HO
GENT

Faculteit Bedrijf en Organisatie

Microservice integration patterns on SAP order-to-cash process.

Lyva Van Damme

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

Promotor:
Karin Samyn
Co-promotor:
Nicolas Pauwelyn

Instelling: HoGent

Academiejaar: 2018-2019

Tweede examenperiode

Faculteit Bedrijf en Organisatie

Microservice integration patterns on SAP order-to-cash process.

Lyva Van Damme

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

Promotor:
Karin Samyn
Co-promotor:
Nicolas Pauwelyn

Instelling: HoGent

Academiejaar: 2018-2019

Tweede examenperiode

Woord vooraf

Deze bachelorproef werd geschreven voor het behalen van het bachelordiploma Toegepaste Informatica. Eerst was het niet gemakkelijk om een onderwerp te vinden. Ik wist niet goed waar ik het over wou hebben. Ik zou daarvoor graag mijn stagebedrijf, Delaware, voor het voorstellen van een onderwerp. Dankzij hun kon ik mijn bachelorproef doen over een onderwerp dat mij echt interesseerde. Ik zou ook graag mijn ouders bedanken voor de steun die zij mij gaven. Voor hun geduld bij de stressvolle situaties. Ook wil ik mijn broer en zus bedanken voor hun hulp. Ik zou graag mijn mede-studenten bedanken voor alle hulp die ze mij gaven.

Samenvatting

Inhoudsopgave

Lijst van figuren

Lijst van tabellen

1. Inleiding

De inleiding moet de lezer net genoeg informatie verschaffen om het onderwerp te begrijpen en in te zien waarom de onderzoeksvraag de moeite waard is om te onderzoeken. In de inleiding ga je literatuurverwijzingen beperken, zodat de tekst vlot leesbaar blijft. Je kan de inleiding verder onderverdelen in secties als dit de tekst verduidelijkt. Zaken die aan bod kunnen komen in de inleiding (**Pollefliet2011**):

- context, achtergrond
- afbakenen van het onderwerp
- verantwoording van het onderwerp, methodologie
- probleemstelling
- onderzoeksdoelstelling
- onderzoeksvraag
- ...

1.1 Probleemstelling

Dit onderwerp werd voorgesteld door Delaware. De doelgroep is dus Delaware.

Uit je probleemstelling moet duidelijk zijn dat je onderzoek een meerwaarde heeft voor een concrete doelgroep. De doelgroep moet goed gedefinieerd en afgelijnd zijn. Doelgroepen als “bedrijven,” “KMO’s,” systeembeheerders, enz. zijn nog te vaag. Als je een lijstje kan maken van de personen/organisaties die een meerwaarde zullen vinden in deze bachelorproef (dit is eigenlijk je steekproefkader), dan is dat een indicatie dat de doelgroep goed gedefinieerd is. Dit kan een enkel bedrijf zijn of zelfs één persoon (je co-promotor/opdrachtgever).

1.2 Onderzoeksvraag

De algmene onderzoeksvraag is: "Hoe microserivce integration patterns een order-to-cash proces in SAP kan beïnvloeden?". De algemene vraag delen we op in volgende puntjes:

- Wat zijn microservices?
- Hoe kan er overgeschakeld worden naar een microservice architectuur?
- Welke aanpassingen kunnen of moeten er gebeuren om de microservices te laten werken?
- Hoe zal de communicatie tussen de verschillende microservices werken?
- Hoe zit een order-to-cash (OTC) proces er uit?
- Welke business requirements heeft een order-to-cash proces?
- Welke invloed heeft de microservice architectuur op de performance van een order-to-cash proces?
- Welke aanpassingen moeten er gebeuren om microservices toe te passen?

Dit zal een theoretische studie zijn. Er zal geen proof of concept komen wegens tijdsgebrek.

1.3 Onderzoeksdoelstelling

Het doel van deze paper is het onderzoeken van de effecten van een microservices architectuur op een OTC in SAP. Het doel houdt ook in dat we aan de hand van een zes-stappen plan, theoretisch, gaan kijken hoe een OTC verandert door microservices.

1.4 Opzet van deze bachelorproef

De rest van deze bachelorproef is als volgt opgebouwd:

In Hoofdstuk 2 wordt een overzicht gegeven van de stand van zaken binnen het onderzoeksdomein, op basis van een literatuurstudie. Hierin zal er meer uitleg gegeven worden over microservices en het order-to-cash proces binnen SAP.

In Hoofdstuk 3 wordt de methodologie toegelicht en worden de gebruikte onderzoekstechnieken besproken om een antwoord te kunnen formuleren op de onderzoeksvragen. Hier zal het onderzoek worden uitgevoerd over hoe microservices het order-to-cash proces zullen beïnvloeden.

In Hoofdstuk 4, tenslotte, wordt de conclusie gegeven en een antwoord geformuleerd op de onderzoeksvragen. Daarbij wordt ook een aanzet gegeven voor toekomstig onderzoek binnen dit domein.

2. Stand van zaken

Dit hoofdstuk bevat je literatuurstudie. In dit onderdeel zal er diep in gegaan worden op het onderwerp. Delen van het onderwerp worden grondig onderzocht om later tot een conclusie te komen.

2.1 Microservices

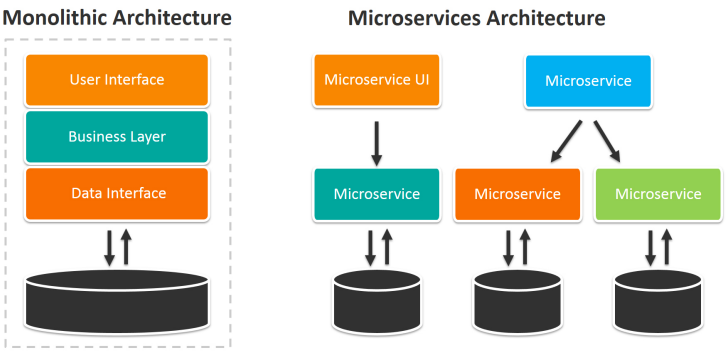
2.1.1 Definitie

Een term die vaak zal terug keren in deze paper is 'monolithic'. Deze term betekend het volgende: Bij een monolithic communiceren alle deeltjes van een applicatie met een grote databank of datastore. In een databank of datastore wordt er info en data opgeslaan om dit dan later te gebruiken. De data kan later gebruikt worden om het gepasseerde semester van een bedrijf te analyseren. Het artikel van Mauersberger (2017) geeft meer uitleg over microservices door de moeilijkheden bij een monolithic aan te kaarten. Bij een verandering binnen een monolithic architectuur, wordt er een heel nieuwe versie van de architectuur uitgebracht. Een verandering brengt een hoop extra werk mee. Dit omvat

- De volledige architectuur moet opnieuw getest worden.
- Deze architectuur kan heel complex worden bij het groter worden en toevoegen van functionaliteiten.
- De complete architectuur moet opnieuw gedeployed worden bij elke update.
- De impact van een verandering kan verkeerd ingeschat worden.
- Bij een fout in een proces, kan de volledige architectuur falen.

De definitie die te vinden is in dit artikel, gaat als volgt: "A method of developing software

applications as a suite of independently deployable, small, modular services in which each service runs a unique process and communicates through a well-defined, lightweight mechanism to serve a business goals.". Als je de definitie leest, zie je drie onderdelen. Het eerste onderdeel vertelt hoe een microservice in elkaar zit. Het is een onafhankelijke, kleine, modulaire services. Modulaire services zijn services waarbij veel delen uitwisselbaar zijn met diverse services. De services staan los van elkaar. Ze hebben geen invloed op elkaar. Daarom zijn ze ook onafhankelijk. Wordt er info gestuurd of gevraagd van services A dan zal dit geen invloed hebben op de andere services. De eenvoudige communicatie is een tweede eigenschap van microservices. Er is nood aan communicatie omdat sommige services wel data moeten uitwisselen om hun 'job' te kunnen doen. De communicatie kan gebeuren op verschillende manier. De manier die gekender is, is 'Messaging via a Message Broker'. Dit wil zeggen dat microservice A een bericht plaats op de wachtrij bij microservice B wanneer die data wil doorsturen. Dan Kan microservices B aan die data wanneer hij die nodig heeft. Ze zullen soms moeten wachten op elkaar maar ze zijn zo goed als onafhankelijk van elkaar. De derde eigenschap omvat dat een microservice wordt gemaakt in functie van een requirement uit de business. Elk product in de business heeft een doel dat moet voldoen aan eisen. Het unieke aan microservices is dat we ze gaan bekijken vanuit de eisen binnen de business. Het doel van microservices is, de problemen die te vinden zijn bij een monolithic, verhelpen. De vorige definite legde uit wat microservices zijn. Dit artikel zegt waar men microservices kan plaatsen. Er is dus één groot framework. Daar zitten meerdere onafhankelijke services in.



Figuur 2.1: Een monolithic architectuur naast een microservice architectuur. **Watts2018**

Figuur 2.1 is afkomstig uit artikel **Watts2018**. Zoals te zien in figuur 2.1 is er een groot verschil tussen een monolithic architectuur en die van een microservice. De monolithic wordt weergegeven in de linkerkant van de foto. Aan de rechterkant van de foto is een voorbeeld te zien van een microservice architectuur. Daar is duidelijk te zien dat elke microservice een eigen databank/datastore heeft. Voor elke functionaliteit wordt een microserives aangemaakt, die dan nog eens apart een databank voor zich krijgt. Als bij microservice A een probleem is dan heeft dit niet meteen impact op de andere services. De communicatie tussen microservice A en de anderen zal wel hinder ondervinden. Maar de andere microservices kunnen wel nog steeds onafhankelijk verder. Het artikel van u Series (g.d.) bestaat uit elf verschillende onderdelen. Eerst wordt er uitleg gegeven over wat microservices zijn. Dit is hun definitie van microservices: "A software architecting pattern that allows software to be developed into relatively small, distinct components. Each of the

components is abstracted by an API(s) and provides a distinct subset of the functionality of the entire application". Ook hier zien we weer het puntje passeren dat een microservice een klein componentje is van een groter geheel. Die eigenschap wordt heel hard benadrukt. Na het uitleggen van microservices, schrijven ze ook over hoe microservices zo scalable zijn. Met scalable bedoelen ze schaalbaarheid. De mogelijkheid van software om mee te groeien als het aantal gebruikers vermeerderd. Dus eigenlijk dat de software nog steeds even goed presteerd bij 10 gebruikers als bij 2 000 gebruikers. Ook lichten ze toe hoe belangrijk API's zijn binnen een microservice architectuur. API's zijn een set van definities die ervoor zorgen dat deeltjes in een programma met elkaar kunnen communcieren. Een voordeel van API's is dat je niet moet weten hoe de andere code werkt. Verder wordt er ook gepraat over de verschillen tussen een monolithic architectuur en een microservice architectuur.

Benetis2016a vertelt dat microservices uit dezelfde ideologie kotmt als Agile en DevOps. DevOps is een samenvoegsel van development en operations. Bij deze methode ligt de nadruk op de samenwerking en communicatie tussen verschillende partijen. Hier zijn de partijen de software engineers en andere IT specialisten. Deze ideologie omvat het volgende: het afbreken van kleine, traag evoluerende architectuur of monolithic en deze in microservices steken. Zoals te zien is in figuur X. We zien dat de monolithic alle puntjes in een kader heeft. Dit staat symbolisch voor het grote geheel dat eigen is aan een monolithic. Alles zit samen in één grote doos. Maar bij microservices is dit niet zo, daar zit elk deeltje/requirement in een aparte doos.

Figuur 2.2: Een monolithic vergeleken met een microservice. **Benetis2016a**

2.1.2 Het belang van microservices

Bij een monolithic kan het aanpassen van een deeltje, veel werk vragen. Dit werd meer uitgelegd bij de definite van microservices. Microservices spelen gemakkelijker in op het periodieke opleveren van delen software. Deze technologie legt niet heel het framework plat als er deeltjes moeten bij gecodeerd worden. Microservices kunnen sneller inspelen op de Agile analyse/ontwikkel methode. Dit is ook een reden waarom microservices zo een opkomst kent. De analyse methode Agile werkt met periodieke opleveringen die kunnen gaan van twee weken tot een maand. In de periode wordt er gewerkt aan een functionaliteit of een eis van de klant. u Series (g.d.) haalt aan dat microservices van belang zijn bij het scalen van software. In het artikel van RDX (2016) worden er verschillende eigenschappen aangekaart. Microservices moeten een doel in de business vervullen. Naast

dit, zorgt microservices er ook voor dat bescherming eenvoudig wordt. Zoals te vinden is in het artikel van **Troisi2019** zijn er acht manieren hoe microservices bescherming bieden. NOG VERDER UITSCHRIJVEN. Het artikel van **Watts2018** geeft het belang van een microservice goed weer. Microservices beschermen het gehele systeem, bij een goede implementatie. Het toepassen van de agile methode is eenvoudiger bij microservices dan bij een monolithic. Met deze technologie is het eenvoudiger om een aanpassing te testen en te onderhouden. Niet de volledige architectuur moet opnieuw getest worden bij een aanpassing. Als team A een aanpassing doet aan hun microservice, zullen de andere teams geen hinder ondervinden.

2.1.3 Algemene aanpak om microservices te implementeren

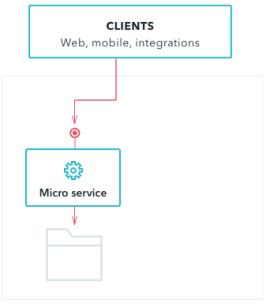
Het interessante artikel van Benetis (2016) over een 6-stappen plan om microservices te implementeren. Een paar woorden die meer verklaring nodig hebben voordat we verder gaan. Een gateway is een netwerkpunt dat dient als toegang tot een ander netwerk. Een gateway is een soort toegangspoort. Implementatie is een procesmatige invoering van een verandering of vernieuwing. Iets implementeren of vernieuwen.

In grote lijnen is dit het zes-stappen plan. Als eerste komt aan bod "serve a business purpose". Hierna komt "protect your stuff". Eens dat gebeurt is, zegt het artikel "see no evil, hear no evil". Dan komt "find your stuffäan bod. Hierna wordt de volgende stap "create a gatewayäangehaald. Als laatste komt "construct eventsäan de beurt. Bij eerste stap, "Serve a business purpose", zegt de titel al veel van wat er verwacht wordt. Een microservices is gebaseerd op een business requirement. En niet het doel dat het IT-team voor ogen heeft. Een voorbeeld van een business requirement is het ophalen van data om die dan te analyseren om daar later dan conclusies uit te trekken. Dit kan in een microservices gegoten worden. Eens het doel voor ogen is bij een microservices, moet er ook gekeken worden naar wat de microservices moet kunnen. Zo lang het bij enkele microservices blijft, is automated deployement etc. niet zo belangrijk. Maar eens we gaan scalen en meerdere microservices in één systeem steken zouden volgende puntjes toch self-sufficient moeten zijn:

- Geautomatiseerde implementatie
- Blootstelling aan andere systemen, toegankelijk eindpunt
- Opslag van data
- Schaalbaarheid en belasting

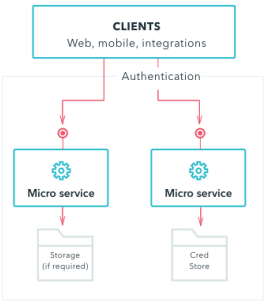
Zoals te zien is op figuur 2.2, is een microservices één klein deeltje in een groot geheel. Later in dit stappenplan zal de figuur ook uitgebreidt worden en zal het geheel duidelijk worden.

Na "Serve a business purpose"komt "Protect your stuff". Dit gaat over de bescherming van een microservices. Als het gaat over bescherming moet dit op elk moment gebeuren. Ookal heb je maar één à twee microservices, of honderden, bescherming is belangrijk. Het is belangrijk op over al de microservices een uniforme manier te vinden om ze te beschermen. De bescherming kan een requirement op zich zijn, dus ook dit kan in een microservices



Figuur 2.3: Een microservice dat voldoet aan een business requirement. Benetis (2016)

worden gestoken. Bescherming is een vage term daarom een korte uitleg van hoe een bescherming er zou kunnen uitzien. De meest bekende manier is natuurlijk authorisatie en authenticatie. Authorisatie is het verkrijgen van rechten om bijvoorbeeld een product toe te voegen op een site. Authenticatie is het aanmelden op facebook bijvoorbeeld. De controle of jij het wel echt bent. Een manier op de microservices te beschermen is gecentraliseerde session opslag. Hier zal verder in de thesis nog dieper op worden ingegaan. Kort uitgelegd betekend gecnetraliseerde session opslag dat de data van de user centraal opgeslagen staat. Zodat alle microservices de zelfde session data lezen en gebruiken. In figuur 2.3 is de

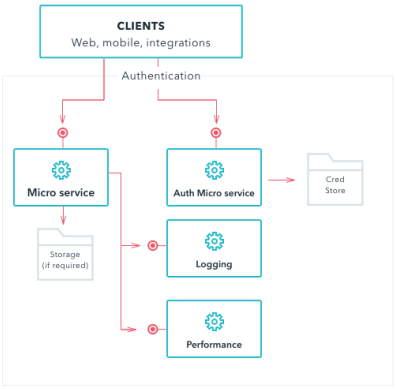


Figuur 2.4: Een microservice waar bescherming aan is toegevoegd. Benetis (2016)

authenticatie in een microservices gestoken.

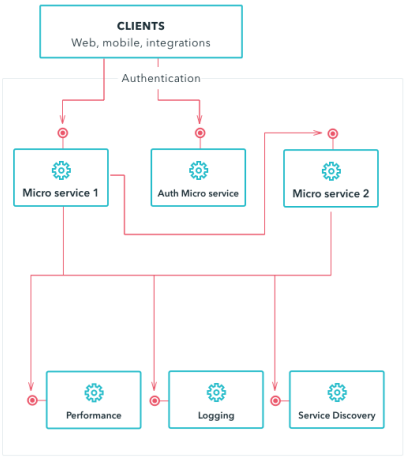
Na de eerste twee stappen komt "See no evil, hear no evil". Eens de microservice is opgezet en gedeployed, moet er gemonitord worden. Daarmee wordt bedoeld dat hoe de microserivces zich gedraagd goed moet bijgehouden worden. Alles zou goed moeten gelogd worden, zodat bij een probleem het geen moeite is om te vinden waar het probleem zich voordeed. Ook hier is het aangeraden om doorheen het hele systeem een uniforme manier van loggen aan te houden. Ook kan men hiervan een microservice maken. Dit wordt ook afgebeeld in figuur 2.4.

Als vierde stap komt er "Find your stuff". In deze stap wordt er gezocht naar een manier om de microservices met elkaar te communiceren. Hiermee wordt bedoelt hoe dat microservices A gegevens vraag aan microservices B. Die dat dan ook vraagt van een andere microservice. Een veel gebruikte techniek hiervoor is "service registry". Een service registry is een databank waar alle services met hun instanties en locatie woren opgeslaan.



Figuur 2.5: Een microservice waar er monitoring is aan toegevoegd om chaos te voorkomen. Benetis (2016)

Daar worden dan ook connecties in opgeslaan. Ook hier wordt er aangeraden om dat in een microservices te gieten. Zodat ook hiervan het gedrag kan gemonitord worden. In figuur 2.5 zie je hoe de service registry kan toegevoegd worden. In de figuur is die terug te vinden onder de naam "service discorvery".



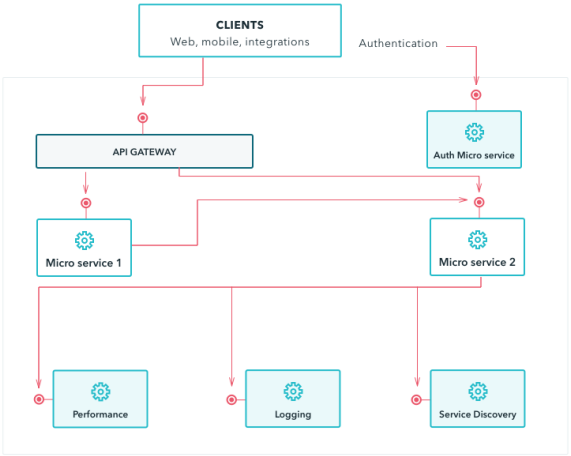
Figuur 2.6: Een microservice dat authenticatie als bescherming toepast. Benetis (2016)

Nu is er een service, bestaande uit microservices, dat beschermd is en kan doen wat het zou moeten doen. Maar niet de volledige service moet open en bloot gelegd worden. En daar zorgt stap 5, "Create a gateway", voor. Een API gateway kan een scherm zijn waar je gegevens op invult en mogelijke acties op doet en die spreken dan de correcte microservices aan. De taak van een gateway is voornamelijk zorgen dat request/aanvragen naar de juiste microservice worden doorgestuurd. Andere taken van een API gateway kunnen volgende zijn

- Beveiliging: een API gateway kan de binnenkomende aanvragen valideren.
- Prestatiegegevens kunnen geregistreerd worden.

- Omzetten van aanvragen in enkele of meerdere microservices.
- Abstractie van de clientinterface. Wanneer er van microservice verandert wordt, moet er niet van interface/scherm verandert worden.

In figuur 2.6 is te zien hoe zo een API gateway kan worden toegepast. Zo is ok te zien dat de authenticatie microservice er niet inzit. Die wordt apart gehouden. De request naar de authenticatie mogen niet langs de API gateway gaan. Omdat ze dan zo meteen 'binnen' zitten. Pas na authenticatie mag men request sturen naar de API gateway.

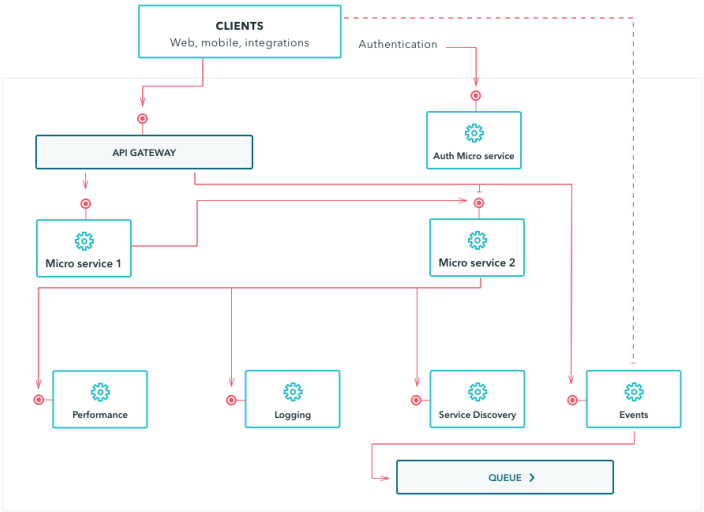


Figuur 2.7: Een microservice dat een gateway gebruikt. Benetis (2016)

Nu is er al een deftige architectuur aanwezig. Stap 6, "construct events", zal het plaatje dan ook compleet maken. De meeste microservices vragen aan asynchrone oplossing. Een niet-gelijktijdige verwerking van aanvragen. Een manier om asynchroon te werken, is werken met een queue of wachtrij. Een bekende manier om asynchroniteit toe te passen is publish/subscribe pattroon. Dit wil zeggen dat microservice A zijn berichten of data op een wachtrij gaat zetten. De microservices die data of berichten van microservice A moeten ontvangen, gaan zich abonneren op die wachtrij. Dus vanaf het moment dat microservice A iets op die wachtrij plaatst, krijgen de geabonneerden een melding en kunnen ze het bericht of de data gaan ophalen. Enkele voordelen van dit als asynchrone oplossing te gebruiken:

- Taken inplannen. Dit kan door deze gewoon op de wachtrij te plaatsen met een timestamp van wanneer deze moet gebeuren of door een wachtrij te maken voor geplande events.
- Abonneren op bepaalde events.
- Het asynchrone systeem laten bloot leggen zodat externe klanten verschillende notificaties kunnen handelen.

In figuur 2.7 wordt de volledige architectuur weergegeven. Daar wordt er ook mooi afgebeeld hoe men events pland. Als event A voor event B moet gebeuren dan zetten ze die zo op de wachtrij event A voor event B. Want een wachtrij werkt volgens het FIFO (first in first out) principe.



Figuur 2.8: Een microservice met asynchronisatie. Benetis (2016)

Benetis2016a beschrijft dat over volgende puntjes goed moet worden nagedacht voordat men de overschakeling maakt naar microservices:

- Heeft de organisatie de microservice architectuur wel nodig?
- Zijn de juiste competenties aanwezig? Microservices zijn in het algemeen complexer dan een monolithic. Zeker omdat dit iets nieuws is.
- Staat iedereen achter deze verandering?

Eens de beslissing gevallen is om over te schakelen, moet er beslist worden hoeveel de infrastructurele vernadringen van de scope inpalmen. Het verloop om microservices te creëeren gaat als volgt in dit artikel. Eerst het uit elkaar trekken van het bestaande systeem, om het dan in microservices te steken, is een goed begin. Zo moet er constant nagedacht worden over de algemene infrastructuur. Een groot valluik in het begin van het proces is een proof of concept maken. Dit eindigt meestal in een infrastructuur dat niet overeenkomt met de waarden van microservices. Het probleem ligt dan meestal bij een onduidelijke scope. De business requirements zijn meestal wel duidelijk, dit geldt dan meestal niet voor niet-functionele requirements. Daarnaast moeten ook nog volgende stappen gerealiseerd worden:

- Bescherming.
- Deployment automatisatie.
- Loggen en monitoren van microservices hun gedrag.

Velen komen niet tot deze stap, door de onderschatting van de overschakeling naar microservices.

2.1.4 De voordelen en nadelen van microserivces

In het artikel van u Series (g.d.) wordt er veel lofzang gedaan over microservices. Het gebruik van microservices zou ervoor zorgen dat de architectuur flexibeler wordt. Met flexibeler wordt bedoelt dat de architectuur zich kan 'aanpassen' of kan inspelen op verschillende situaties. Er kunnen microservices hergebruikt worden. Dankzij microservices is het hermodeleren, implementeren van nieuwe technologieën, ... Kleinere deeltjes zijn gemakkelijker te documenteren. De snelheid van microservices zijn een groot pluspunt. Hiermee wordt er geprobeert om aan te halen dat microservices sneller reageren omdat zo kleine, onafhankelijke services zijn. Ze moeten geen 'onnodige' stappen maken om de wens van de klant te vervullen. **Watts2018** geeft enkele voordelen van een microservice. Een developer is onafhankelijk. Ze hebben vrijheid. Ook het scalen van een microservice is veel eenvoudiger. Dit komt door dat microservices minder resources nodig hebben dan een volledige monolithic. Resources zijn hulpbronnen. Zoals al vaak aangehaald in deze bachelorproef, zijn microservices onafhankelijk en zouden ze daarom ook geen hulpbronnen nodig mogen hebben. Binnen een monolithic zijn deeltjes afhankelijk van elkaar en hebben elkaar dus nodig om goed te kunnen functioneren. De deeltjes binnen de monolithic hebben elkaar dus nodig en mogelijk als hulpbron. Een ander voordeel is bij het falen van een microservices, de andere microservices er geen last van zullen hebben. Dit komt door hun onafhankelijkheid. Benetis (2016) geeft aan dat volgende puntjes voordelen zijn van microservices:

- Sneller en gemakkelijker developen.
- Het refactoren van deeltjes is eenvoudiger door de onafhankelijkheid van de services.
- De schaalbaarheid is eenvoudiger dan bij een monolithic. We kunnen microservices gewoon 'klonen' of kopieëren.
- Het deployen van een onderdeel gaat sneller omdat het team gespecialiseerd is in die bepaalde service.
- Als er iets faalt dan is de impact veel kleiner dan bij een monolithic. Dit komt ook door de onafhankelijkheid van de services.

Maar om ervoor te zorgen dat dit allemaal vlot verloopt moeten er ook aanpassingen binnen in het bedrijf/ de organisatie gebeuren.

- Een project zal ingedeeld moeten worden in kleine requirements. De scope zal gedetailleerder moeten zijn.
- De teams zullen kleiner moeten worden gemaakt. Zodat er meer op de Agile methode kan gewerkt worden.
- Er zal een sterke band komen met DevOps. Dit komt omdat veel services volledige automatische deployment vragen.
- Ook de communicatie tussen de services zal beter moeten worden uitgedacht.
- Documenteren is belangrijk. Dit is niet enkel het geval voor microservices maar ook voor elk project.

2.1.5 Voorbeelden

In dit deeltje zal je meer te weten komen over hoe grote technologische bedrijven microservices toepassen en hoe ze naar deze technologie zijn overgeschakeld. Een term die hier vaak zal gebruikt worden, is een monolithic. Dit is de tegenhanger van microservices. Sommige zweren bij monolithic en anderen hebben gouden woorden voor microservices. De beslissing om één van de twee technieken te kiezen, ligt bij wat je precies nodig hebt. Wat de business nodig heeft.

Amazon

Mauersberger (2017) gaf een voorbeeld waarom Amazon overstapte. Zoals veel grote bedrijven is Amazon begonnen met een grote monolithic. Een van de nadelen die Amazon ondervond aan deze technologie is de moeilijkheid van het inschatten van de zwaarte op de servers. Daardoor verloor Amazon veel geld en was er nood aan herstructurering. III (2015) legt ook uit waarom Amazon is overgeschakeld. Het blijkt dat Amazon in 2001 de grootste monolithic was in de retail business. Doordat Amazon een snel groeiende onderneming was, werd de monolithic architectuur heel ingewikkeld. Amazon hun aanpak was, gooi wat je hebt niet weg maar maak het eenvoudiger.

Apple

Facebook

Netflix

2.2 Order-to-cash proces in SAP

2.2.1 Definite

Obrien2017 geeft aan dat een order-to-cash bestaat uit processen met business requirements. Dit proces start bij het plaatsen van een order en eindigt bij het innen van het geld. Het proces is in de grote lijnen hetzelfde. Het snel doornemen van het proces, geeft een vertekend beeld. Elk deeltje van dit proces heeft moeilijkheden en uitdagingen. Dit proces is heel belangrijk voor bedrijven. Het is de core van de business. Bij een slechte implementatie, kan je klanten verliezen of geld verliezen.

2.2.2 Technologie

Onderdelen van een order-to-cash proces

Er zijn vier grote onderdelen, namelijk:

- Voor-verkoopsactiviteiten.
- Het order proces.

- Order afwerking.
- Betaling.

Bij de voor-verkoopsactiviteiten verstaan we het contact dat moet worden gemaakt worden met klant. De klant moet overtuigd worden van het product. Na contact komt er al dan niet een offerte. Soms kan er ook van contact rechtstreeks naar een order gaan. Het orderproces bevat maar één onderdeel namelijk: de sales order. Binnen de order afwerking valt het leveren van goederen, het verzenden van goederen. **Kumaran2015** geeft een mooi overzicht van de stappen die een order to cash proces kan bevatten. Zo begint het artikel met een toelichting dat dit proces een core proces is van de business. Dit wil zeggen dat een bedrijf, zonder dit proces, geen winst kan maken. Het is een essentieel onderdeel van zo wat elk bedrijf. Hoe deze vorm krijgt binnen een bedrijf, is heel verschillend. Een OTC proces start met het ontvangen van orders. Daarna kan, dit gebeurt niet overal, er gekeken worden naar het krediet van de klanten. Mag deze klant wel nog een bestelling plaatsen? Hierna wordt het order opgenomen in het systeem. Het product wordt verzonden en geleverd. En als laatste wordt de factuur betaald. Dit is welliswaar het 'perfecte' verloop van een order to cash proces. Er zijn ook enkele uitdagingen verbonden met dit proces. In

Figuur 2.9: Order-to-cash proces volgens **Kumaran2015**.

de volgende opsomming vindt je er enkele:

- Hou de orders goed bij, anders loopt het fout vanaf het begin.
- Zonder automatisering verlies je veel tijd. Kostbare tijd.
- Zorgen dat de logistiek 'on point' is, is heel belangrijk. Dit is een belangrijk onderdeel binnen het proces.
- Bij wanbetalingen, moet de customer service ingrijpen. Ook zij zijn een belangrijk onderdeel van het proces.

Het managen van een OTC proces is even belangrijk als de andere aspecten. Een slecht gemanaged proces kan op lange termijn duurder uitkomen dan een onderdeel dat nog niet geautomatiseerd is.

Wat biedt SAP zelf aan voor microservices

2.2.3 Een order-to-cash proces vanuit de business

2.2.4 Het proces afstemmen met de business

2.3 Requirements van de business

3. Methodologie

TODO: Hoe ben je te werk gegaan? Verdeel je onderzoek in grote fasen, en licht in elke fase toe welke stappen je gevolgd hebt. Verantwoord waarom je op deze manier te werk gegaan bent. Je moet kunnen aantonen dat je de best mogelijke manier toegepast hebt om een antwoord te vinden op de onderzoeksvraag.

4. Conclusie

Curabitur nunc magna, posuere eget, venenatis eu, vehicula ac, velit. Aenean ornare, massa a accumsan pulvinar, quam lorem laoreet purus, eu sodales magna risus molestie lorem. Nunc erat velit, hendrerit quis, malesuada ut, aliquam vitae, wisi. Sed posuere. Suspendisse ipsum arcu, scelerisque nec, aliquam eu, molestie tincidunt, justo. Phasellus iaculis. Sed posuere lorem non ipsum. Pellentesque dapibus. Suspendisse quam libero, laoreet a, tincidunt eget, consequat at, est. Nullam ut lectus non enim consequat facilisis. Mauris leo. Quisque pede ligula, auctor vel, pellentesque vel, posuere id, turpis. Cras ipsum sem, cursus et, facilisis ut, tempus euismod, quam. Suspendisse tristique dolor eu orci. Mauris mattis. Aenean semper. Vivamus tortor magna, facilisis id, varius mattis, hendrerit in, justo. Integer purus.

Vivamus adipiscing. Curabitur imperdiet tempus turpis. Vivamus sapien dolor, congue venenatis, euismod eget, porta rhoncus, magna. Proin condimentum pretium enim. Fusce fringilla, libero et venenatis facilisis, eros enim cursus arcu, vitae facilisis odio augue vitae orci. Aliquam varius nibh ut odio. Sed condimentum condimentum nunc. Pellentesque eget massa. Pellentesque quis mauris. Donec ut ligula ac pede pulvinar lobortis. Pellentesque euismod. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent elit. Ut laoreet ornare est. Phasellus gravida vulputate nulla. Donec sit amet arcu ut sem tempor malesuada. Praesent hendrerit augue in urna. Proin enim ante, ornare vel, consequat ut, blandit in, justo. Donec felis elit, dignissim sed, sagittis ut, ullamcorper a, nulla. Aenean pharetra vulputate odio.

Quisque enim. Proin velit neque, tristique eu, eleifend eget, vestibulum nec, lacus. Vivamus odio. Duis odio urna, vehicula in, elementum aliquam, aliquet laoreet, tellus. Sed velit. Sed vel mi ac elit aliquet interdum. Etiam sapien neque, convallis et, aliquet vel, auctor non, arcu. Aliquam suscipit aliquam lectus. Proin tincidunt magna sed wisi. Integer blandit

lacus ut lorem. Sed luctus justo sed enim.

Morbi malesuada hendrerit dui. Nunc mauris leo, dapibus sit amet, vestibulum et, commodo id, est. Pellentesque purus. Pellentesque tristique, nunc ac pulvinar adipiscing, justo eros consequat lectus, sit amet posuere lectus neque vel augue. Cras consectetuer libero ac eros. Ut eget massa. Fusce sit amet enim eleifend sem dictum auctor. In eget risus luctus wisi convallis pulvinar. Vivamus sapien risus, tempor in, viverra in, aliquet pellentesque, eros. Aliquam euismod libero a sem.

Nunc velit augue, scelerisque dignissim, lobortis et, aliquam in, risus. In eu eros. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Curabitur vulputate elit viverra augue. Mauris fringilla, tortor sit amet malesuada mollis, sapien mi dapibus odio, ac imperdiet ligula enim eget nisl. Quisque vitae pede a pede aliquet suscipit. Phasellus tellus pede, viverra vestibulum, gravida id, laoreet in, justo. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Integer commodo luctus lectus. Mauris justo. Duis varius eros. Sed quam. Cras lacus eros, rutrum eget, varius quis, convallis iaculis, velit. Mauris imperdiet, metus at tristique venenatis, purus neque pellentesque mauris, a ultrices elit lacus nec tortor. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent malesuada. Nam lacus lectus, auctor sit amet, malesuada vel, elementum eget, metus. Duis neque pede, facilisis eget, egestas elementum, nonummy id, neque.

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

Hoe Microservice integration patterns een order to cash proces in SAP beinvloedt. Dit onderwerp werd gekozen omdat deze nieuwe technology een interessante invloed kan hebben op de order-to-cash proces. Dit is een manier om een proces robuuster te maken. In de meeste software wordt gebruik gemaakt van één grote databank of meerdere databanken die in staan zijn om meerde services te voorzien van data. Bij microservice integration patterns wordt voor elke service een aparte databank opgesteld. Dit is maar een klein deeltje van een microservice. De microservices moeten voldoen aan business requirements. SAP zelf heeft ook al veel ondernomen omtrend microservices. Eén van hun oplossingen is Kyma. Maar de belangrijkste vraag is namelijk: Hoe microservice integration patterns een order-to-cash in SAP beïnvloedt. Deze bachelorproef zal voor het grootste deel een theoretische vergelijking zijn. Omdat deze studie als bachelorproef dient, is er maar beperkte tijd en resources om een onderzoek te doen.

A.2 Literatuurstudie

Over het onderwerp "Microservice Integration Patterns on Order-to-Cash proces in SAP", zijn er nauwelijks thesissen te vinden. De meeste informatie komt uit artikels die meer uitleg geven over microservices en artikels met een uitgebreide beschrijving over wat

order-to-cash inhoudt. Voor wie denkt dat microservices iets nieuws is, zit er een beetje naast. Grote bedrijven zoals Netflix, Twitter, Amazon en facebook maken al gebruik van deze technologie. Ciber, 2018

A.2.1 Wat zijn microservices?

Het bouwen van aparte functies/modules met hun eigen interface en methoden. Deze manier van werken is in het voordeel van Agile. Bij Agile wordt er gewerkt met deeltjes software opleveren en opgeleverde software, daar wordt er zo goed als niks meer aan veranderd. Microservices worden onderverdeeld aan de hand van business requirements. Deels zorgen microservices ervoor dat er beter moet worden samengewerkt met de business.

A.2.2 Waarom microservices gebruiken

In het artikel van Gunaratne, 2018 werd besproken hoe je een microservice werkt. En waarom deze gebruikt worden. Volgens dit artikel zijn microservices een goede, nieuwe techniek die op lange termijn huidige SOA's kan vervangen.

Atrash, 2018 beschrijft waarom je deze techniek kunt gebruiken, in de plaats van de kleinere SOA-services. Ook hier wordt verwezen naar de belangrijkheid van de requirements van de business. Door de grootte van deze services, is er de mogelijkheid om te caching.

Devoteam, 2018 legt uit waarom microservices een kleine-SOA is. Een microservice omvat bepaalde, aanvullende, concepten omliggend deze 'kleinere services' en dit is waar ze beginnen met aantonen van de verschillen.

Is de integratie van microservices wel mogelijk? Deze vraag beantwoordt door Van Bart, 2018. Software wordt meestal nog geimplementeerd in de 3-tiers manier. Ook wel monolithic genoemd. De applicatie is uit een alleenstaande unit gemaakt. Eén verandering heeft een impact op de volledige applicatie. Is dit dan een geldige reden om voor microservices te kiezen? Dat hangt af van wat je applicatie percies nodig heeft. Niet alle applicaties worden er beter van om een microservice te implementeren. Een microservice bestaat er uit om op zichzelf te werken. Dit wordt uitgelegd in het volgende deel.

A.2.3 Principes voor Microservices Integration

De principes van microservices integration werden uitgelegd in volgend artikel Aradheye, 2018 In het artikel wordt op een duidelijke manier uitgelegd hoe microservices worden gebruikt. Microservices worden het best opgesteld aan de hand van business units. Een microservice wordt benaderd vanuit de business requirements. Deze hebben, volgens dit artikel, een betere performantie dan de huidig gebruikte techniek. Microservices worden opgedeeld in verschillende klasses. Bijvoorbeeld:

- één voor klantendata

- één voor bestellingen
- één voor "wil-ik"lijsten

De belangrijkste eigenschappen van microservices zijn:

- Microservice bestaat uit meerdere componenten
- Gemaakt voor de business
- Microservice maakt gebruik van simpele routing
- Een microservice is gedecentraliseerd
- Een microservice werkt zelfstandig

A.2.4 Order-to-cash in SAP

Order-to-cash is een van de vele processen in SAP die vast gelegd zijn. Dit proces legt uit hoe men van een bestelling naar de inning van het geld gaat. Er zijn verschillende versies van hoe het proces gaat. Volgens Akthar, 2018 verloopt het proces als volgt: er wordt een order geplaatst dan wordt die bestelling geleverd, daarna wordt er een factuur opgesteld en als laatste wordt het geld geind. Het proces op zich is niet ingewikkeld. OpenSAP, 2018 geeft veel meer uitleg over wat er achter de schermen gebeurd. Bij dit proces wordt de financiële kant van SAP aangesproken, ook de verkoop en distributie alsook de stock worden aangesproken. Een gedetailleerder proces houdt in dat er een sales order gemaakt wordt. Dan wordt de stock bekeken. Afhankelijk van de beschikbaarheid van de goederen kan er een levering gepland worden. Zijn de goederen beschikbaar, dan kan er na de planning van de levering, effectief geleverd worden. Als volgt wordt er een factuur opgemaakt. Als laatste komt dan het betalingsproces.

A.2.5 Kyma

Kyma is een open-source project van SAP. Het is vooral gebaseerd op Kubernetes. Op deze manier kan je oplossingen in de Cloud maken. Kyma is special omdat zij zo goed als alle oplossingen op één plek hebben. Zij hebben een application connector. Kyma is serverless. Ze maken service management eenvoudiger. Kyma, 2019 Volgens het artikel van Semerdzhiev, 2018 is er meer nood aan openheid en een modernere architectuur. Dat is ook de reden waarom Kyma een open source project is. Kyma ondersteund container-based werken (zoals docker) alsook cloud-native apps.

A.3 Methodologie

In dit werk gaan we onderzoeken op welke manier microservices een invloed zou kunnen hebben op een order-to-cash proces in SAP. De services die ze nu gebruiken vergelijken met microservices. Kunnen microservices de werking van SAP versnellen en performanter maken bij fouten? Welke messaging manier zou het beste kunnen zijn. Eerst willen we het volledige order-to-cash proces verstaan. Dan gaan we gaan onderzoeken welke

verschillende mogelijkheden er zijn in verband met microservices. Kyma, PaaS.io of zijn er nog andere die een grote rol kunnen spelen. Ook moet er gekeken worden welke manier van communiceren tussen de microservices het beste is. Voor dit onderzoek zal er veel literatuur studie gedaan worden.

A.4 Verwachte resultaten

Naar de gelezen literatuur kijkende, zou Kyma eigenlijk de beste oplossing moeten zijn. Deze is namelijk zelf afkomstig van SAP. Dit zou een goede oplossing moeten zijn. Maar zijn microservices wel haalbaar in een order-to-cash proces in SAP. Eerst zal dit duidelijk moeten worden vooraleer we gaan kijken naar welke microservice de best mogelijk noden opvult.

A.5 Verwachte conclusies

De conclusie die we uit dit onderzoek kunnen trekken: microservices integration patterns zijn voordeliger en gebruiksvriendelijker dan het huidige systeem is voor de mensen die de software gaan gebruiken. Bij fouten aan een service, zal het platform nog beschikbaar zijn. Het risico is wel dat door de relatief nieuwe techniek, er enkele dingen niet zullen lopen zoals we zouden willen. Het is mogelijk dat we maar tot een gedeeltelijke conclusie komen.

Bibliografie

Akthar, J. (2018). What order-to-cash cycle controls in SAP ensure compliance? *SAP*.

Aradheye, Y. (2018). Principles for Microservices Integration. *DZone*.

Atrash, M. A. (2018). Why microservice architecture? *Develoteam*.

Benetis, R. (2016, november 11). A 6-point plan for implementing a scalable microservices architecture. *https://www.devbridge.com/articles/a-6-point-plan-for-implementing-a-scalable-microservices-architecture/*.

Ciber. (2018). Microservices: het einde van de SAP spaghetti? *https://www.ciber.nl/blog/deel-2-microservices-het-einde-van-de-sap-spaghetti*.

Devoteam. (2018). Microservices: just another word fo 'tiny-SOA'? *Devoteam*.

Gunaratne, I. (2018). Wiring Microservices, Integration Microservices & APIs. *Container-Mind*.

III, S. M. F. (2015, oktober 8). What led Amazon to its own microservices architecture. *https://thenewstack.io/led-amazon-microservices-architecture/*.

Kyma. (2019, februari 13). What is kyma? *https://kyma-project.io/*.

Mauersberger, L. (2017, juli 18). Why Netflix, Amazon, and Apple Care about microservices. *https://blog.leanix.net/en/why-netflix-amazon-and-apple-care-about-microservices*.

OpenSAP. (2018). Order-to-cash overview. *Order-to-cash overview*.

RDX. (2016, december 13). REST in Peace: Microservices vs monoliths in real-life examples. *https://medium.freecodecamp.org/rest-in-peace-to-microservices-or-not-6d097b6c8279*.

Semerdzhiev, K. (2018, juli 24). Introducing project Kyma. *https://kyma-project.io/blog/introducing-project-kyma/*.

u Series, A. (g.d.). Microservices 101: Understanding and leveraging microservices. *https://www.programmableweb.com/api-university/microservices-101-understanding-and-leveraging-microservices*.

Van Bart, A. (2018). Microservices and integration: Doing IT rigth? *devoteam*.