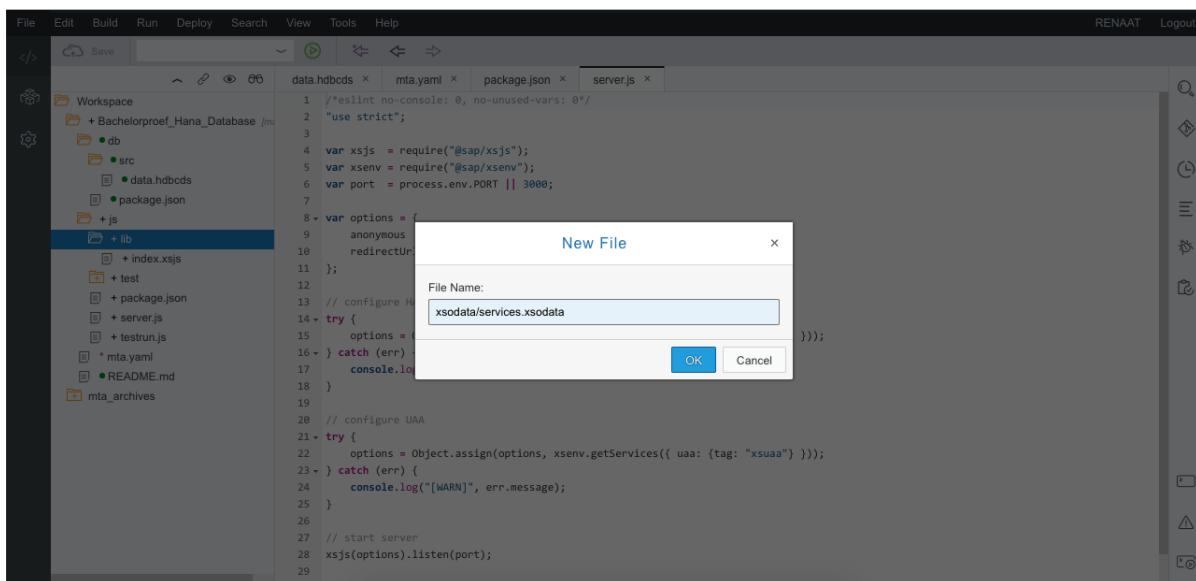
Figuur 5.25: Opzetten HANA, stap 18



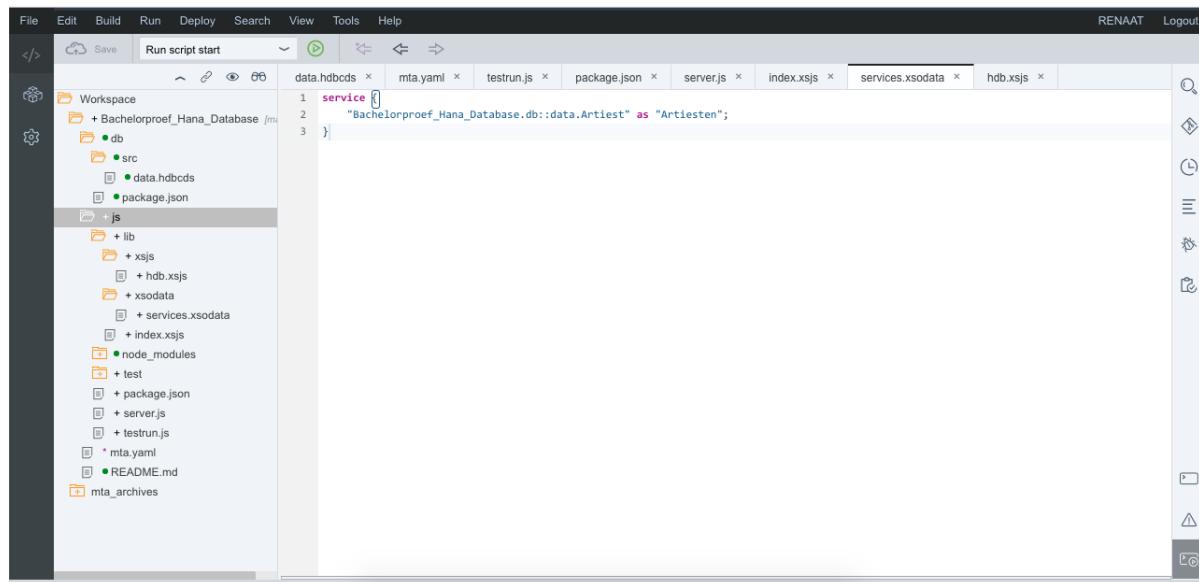
Figuur 5.26: Opzetten HANA, stap 19



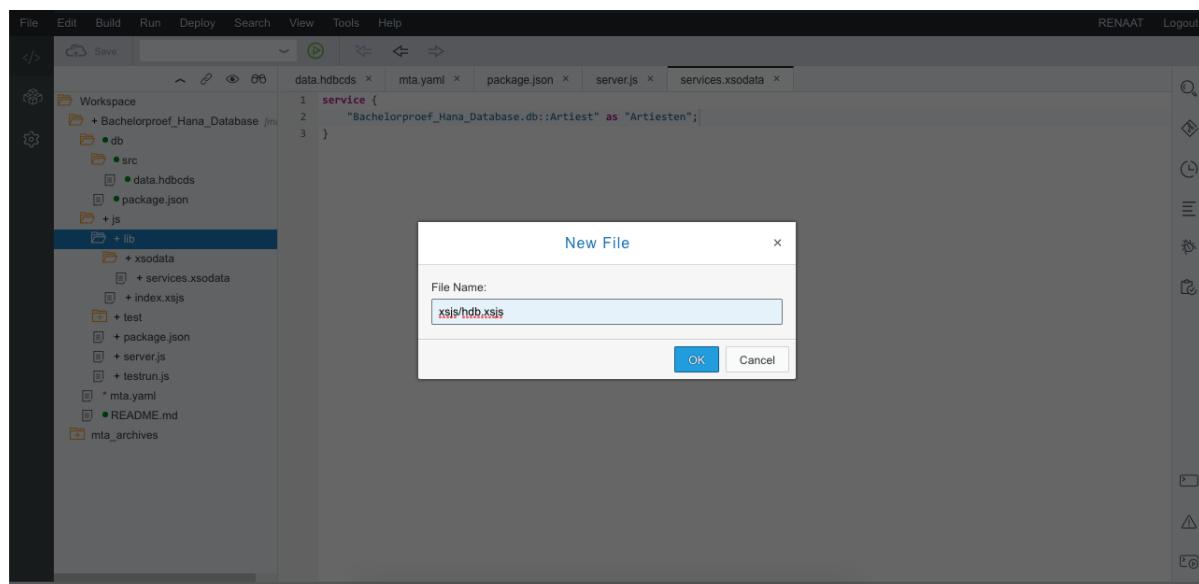
Figuur 5.27: Opzetten HANA, stap 20



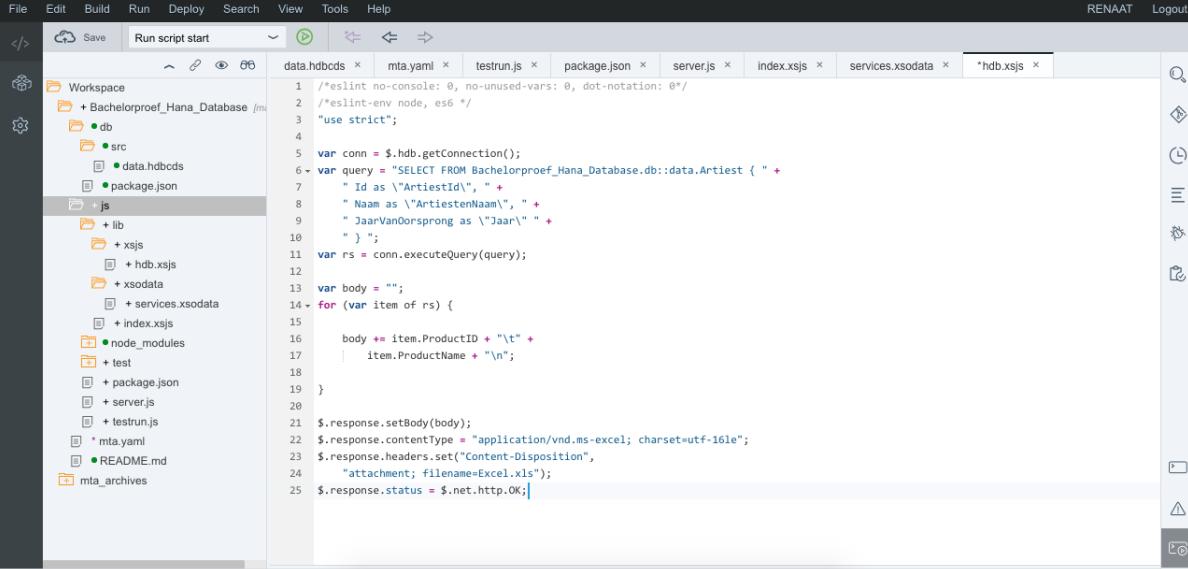
Figuur 5.28: Opzetten HANA, stap 21



Figuur 5.29: Opzetten HANA, stap 22



Figuur 5.30: Opzetten HANA, stap 23

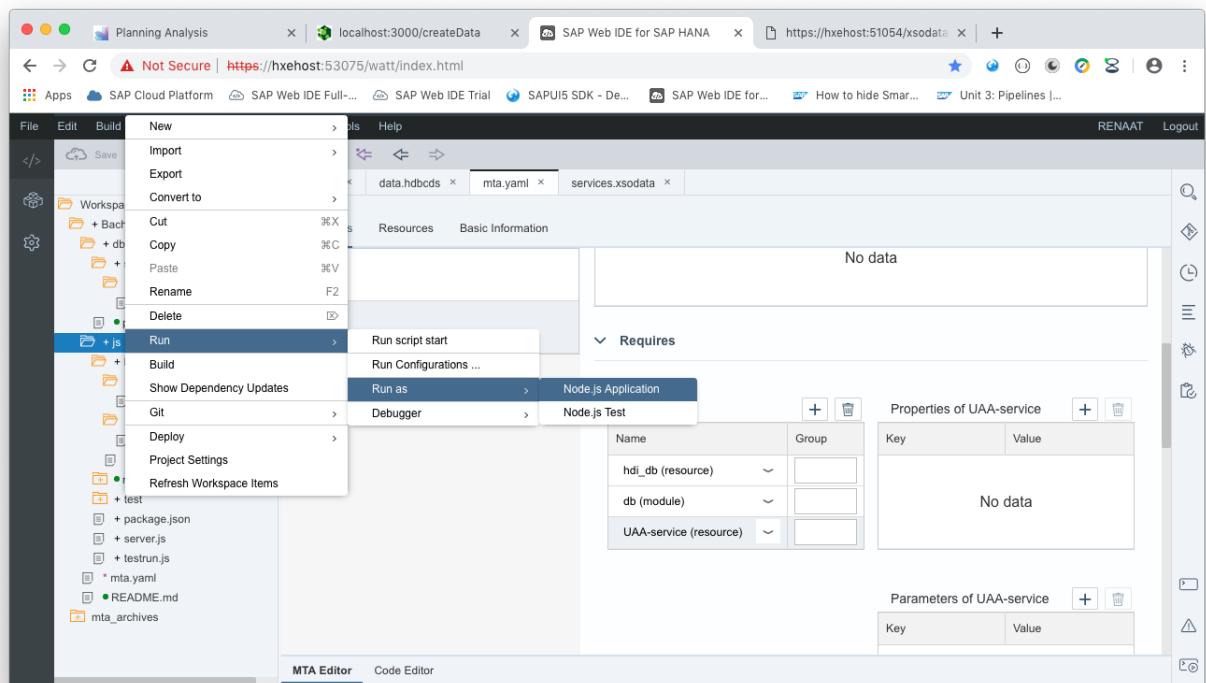


```

File Edit Build Run Deploy Search View Tools Help
Save Run script start
RENAAT Logout
Workspace
+ Bachelorproof_Hana_Database [mta]
  + db
  + src
    data.hdbcods
    package.json
  + jb
    + lib
    + xsjs
      + hdb.xsjs
    + xsodata
    + services.xsodata
      + index.xlsjs
    node_modules
    + test
      + package.json
      server.js
      testrun.js
      mta.yaml
      README.md
  mta_archives
data.hdbcods x mta.yaml x testrun.js x package.json x server.js x index.xlsjs x services.xsodata x *hdb.xsjs x
1 /*eslint no-console: 0, no-unused-vars: 0, dot-notation: 0*/
2 /*eslint-env node, es6 */
3 "use strict";
4
5 var conn = $hdb.getConnection();
6 var query = "SELECT FROM Bachelorproof_Hana_Database.db::data.Artiest { "
7   " Id as \"ArtiestId\", "
8   " Naam as \"ArtiestenNaam\", "
9   " JaarVanOorsprong as \"Jaar\" "
10  " } ";
11 var rs = conn.executeQuery(query);
12
13 var body = "";
14 for (var item of rs) {
15
16   body += item.ProductID + "\t"
17   | item.ProductName + "\n";
18 }
19
20
21 $.response.setBody(body);
22 $.response.contentType = "application/vnd.ms-excel; charset=utf-16le";
23 $.response.headers.set("Content-Disposition",
24   "attachment; filename=Excel.xls");
25 $.response.status = $.net.http.OK;

```

Figuur 5.31: Opzetten HANA, stap 24



Figuur 5.32: Opzetten HANA, stap 25

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

Uit vorig hoofdstuk is gebleken dat (SCHRAPPEN WAT NIET PAST): Jenkins & Travis & Bamboo 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.

### 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

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.

## 6.3 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).

## 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 te 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 (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 testen moeten deze verandering ook doorstaan.

### **Testing in SAPUI5**

De developers van SAP hebben ook eigen test-frameworks ontwikkeld. Voor Unit testen zijn er de QUnit tests en voor de integratie testen hebben ze OPA, One-Page Acceptance Test, ontwikkeld.

## **6.5 Short List**

Uit het hoofdstuk5 hebben we in grote lijnen de vergelijking gemaakt tussen verschillende build schedulers. Hier gaan we een gedetailleerde vergelijking te maken door de build schedulers uit te testen aan de hand van de voorbeeldapplicatie die uitgewerkt is in hoofdstuk5. We hebben Jenkins en In het vorige hoofdstuk hebben we de Ubuntu server

```
Last login: Sat Apr 13 09:24:41 2019 from 91.176.244.245
[root@ubuntu-s-1vcpu-1gb-ams3-01:~# sudo apt install openjdk-8-jdk
```

Figuur 6.1: Installatie Jenkins, stap 1

helemaal klaar gezet voor het gebruik met een build scheduler. De opzet van Jenkins en wordt hier beschreven.

## Jenkins

Omdat Jenkins op Java draait is het nodig om Java 8 te installeren. Dit is dan ook de eerste stap in het proces. De eerste stap is inloggen in de server door ssh root@188.166.61.128 in te typen. Daarna moet je het commando sudo apt install openjdk-8-jdk intypen om JAVA 8 te installeren. De server vraagt bevestiging om te downloaden en te installeren, er moet enkel maar y worden getypt en op return gedrukt worden om te bevestigen. Dit kan u zien in figuren 6.1 en 6.2. Nu kunnen we overgaan tot de installatie van Jenkins op de server. We maken gebruik van de packages die Jenkins onderhoud om de software te installeren op Ubuntu zoals te zien is in figuur 6.3. Eens dit gebeurd is moeten we de Debian package repository toevoegen aan de server's sources.list. De apt - een command-line tool binnen Ubuntu om software te installeren - moet eerst geupdate worden, nadien kan Jenkins effectief geïnstalleerd worden. Deze stappen kan u volgen in figuren 6.4 en 6.5.

Na de installatie kunnen we Jenkins starten op de server door het commando 'sudo systemctl start jenkins' in te typen. Omdat dit commando geen uitvoer geeft moeten we het commando 'sudo systemctl status jenkins' intypen. Als je active ziet verschijnen wil dit zeggen dan Jenkins runt op de Ubuntu server. Deze stappen kan je volgen in figuur 6.6. Eens Jenkins gestart is moeten we die kunnen aanspreken vanuit een browser, daarom moeten we de regels van de firewall wat aanpassen. Standaard draait Jenkins op poort 8080. Deze moet geopend worden op de server door het commando 'sudo ufw allow 8080' in te voeren zoals in figuur 6.7 te zien is.

Nu moet Jenkins geconfigureerd worden, dit moet vanuit elke browser gebeuren. Er moet naar <http://188.166.61.128:8080> gesurft worden om de configuratie te voltooien, zoals aangegeven in figuur 6.8. De cijfers voor de :8080 staan voor het ip-adres van de Ubuntu server, de :8080 is de poort waar je naartoe wil gaan. Hier draait - zoals in de vorige stappen geconfigureerd - Jenkins op. Het paswoord dat gevraagd wordt is terug te vinden in de 'initialAdminPassword' file dat zich in de map '/var/lib/jenkins/secrets/' bevindt op de server. Het is voldoende om op de server volgend commando in te voeren: 'sudo cat /var/lib/jenkins/secrets/initialAdminPassword'. Het paswoord dat als uitvoer komt moet gekopieerd worden en geplakt worden in het invoerveld in de web browser zoals in figuur 6.9 te zien is. De volgende stap is het installeren van de voorgestelde plugins. Er wordt gevraagd om een eerste user te configureren, maar voor deze doeleinden is het voldoende om als admin en met het initiële paswoord verder te gaan, er mag dus op 'Continue as admin' gedrukt worden. Nu moet er instantie gemaakt worden door te controleren of de url die ingevuld is klopt met de url die enkele stappen geleden ingegeven is om de webpagina te openen. Wanneer deze klopt kan er op 'Save and Finish' en nadien op 'Start

```

Reading package lists... Done
Building dependency tree
Reading state information... Done
The following package was automatically installed and is no longer required:
  *grub-pc-bin
Use 'sudo apt autoremove' to remove it.
The following additional packages will be installed: één bron van informatie te voorzien voor test data, vooral in
  adwaita-icon-theme at-spi2-core ca-certificates-java dconf-gsettings-backend dconf-service fontconfig fontconfig-service en
  fonts-dejavu-core fonts-dejavu-extra glib-networking glib-networking-common glib-networking-services gsettings-desktop-schemas
  gtk-update-icon-cache hicolor-icon-theme humanity-icon-theme java-common libasound2 libasound2-data libasyncns0
  libatk-bridge2.0-0 libatk-wrapper-java libatk-wrapper-java-jni libatk1.0-0 libatksp1.0-0 libavahi-client3
  libavahi-common-data libavahi-common3 libcairo-gobject2 libcairo2 libcolord2 libcroco3 libcurl2 libdatriel libdcnf1
  libdrm-amdgpu libdrm-intel1 libdrm-nouveau2 libdrm-radeon1 libeoxy0 libflac8 libfontconfig1 libfontenc1 libgd-pixbuf2.0-0
  libgd-pixbuf2.0-0-bin libgd-pixbuf2.0-common libgf7 libgl1 libgl1-mesa-dri libgl1-mesa-glx libglapi-mesa libglvnd0 libglx-mesa0
  libglx0 libgraphite2-3 libgtk-3-0 libgtk-3-bin libgtk-3-common libharfbuzz0b libice-dev libice6 libjbig0 libjpeg-turbo8 libjpeg8
  libjson-glib-1.0-0 libjson-glib-1.0-common liblcms2-2 liblomm7 libnspr4 libness3 libogg0 libpango-1.0-0 libpangocairo-1.0-0
  libpangoft2-1.0-0 libpccaccess8 libpccsclite1 libpixman-1-0 libproxy1v5 libpthread-stubs0-dev libpulse0 librest-0.7-0 librsvg2-2
  librsvg2-common libsensors4 libsm-dev libsm6 libsnf1 libsoup2.4-1 libthai-data libthai0 libtiff5
  libvorbis0a libvorbisenc2 libwayland-client0 libwayland-cursor0 libwayland-egl1 libxi1-dev libxi1-doc libxi1-xcb1 libxau-dev
  libxaw7 libxcb-dri2-0 libxcb-dri3-0 libxcb-glx0 libxcb-present0 libxcb-render0 libxcb-shm0 libxcb-sync1 libxcb1-dev
  libxcompositel libxcursor1 libxdamage1 libxdmcp-dev libfixes3 libxft2 libxi6 libxinerama1 libxkbcommon0 libxmu6 libxpm4
  libxrandr2 libxrender1 libxshmfence1 libxt-dev libxt6 libxtst6 libxv1 libxxf86dg1 libxxf86vm1 openjdk-8-jdk-headless
  openjdk-8-jre openjdk-8-jre-headless ubuntu-mono x11-common x11-utils x11proto-core-dev x11proto-dev xorg-sgml-doctools
  xtrans-dev
Suggested packages:
  default-jre libasound2-plugins alsamixer colord cups-common gvfs libice-doc liblcms2-utils pcscd pulseaudio librsvg2-bin
  lm-sensors libsm-doc libxcb-doc libxt-doc openjdk-8-demo openjdk-8-source visualvm icedtea-8-plugin libnss-mdns
  fonts-ipafont-gothic fonts-ipafont-mincho fonts-wqy-microhei fonts-wqy-zenhei fonts-indic mesd-utils
The following NEW packages will be installed:
  adwaita-icon-theme at-spi2-core ca-certificates-java dconf-gsettings-backend dconf-service fontconfig fontconfig-config
  fonts-dejavu-core fonts-dejavu-extra glib-networking glib-networking-common glib-networking-services gsettings-desktop-schemas
  gtk-update-icon-cache hicolor-icon-theme humanity-icon-theme java-common libasound2 libasound2-data libasyncns0
  libatk-bride2.0-0 libatk-wrapper-java libatk-wrapper-java-jni libatk1.0-0 libatksp1.0-0 libavahi-client3
  libavahi-common-data libavahi-common3 libcairo-gobject2 libcairo2 libcolord2 libcroco3 libcurl2 libdatriel libdcnf1
  libdrm-amdgpu libdrm-intel1 libdrm-nouveau2 libdrm-radeon1 libeoxy0 libflac8 libfontconfig1 libfontenc1 libgd-pixbuf2.0-0
  libgd-pixbuf2.0-0-bin libgd-pixbuf2.0-common libgf7 libgl1 libgl1-mesa-dri libgl1-mesa-glx libglapi-mesa libglvnd0 libglx-mesa0
  libglx0 libgraphite2-3 libgtk-3-0 libgtk-3-bin libgtk-3-common libharfbuzz0b libice-dev libice6 libjbig0 libjpeg-turbo8 libjpeg8
  libjson-glib-1.0-0 libjson-glib-1.0-common liblcms2-2 liblomm7 libnspr4 libness3 libogg0 libpango-1.0-0 libpangocairo-1.0-0
  libpangoft2-1.0-0 libpccaccess8 libpccsclite1 libpixman-1-0 libproxy1v5 libpthread-stubs0-dev libpulse0 librest-0.7-0 librsvg2-2
  librsvg2-common libsensors4 libsm-dev libsm6 libsnf1 libsoup2.4-1 libthai-data libthai0 libtiff5
  libvorbis0a libvorbisenc2 libwayland-client0 libwayland-cursor0 libwayland-egl1 libxi1-dev libxi1-doc libxi1-xcb1 libxau-dev
  libxaw7 libxcb-dri2-0 libxcb-dri3-0 libxcb-glx0 libxcb-present0 libxcb-render0 libxcb-shm0 libxcb-sync1 libxcb1-dev
  libxcompositel libxcursor1 libxdamage1 libxdmcp-dev libfixes3 libxft2 libxi6 libxinerama1 libxkbcommon0 libxmu6 libxpm4
  libxrandr2 libxrender1 libxshmfence1 libxt-dev libxt6 libxtst6 libxv1 libxxf86dg1 libxxf86vm1 openjdk-8-jdk-headless
  openjdk-8-jre openjdk-8-jre-headless ubuntu-mono x11-common x11-utils x11proto-core-dev x11proto-dev xorg-sgml-doctools
  xtrans-dev
0 upgraded, 143 newly installed, 0 to remove and 38 not upgraded.
Need to get 82.0 MB of archives.
After this operation, 456 MB of additional disk space will be used.
Do you want to continue? [Y/n] y

```

Figuur 6.2: Installatie Jenkins, stap 2

```

root@ubuntu-s-1vcpu-1gb-ams3-01:~# wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.key | sudo apt-key add -
OK

```

Figuur 6.3: Installatie Jenkins, stap 3

```

root@ubuntu-s-1vcpu-1gb-ams3-01:~# sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'
root@ubuntu-s-1vcpu-1gb-ams3-01:~# sudo apt update
Hit:1 http://mirrors.digitalocean.com/ubuntu bionic InRelease
Get:2 http://security.ubuntu.com/ubuntu bionic-security InRelease [88.7 kB]
Get:3 http://mirrors.digitalocean.com/ubuntu bionic-updates InRelease [88.7 kB]
Get:4 http://mirrors.digitalocean.com/ubuntu bionic-backports InRelease [74.6 kB]
Ign:5 http://pkg.jenkins.io/debian-stable binary/ InRelease
Get:6 http://pkg.jenkins.io/debian-stable binary/ Release [2042 B]
Get:7 http://security.ubuntu.com/ubuntu bionic-security/main amd64 Packages [320 kB]
Get:8 http://pkg.jenkins.io/debian-stable binary/ Release.gpg [181 B]
Get:9 http://pkg.jenkins.io/debian-stable binary/ Packages [14.7 kB]
Fetched 589 kB in 1s (707 kB/s)
Reading package lists... Done
Building dependency tree
Reading state information... Done
36 packages can be upgraded. Run 'apt list --upgradable' to see them.
root@ubuntu-s-1vcpu-1gb-ams3-01:~# sudo apt install Jenkins
Reading package lists... Done
Building dependency tree
Reading state information... Done
E: Unable to locate package Jenkins

```

Figuur 6.4: Installatie Jenkins, stap 4

```
[root@ubuntu-s-1vcpu-1gb-ams3-01:~# sudo apt install jenkins
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following package was automatically installed and is no longer required:
  grub-pc-bin
Use 'sudo apt autoremove' to remove it.
The following additional packages will be installed:
  daemon
The following NEW packages will be installed:
  daemon jenkins
0 upgraded, 2 newly installed, 0 to remove and 38 not upgraded.
Need to get 76.8 MB of archives.
After this operation, 77.7 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://mirrors.digitalocean.com/ubuntu bionic/universe amd64 daemon amd64 0.6.4-1build1 [99.5 kB]
Get:2 http://pkg.jenkins.io/debian-stable binary/ jenkins 2.164.2 [76.7 MB]
Fetched 76.8 MB in 4s (18.8 MB/s)
Selecting previously unselected package daemon.
(Reading database ... 106889 files and directories currently installed.)
Preparing to unpack .../daemon_0.6.4-1build1_amd64.deb ...
Unpacking daemon (0.6.4-1build1) ...
Selecting previously unselected package jenkins.
Preparing to unpack .../jenkins_2.164.2_all.deb ...
Unpacking jenkins (2.164.2) ...
Processing triggers for ureadahead (0.100.0-20) ...
Processing triggers for systemd (237-3ubuntu10.19) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
Setting up daemon (0.6.4-1build1) ...
Setting up jenkins (2.164.2) ...
Processing triggers for systemd (237-3ubuntu10.19) ...
Processing triggers for ureadahead (0.100.0-20) ...
```

Figuur 6.5: Installatie Jenkins, stap 5

using Jenkins' geklikt worden. Het dashboard van Jenkins wordt geopend. Bovenstaande stappen kan u allemaal volgen in de figuren 6.10, 6.11, 6.12, 6.13, 6.14 en 6.15.

## 6.6 Conclusie

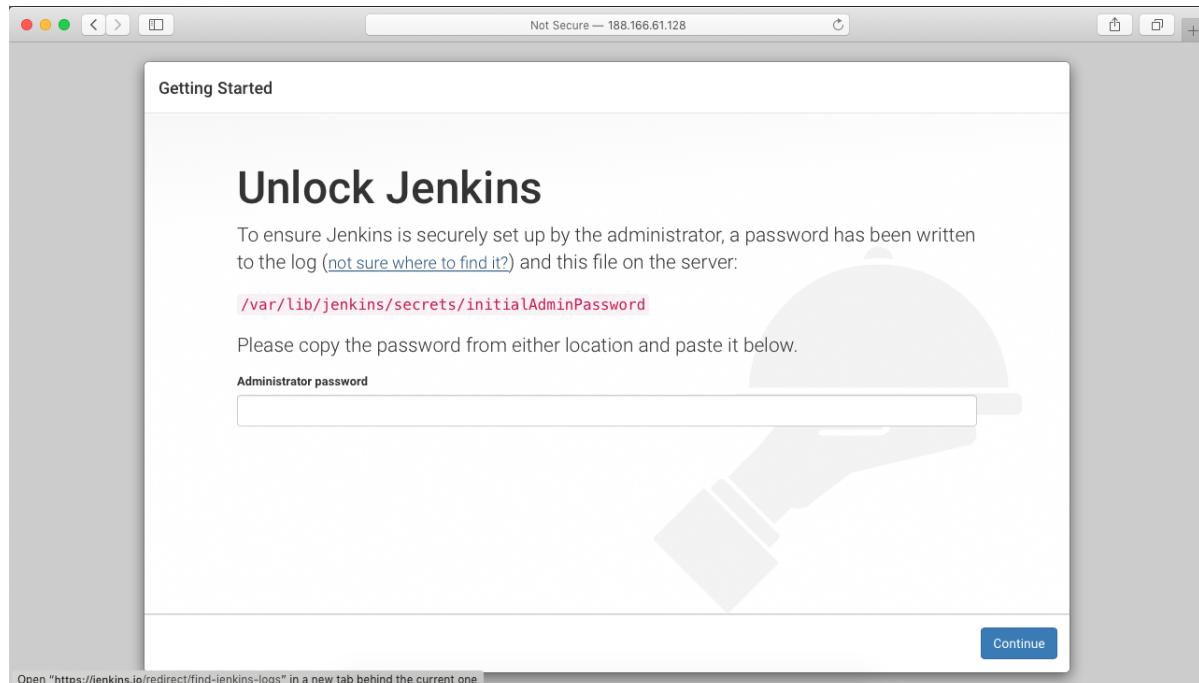
```
root@ubuntu-s-1vcpu-1gb-ams3-01:~# sudo systemctl start jenkins
root@ubuntu-s-1vcpu-1gb-ams3-01:~# sudo systemctl status jenkins
● jenkins.service - LSB: Start Jenkins at boot time
  Loaded: loaded (/etc/init.d/jenkins; generated)
  Active: active (exited) since Mon 2019-04-22 16:39:09 UTC; 14min ago
    Docs: man:systemd-sysv-generator(8)
   Tasks: 0 (limit: 1152)
  CGroup: /system.slice/jenkins.service

Apr 22 16:39:08 ubuntu-s-1vcpu-1gb-ams3-01 systemd[1]: Starting LSB: Start Jenkins at boot time...
Apr 22 16:39:08 ubuntu-s-1vcpu-1gb-ams3-01 jenkins[15737]: Correct java version found
Apr 22 16:39:08 ubuntu-s-1vcpu-1gb-ams3-01 jenkins[15737]: * Starting Jenkins Automation Server jenkins
Apr 22 16:39:08 ubuntu-s-1vcpu-1gb-ams3-01 su[15782]: Successful su for jenkins by root
Apr 22 16:39:08 ubuntu-s-1vcpu-1gb-ams3-01 su[15782]: + ??? root:jenkins
Apr 22 16:39:08 ubuntu-s-1vcpu-1gb-ams3-01 su[15782]: pam_unix(su:session): session opened for user jenkins by (uid=0)
Apr 22 16:39:08 ubuntu-s-1vcpu-1gb-ams3-01 su[15782]: pam_unix(su:session): session closed for user jenkins
Apr 22 16:39:09 ubuntu-s-1vcpu-1gb-ams3-01 jenkins[15737]: ...done.
Apr 22 16:39:09 ubuntu-s-1vcpu-1gb-ams3-01 systemd[1]: Started LSB: Start Jenkins at boot time.
```

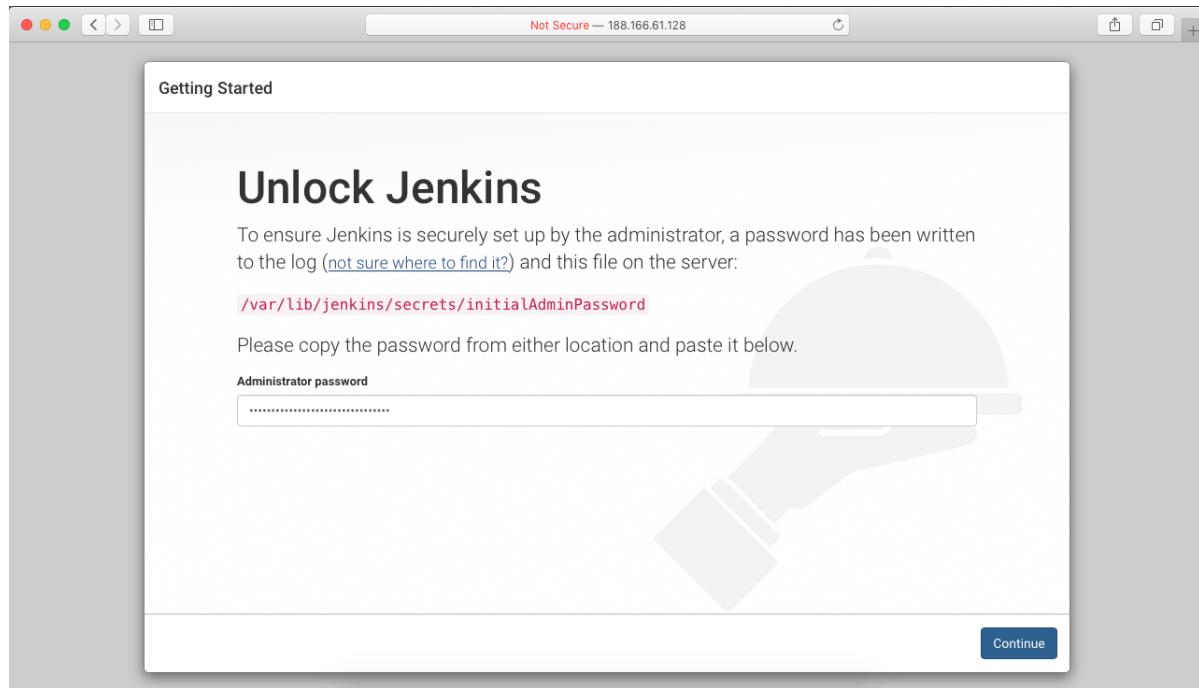
Figuur 6.6: Opzetten Jenkins, stap 1

```
[root@ubuntu-s-1vcpu-1gb-ams3-01:~# sudo ufw allow 8080
Rule added
Rule added (v6)
```

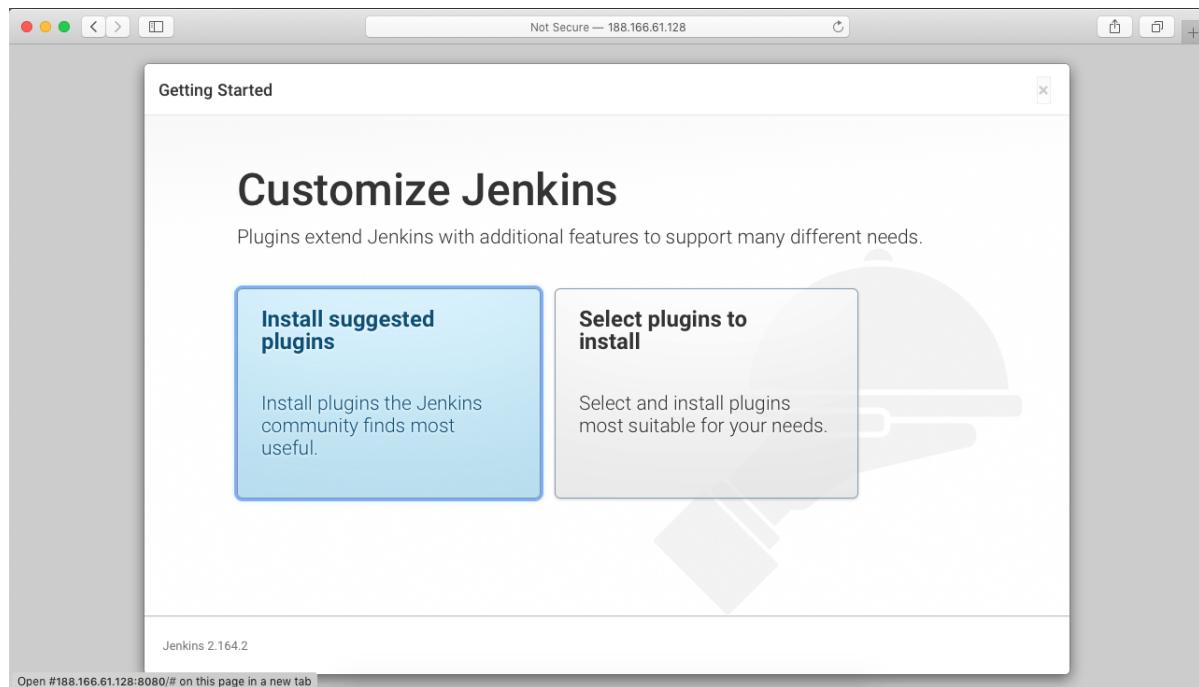
Figuur 6.7: Opzetten Jenkins, stap 2



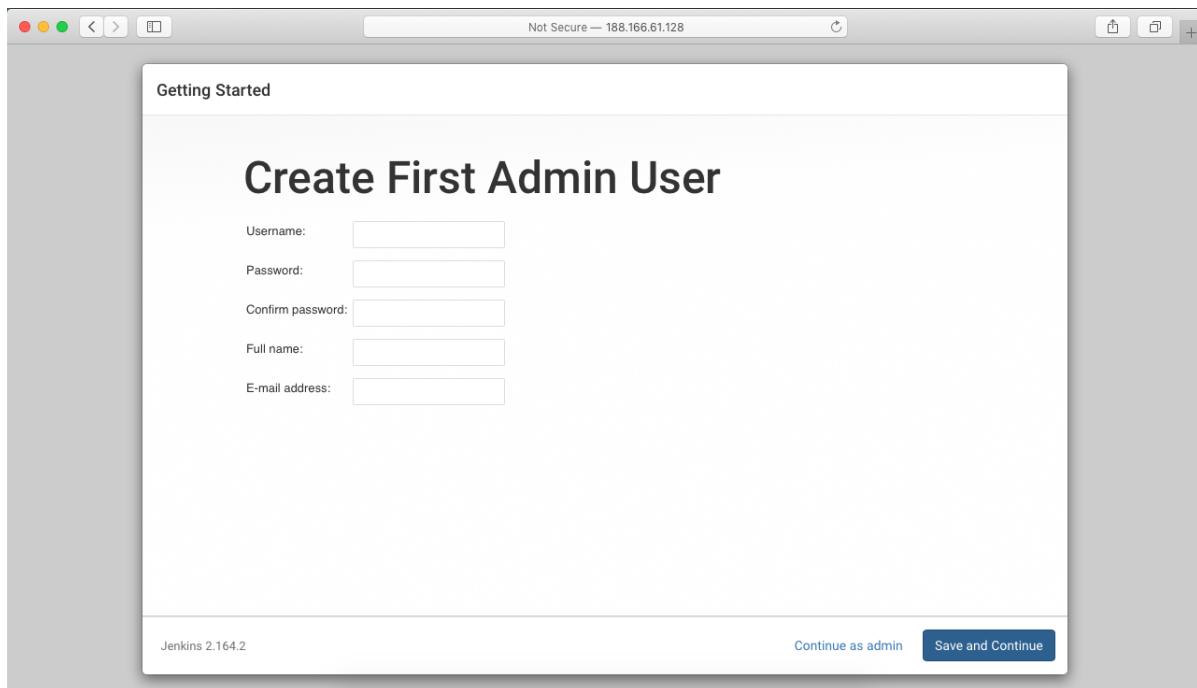
Figuur 6.8: Opzetten Jenkins, stap 3



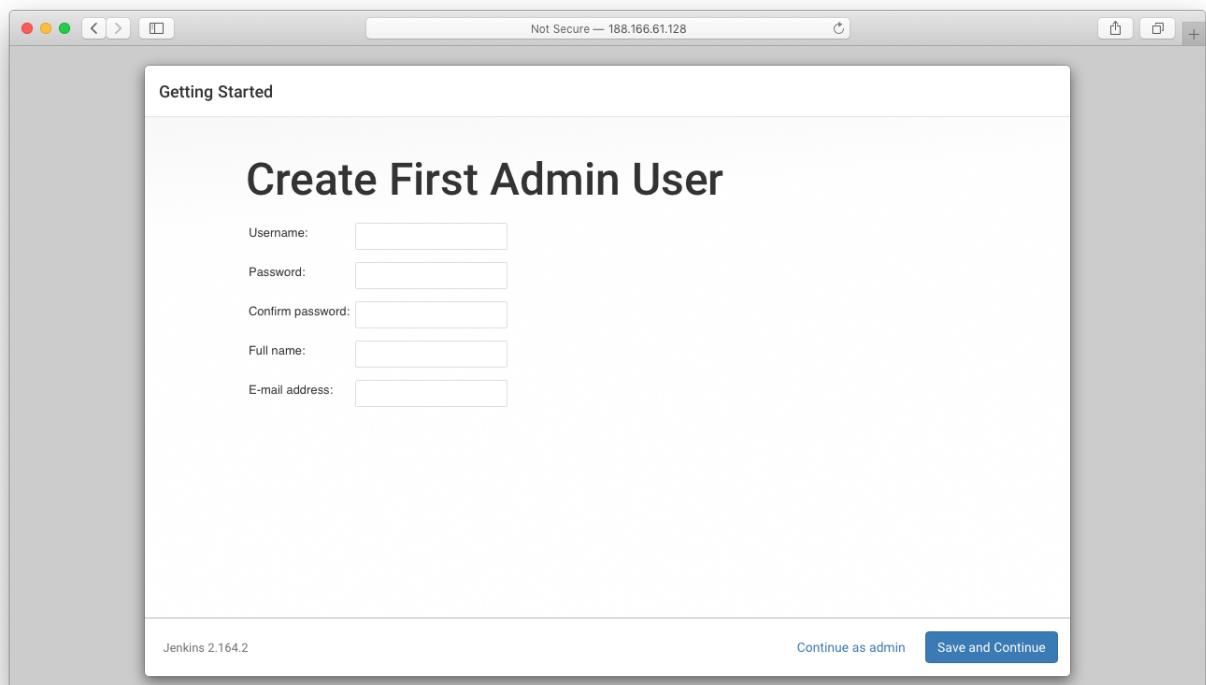
Figuur 6.9: Opzetten Jenkins, stap 4



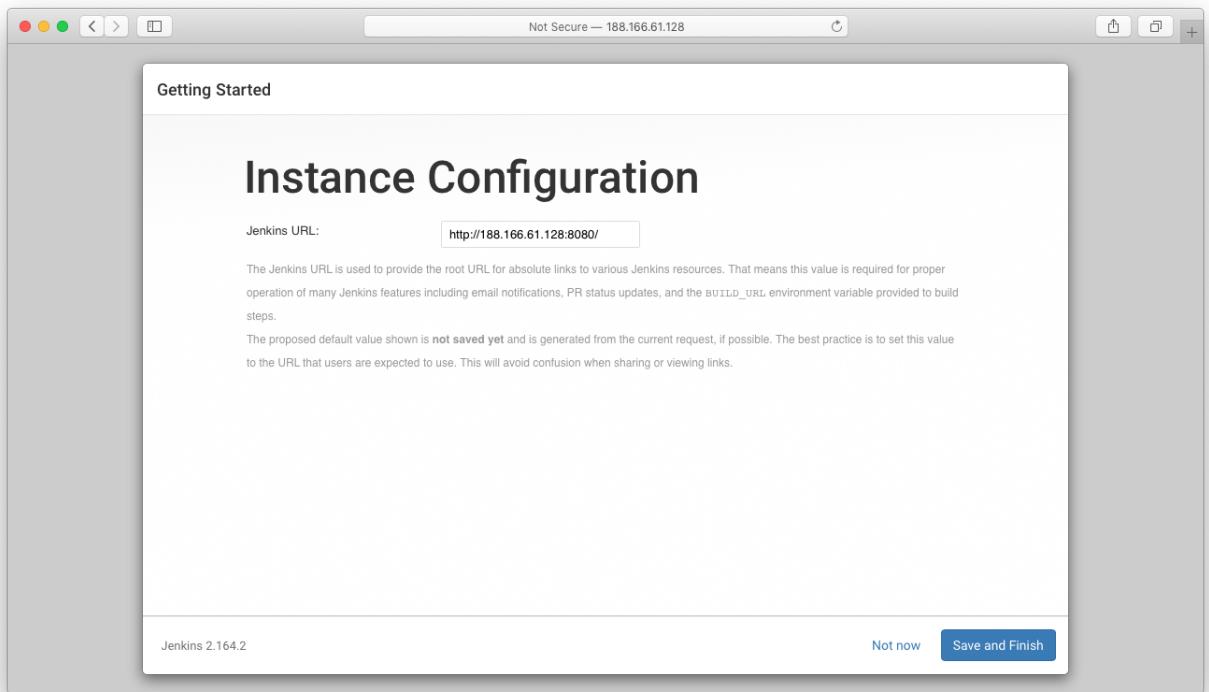
Figuur 6.10: Opzetten Jenkins, stap 5



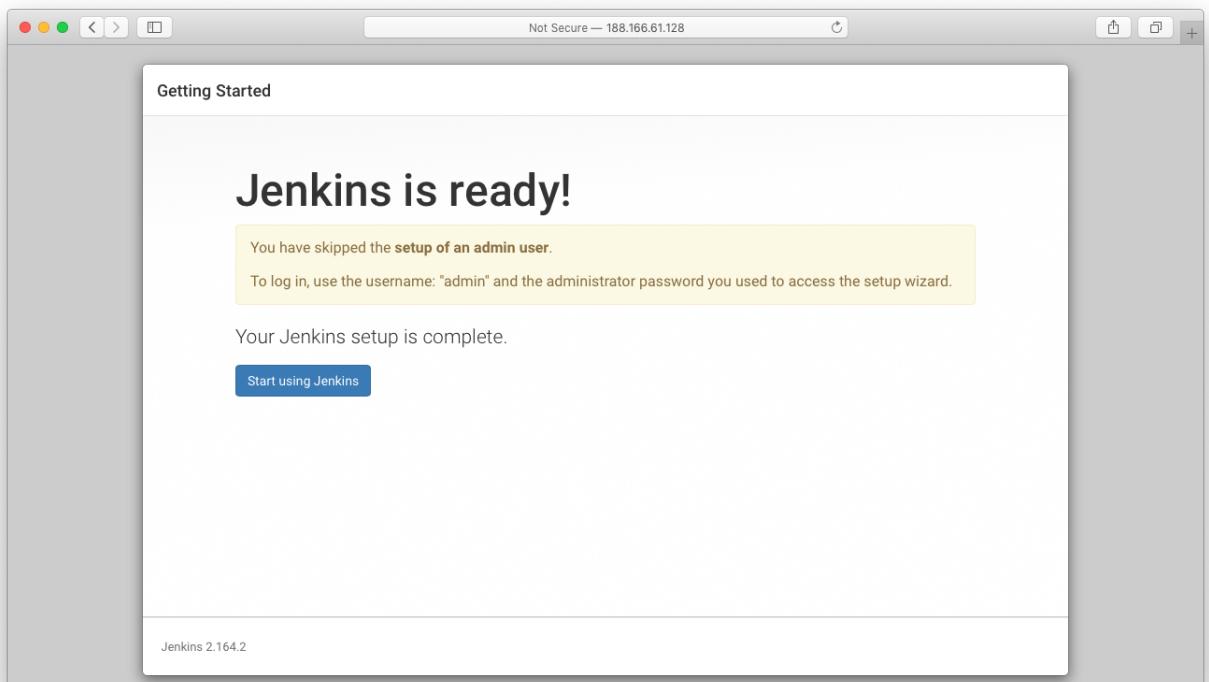
Figuur 6.11: Opzetten Jenkins, stap 6



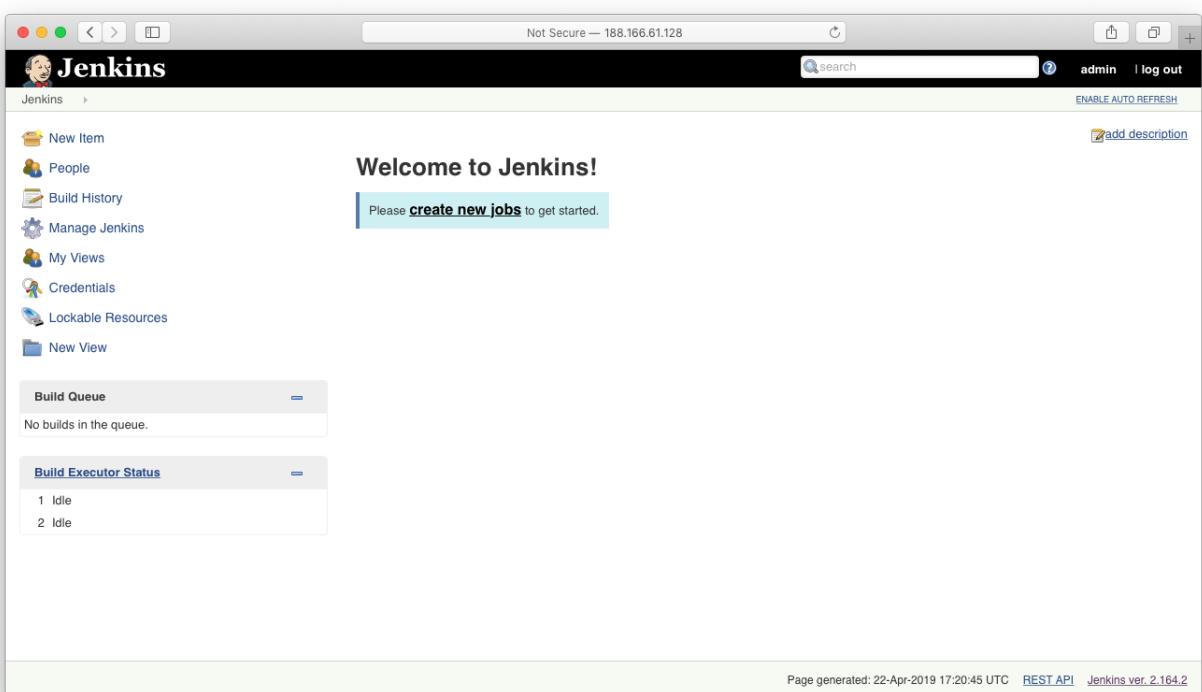
Figuur 6.12: Opzetten Jenkins, stap 7



Figuur 6.13: Opzetten Jenkins, stap 8



Figuur 6.14: Opzetten Jenkins, stap 9



Figuur 6.15: Opzetten Jenkins, stap 10



## **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.

Onderzoeks vragen:

- 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, configurerbaarheid, 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<sup>1</sup>. 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.

---

<sup>1</sup><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 zich gespecialiseerd in het maken van ERP pakketten. Dat is software dat alle processen van het bedrijf opneemt<sup>2</sup>. 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-

---

<sup>2</sup><https://www.sap.com/products/enterprise-management-erp.html>

en nadelen van de verschillende tools onderzoeken op vlak van snelheid, configurerbaarheid 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<sup>3</sup>. 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

---

<sup>3</sup><https://about.gitlab.com>

kost<sup>4</sup>. 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 praktijk voorbeeld 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 configurerbaarheid met de tools die Amista gebruikt en beschikbare documentatie.

---

<sup>4</sup><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](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.

- 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>
- 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>