SaaS Enablement

Bestaande Software Aanbieden als Cloud –dienst

Koen Vanderkimpen

Smals SECTIE ONDERZOEK 07/2015 • RESEARCH NOTE • 37

Abstract

De druk om cloud-technologie te gaan gebruiken wordt steeds groter, en de public cloud zal een groot stuk van de koek voor zich nemen dankzij de belofte van grotere versatiliteit en lagere kosten. Wanneer men gevoelige data echter niet wil uitleveren aan multinationale cloud-spelers, zal men van de private cloud een valabel alternatief moeten maken, dat er via een verhoogde bruikbaarheid mee kan wedijveren.

SaaS Enablement, het opwaarderen van een applicatie zodat het mogelijk wordt om deze in een cloud-context als SaaS aan te bieden, is één van de elementen die een private cloud meer aantrekkingskracht kunnen geven. In deze tekst bepreken we wat de mogelijkheden zijn voor deze strategie, en hoe we ze kunnen toepassen met een zo min mogelijke effort.

Het resultaat is Instance as a Service, een relatief snelle en eenvoudige manier om te SaaS-Enablen, die maximaal nut haalt uit een onderliggend aPaaS-platform.

SaaS Enablement is echter slechts een stukje van de puzzel: extra meerwaarde zal moeten komen uit verregaande integratiemogelijkheden (“Enterprise Enablement”) en extra dienstverlening.

Résumé

La technologie cloud s'impose de plus en plus et le cloud public se taillera la plus grande part du gâteau vu sa promesse de versatilité supérieure et de coûts inférieurs. Toutefois, si l'on ne souhaite pas confier des données sensibles à des fournisseurs de cloud multinationaux, il convient de faire du cloud privé une alternative valable, capable de rivaliser par une utilisabilité accrue.

Le SaaS Enablement, qui consiste à remanier une application de telle sorte qu'elle puisse être proposée dans un contexte cloud sous forme de SaaS, figure parmi les moyens susceptibles d'augmenter le pouvoir d'attraction d'un cloud privé. Dans ce texte, nous aborderons les possibilités pour cette stratégie et verrons comment les appliquer moyennant un effort minimal.

Le résultat s'appelle Instance as a Service, une méthode de SaaS Enablement relativement simple et rapide, qui tire au maximum profit d'une plateforme aPaaS sous-jacente.

Le SaaS Enablement n'est toutefois qu'une pièce du puzzle : la grande valeur ajoutée est à chercher dans les possibilités d'intégration poussée (“Enterprise Enablement”) et les services supplémentaires.

Inhoud

1. INLEIDING

De cloud is vandaag de dag nog steeds een sterke hype, en, qua technologie, een onafwendbare evolutie. Op businessniveau biedt ze tal van nieuwe mogelijkheden, maar ook risico's. Het is dus van belang goed voorbereid te zijn om op een optimale manier te kunnen instappen in het cloud-verhaal.

Een spanningsveld dat momenteel sterk speelt bij vele organisaties, is dat van *public* versus *private*. Meer bepaald: zal men gaan gebruikmaken van diensten in de public cloud, aangeboden door externe, veelal Amerikaanse, leveranciers, of probeert men eerder een eigen cloud uit te bouwen, in het eigen datacenter, waarop men dan zelf de nodige diensten kan aanbieden en uitbaten? De eerste optie zal typisch de goedkoopste zijn, aangezien de public cloud spelers kunnen profiteren van een enorm schaalvoordeel. Een private cloud wordt daarentegen als veiliger en betrouwbaarder beschouwd, omdat men de volledige controle behoudt over de eigen infrastructuur en – vooral – data[1].

In deze nota zullen we de tweede visie aanhangen: die van het belang van controle over de eigen data, zeker wanneer het gevoelige, persoonsgebonden of zelfs medische data kan betreffen. In deze visie willen we dus gebruikmaken van een private cloud, maar om te kunnen concurreren met de goedkopere public cloud, zullen we op zoek moeten gaan naar andere vormen van meerwaarde, die liefst ook verder gaan dan louter het controlebehoud. We zullen dan ook kijken naar een mogelijkheid om een private cloud functioneel zo aantrekkelijk mogelijk te maken, en dit liefst aan een zo laag mogelijke meerkost.

1.1.1. SaaS Enablement

Specifiek in deze tekst zullen we onderzoeken wat de mogelijkheden zijn om software, die reeds werd ontwikkeld voor stand-alone gebruik, aan te bieden als cloud-dienst. We gaan dus kijken wat we kunnen doen om deze bruikbaar te maken als Software as a Service (SaaS). Dit mogelijk maken (= *enablen*) zullen we benoemen als *SaaS Enablement*.

Er zijn hiervoor twee goede redenen: ten eerste stijgt de toegevoegde waarde van SaaS binnen IT, maar ook binnen de algemene economie, nog steeds. Het is dus zeker de moeite waard om hierin te investeren. Daarnaast zien we dat er in de IT-wereld reeds enorm veel nuttige software beschikbaar is, en dat we meestal slechts de keuze hebben tussen een niet-cloud versie, of SaaS die enkel in een public cloud beschikbaar is.

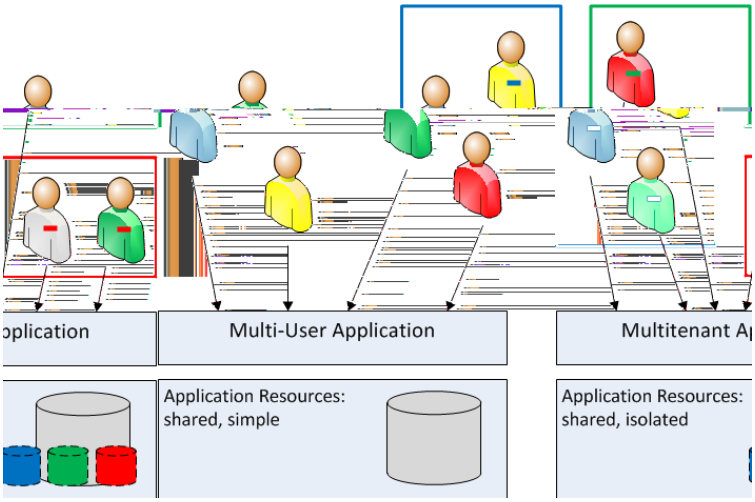
In de rest van de inleiding volgt eerst een glossarium. Daarna zullen we nog iets verder uitweiden over de cloud, en dan vooral over SaaS, en geven we nog iets meer algemene uitleg over het nieuwe concept SaaS Enablement, alsook een paar alternatieven ervoor. In het daaropvolgende hoofdstuk gaan we dieper in op de vereisten om te kunnen SaaS-Enablen. Hoofdstuk drie zal dan een effectieve strategie voor SaaS Enablement uit de doeken doen. Daarna volgen nog de conclusie en aanbevelingen.

[1] Dit is zo indien de private cloud on-premise wordt gehost, zie het glossarium voor details.

1.2. Glossarium

Vooraleer we verder gaan met de tekst, geven we een korte uitleg van een aantal van de termen die we geregeld zullen gebruiken, en waarvan de uitleg te veel zou storen in de tekst zelf. Een term is perfect over te slaan voor wie er reeds goed mee bekend is.

- **Multi-tenancy**. Dit is een eigenschap van een IT-dienst waarbij één in-stallatie van de dienst (hetgeen een abstract begrip is: dit kan gaan van één server tot een installatie bovenop een platform waaronder een hele cluster van servers draait) meerdere klanten of 'tenants' kan be-dienen. Hierbij bestaan de klanten uit groepen van gebruikers die een bepaalde eigenschap de-len, zoals b.v. de werk-nemers van één bedrijf. Belangrijk is daarbij, on-danks het delen van één enkel systeem, dat er een functionele isolatie is van de klanten, waarbij ze b.v. geen toegang hebben tot elkaars data. Daarnaast beoogt men ook een technische scheiding, zodat b.v. een overbelasting in het deel van één tenant geen overlast veroorzaakt bij de andere. Multi-tenancy is niet te verwarren met een multi-user systeem dat wel door meerdere gebruikers kan gebruikt worden, maar niet de isolatie tussen tenants voorziet.



Figuur 1 : Multi-User vs Multitenant

- **1-N model**. Een gebruiksmodel waarbij één zaak wordt gebruikt door meerdere partijen (zoals bij multi-tenancy) of één partij meerdere zaken kan gebruiken.
- **Git**. Git is een VersieControleSysteem (VCS). Het wordt gebruikt om de verschillen tussen verschillende versies van bestanden en groepen van bestanden bij te houden en te beheren. Het is bovendien een geDistribueerd VCS (DVCS), wat wil zeggen dat het geen centrale server behoeft. Andere voorbeelden van VCS zijn CVS, Subversion en Mercurial. Git is momenteel het populairste VCS voor nieuwe projecten.
- **Private Cloud**. In dit cloudmodel ('cloud deployment model') blijft een instelling baas over de eigen cloud en is deze bestemd voor privégebruik door deze instelling. Ze kan de infrastructuur dus volledig zelf beheren. Vaak wordt er vanuit gegaan dat dit betekent dat de (fysieke) infrastructuur zich ook on-premise bevindt. Dit is echter niet altijd het geval; er bestaat namelijk ook Virtual Private Cloud (VPC). Een VPC kan men huren in de public cloud, en is een goed geïsoleerde cluster van door de huurder beheerde middelen. Vanwege de vereiste controle, situeert het aanbod van VPC's zich vooral in de IaaS-markt. Wanneer we het in deze tekst over

private cloud hebben, doelen we op een private cloud op eigen fysieke infrastructuur.

- **Community Cloud**. In dit cloudmodel beheert een groep van bedrijven of instellingen samen een cloud, of ze besteedt dit beheer uit aan één van de groepsleden of een goed gecontroleerde derde partij. Voor de rest lijkt dit model zeer sterk op private cloud, en, indien het er één betreft die zich niet (virtueel) in de public cloud bevindt, is deze tekst dus eveneens van toepassing op een community cloud.
- **Tier**. Een tier is een laag, binnen een software-architectuur bestaande uit meerdere lagen (*multi-tier architecture*, *N-tier architecture*). Het idee achter deze soort architecturen is om de software makkelijker te kunnen opdelen in afzonderlijk uitrolbare, onderhoudbare en soms zelfs afzonderlijk ontwikkelbare componenten. Iedere component heeft dan zijn eigen verantwoordelijkheden. Typisch zal men in een tier een groot blok functionaliteit afsplitsen zoals databeheer, applicatielogica en/of presentatie. Een veel voorkomend voorbeeld bekomt men wanneer men de drie laatstgenoemde gebruikt als opsplitsing; dit noemt dan de *3-tierarchitectuur*.

1.3. De Cloud & haar drie Toeleveringsmodellen

Er zijn mensen die bij het horen van de term “Cloud” eens goed met hun ogen rollen en ons fijntjes weten te vertellen dat “dat al heel lang bestaat; een server waar je via het internet aan kan, dat is niks nieuws onder de zon”. In zekere zin hebben ze een punt: al lang voordat de cloud bestond, was er technologie waarmee men servers en applicaties via het internet kon besturen. Ook virtualisatie van servers, zelfs op grote schaal, bestaat al ettelijke jaren en veel langer dan de cloud. En ja, technologie voor gedistribueerde computersystemen die elastische schaalbaarheid ondersteunen: ook dat bestond al eerder.

Wat maakt de cloud dan zo uniek, en toch ook zo vernieuwend? Het antwoord zit hem in het gebruiksmodel: het alomtegenwoordige “*as a Service”*-verhaal. Het uiteindelijke doel van cloud-technologie is om computing (servers, applicaties, platformen, …) zo makkelijk toeleverbaar en bruikbaar te maken als elektriciteit of water: je hoeft als het ware maar de kraan open te draaien om rekenkracht, webservers, databases, etc. ter beschikking te hebben. Concreet werkt dit door eenvoudig bruikbare webinterfaces aan te bieden om alles aan te vragen en te beheren, en daarnaast ook door het facturatiemodel meer en meer op dat van de gas- of watermaatschappij te laten lijken: *pay as you go* moet ervoor zorgen dat je enkel betaalt wat je verbruikt. IT-diensten als nutsvoorzieningen dus. *Computing as a Service*.

> ***Cloud computing is een stijl van computing die wordt gekenmerkt door het aanbieden van IT-diensten komende uit een grote pool, door alle gebruikers gedeeld, via internettechnologie, op een elastisch schaalbare manier en dit via on-demand self-service en gefactureerd via pay-as-you-go.***

Samengevat is de cloud dus een amalgaam van reeds bestaande, doch steeds evoluerende IT-technologieën, die samen, en op grote schaal gebruikt, een geheel nieuw verbuiksmodel voor IT toelaten.

Het verbruiks- of toeleveringsmodel is verder op te splitsen in drie grote families: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) en Software as a Service (SaaS). Hierover meer uitleg in de volgende secties.

1.3.1. Infrastructure en Platform as a Service (IaaS & PaaS)

We kunnen de drie toeleveringsmodellen van de cloud een beetje zien als een technologie-stack. Dit noemt men dan de *Cloud Stack*. De onderste en de middelste laag van deze stack worden gevormd door IaaS en PaaS.

Infrastructure as a Service is de basislaag. Met deze dienstverlening kan men virtuele machines, meestal voor gebruik als servers, over het internet aanmaken, beheren en gebruiken. Het bekendste voorbeeld hiervan is de pionier Amazon Elastic Compute Cloud (EC2)[2].

Figuur 2 : De cloud stack

IT-bedrijven en IT-departementen van andere bedrijven zijn een paar jaar geleden gretig op IaaS beginnen springen omwille van, onder andere, de volgende voordelen: men hoeft niet langer een eigen fysiek datacenter te beheren, en men hoeft niet precies op voorhand te weten hoevéél infra-structuur men zal nodig hebben om de behoeften te vervullen. Daarnaast biedt IaaS ook een verregaand niveau van controle: eens men de virtuele machines draaiende heeft, kan men er in belangrijke mate zijn zin mee doen. Men kan er op installeren wat men wil, en de eigen controle is dus nog vrij groot.

IaaS in de publieke cloud bestaat reeds geruime tijd en is zelfs een tijd lang synoniem geweest voor cloud[3]. Het is de technologie die van de cloud zelf zo'n hype heeft gemaakt. Pas vrij recent zijn ook heel wat bedrijven op de kar van 'private IaaS' (IaaS in een private cloud dus) gesprongen. Kenmerkend hieraan is de grote hype rond producten zoals OpenStack. Private IaaS-projecten zijn echter altijd erg uitdagend, en dit om meerdere redenen: zo is het b.v. quasi onmogelijk om qua prijs competitief[4] te zijn met de "public cloud"-

[2] http://aws.amazon.com/ec2/

[3] Althans wat de enterprise betreft. Voor eindgebruikers stond cloud eerder synoniem voor opslag in de cloud (à la DropBox)

[4] http://www.networkworld.com/article/2925186/cloud-computing/gartner-amazon-s-cloud-is-10x-bigger-than-its-next-14-competitors-combined.html

spelers; men zal dus moeten differentiëren[5] om een kwalitatief voordeel te behalen (b.v. een focus op data security).

> ***Qua prijs en aanbod is het zeer moeilijk concurreren met de public cloud. Kleinere spelers moeten meerwaarde bieden op andere vlakken om competitief te zijn.***

Waar IaaS overeenkomt met de machine waarop men applicaties zal installeren, komt *Platform as a Service* eerder overeen met de middleware. Dit is de infrastructuur die de meeste applicaties nodig hebben om goed te kunnen werken, zeker op enterprise-niveau. De meest voorkomende vorm van PaaS is *Application PaaS* (aPaaS), dat zich focust op het ondersteunen en het bouwen van applicaties.

Kenmerkend aan PaaS is dat onderliggende resources (servers, opslagruimte, netwerkconnectiviteit) gepoold kunnen worden en gebruikt worden om middelen met een hoger niveau van abstractie aan te bieden. Het gaat dan bijvoorbeeld om kant-en-klare database connecties, security tools, applicatieservers of zelfs ondersteuning voor levenscyclusbeheer. Hierdoor hoeft de gebruiker (een softwareontwikkelaar) deze onderliggende resources niet meer zelf te beheren. Bovendien krijgt hij ondersteuning bij het schalen van het verbruik van zijn applicatie.

PaaS-platformen vangen een aantal taken op die bij een software-ontwikkelingsproject telkens opnieuw moeten gebeuren en soms vertraging opleveren. Deze zijn onder meer het opzetten van servers (inclusief OS), het installeren en configureren van de vereiste applicatie- en webservers, databases en andere middleware, en vaak ook het opzetten van de tooling rond de applicatielevenscyclus, zoals b.v. versiecontrole. Daarnaast biedt PaaS vaak mechanismen om het schalen van een toepassing te ondersteunen en om op te meten hoeveel van de onderliggende middelen een applicatie verbruikt.

Platform as a Service is tot dusver een beetje het kleine broertje van de cloud stack gebleven, mede doordat het concept pas iets later is gerijpt dan de beide andere. Voor bedrijven die intensief aan softwareontwikkeling doen, biedt het nochtans aanzienlijke voordelen.

Bij Smals Onderzoek deden we reeds uitgebreid onderzoek naar aPaaS-technologie, zo beschikken we over een research nota[6] en verscheidene blogs[7].

---

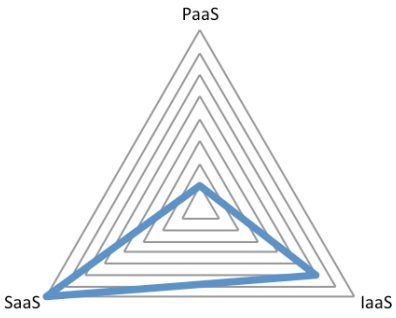
[5] Six Reasons Private Clouds Fail, and How to Succeed, Gartner, Oktober 2014

[6] Research Note 31: Application platform as a Service, maart 2014, https://www.smalsresearch.be/publications/document/?docid=100

[7] https://www.smalsresearch.be/tag/paas/

1.3.2. Software as a Service (SaaS)

Tegenwoordig kennen we Software as a Service dus vooral dankzij de alomtegenwoordige cloud-hype, maar dit toeleveringsmodel voor software bestond eigenlijk al veel eerder. Sterker nog: SaaS heeft, binnen de cloud-markt, nog steeds het grootste marktaandeel van de drie toeleveringsmodellen.



Figuur 3: Cloud-marktverdeling volgens toeleveringsmodel

Dit komt mede doordat SaaS de bovenste laag inneemt van de cloud stack, en meteen ook de meest gevarieerde. SaaS kan namelijk zowat eender welke applicatie betreffen, zolang ze maar *as a Service* wordt aangeboden. Een voorbeeld van SaaS is Google Apps for Work[8]: Voor een bepaalde prijs kunnen bedrijven zich op deze dienst inschrijven om een eigen en geïsoleerde versie te krijgen van populaire Google diensten zoals e-mail, Docs en Drive.

Cloud-spelers die SaaS aanbieden, hebben daardoor een groot competitief voordeel tegenover andere cloud-providers. Dankzij het aanbieden van kant-en-klare applicaties kunnen ze zich namelijk richten tot allerlei soorten klanten. Een IaaS- en/of PaaS-aanbod daarentegen, is meestal slechts interessant voor IT-bedrijven of IT-afdelingen binnen andere bedrijven.

> ***Een sterk SaaS-aanbod opent de markt naar de volledige business van het doelpubliek, en niet enkel naar de IT-departmenten.***

Software as a Service kunnen we definiëren als software die beheerd, maar niet noodzakelijk gebruikt wordt door een leverancier en die, gebruikmakende van internettechnologie, via multi-tenancy wordt aangeboden. Onderliggend vormt de software doorgaans een uniform, of toch op zijn minst een uniform beheerbaar en onderhouden, geheel terwijl deze naar buiten toe voor elke tenant apart lijkt te werken.

SaaS is dus software die volledig onder beheer blijft van zijn leverancier. Deze leverancier, ook provider genoemd, zal de applicatie typisch voorzien op basis van één enkele ‘code base’ (het geheel van broncode en andere configuratie en hulpmiddelen), die echter in een 1-N model wordt gebruikt door alle klanten, gefactureerd volgens één of andere formule volgens het werkelijke verbruik (pay as you go).

Wanneer de software bovendien schaalbaar is, dan kunnen we SaaS ook beschouwen als een cloud-technologie. Praktisch zal dit uiteraard quasi altijd het geval zijn: een niet-schaalbaar product bouwen is weliswaar gemakkelijker en goedkoper, maar wanneer het verbruik stijgt en men dit wil volgen wordt het een pak duurder om het alsnog op schaal aan te bieden. De ROI van een schaalbaar SaaS-product zal op termijn dus groter zijn. Wil men bovendien een goede SLA (Service Level Agreement) kunnen garanderen, dan is het gebruik van cloud-technologie aangewezen.

[8] https://www.google.be/intx/nl_be/work/apps/business/

1.4. SaaS Enablement

Voor een onderneming die een private cloud wil opzetten voor intern gebruik en/of een aantal van haar klanten, zijn de volgende twee zaken belangrijk:

1) Het cloud-aanbod moet een kwalitatieve meerwaarde bieden tegenover de grote “public cloud”-spelers, want concurreren op prijs is haast onmogelijk.
2) De private cloud moet minstens dezelfde mate van gebruikerservaring bieden als de public cloud. Dit wil vooral zeggen dat er een aanbod aan kant-en-klare diensten moet zijn waaruit de klant kan kiezen en die hij of zij via self-service kan inschakelen. Daar staat weliswaar tegenover dat er geen diensten op maat worden gebouwd!

Een voor de hand liggende meerwaarde is natuurlijk het bewaren van de vertrouwelijkheid van de gegevens bij een goed te vertrouwen partij: het eigen datacenter. Vraag is of dit de komende jaren nog een voldoende argument blijft om de grotere kost te kunnen verantwoorden.

Een sterkere meerwaarde vinden we in integratiemogelijkheden en compatibiliteit met een reeds bestaand non-cloud aanbod aan eigen producten en diensten die reeds worden afgenomen, gecombineerd met het aanbieden van een interessante waaier aan nieuwe cloud-diensten die erg specifiek op het eigen cliënteel gericht zijn en daardoor niet in de public cloud voorkomen. Ten slotte kunnen ook bestaande geleverde producten en diensten een meer *cloudy* karakter krijgen.

Bij verschillende van deze zaken zal SaaS een rol kunnen spelen. Het aanbieden van Software as a Service, naast een IaaS- en PaaS-aanbod, is namelijk een enorme troef naar de klanten toe. Het opent het aanbod naar de gehele business van de klant, daar waar de lager op de cloud stack gelegen toeleveringsmodellen zich vooral richten op de IT-departementen.

SaaS-ontwikkeling verschilt echter op vele vlakken van traditionele ontwikkeling omdat SaaS-toepassingen niet gericht zijn op één mogelijk gebruik, maar op verschillende klanten of tenants (multiple tenancy) en dus ook intrinsiek configureerbaar, schaalbaar en beschikbaar moeten zijn. Het nadeel hiervan is tot op heden dat er veel inspanning nodig is om aan deze sterke niet-functionele vereisten te kunnen voldoen. Bovendien is het niet wenselijk om de vele software die reeds is ontwikkeld, volledig te gaan re-engineeren om deze als SaaS te kunnen ontplooien.

Hiervoor willen we dus SaaS Enablement kunnen inzetten. Het doel van deze SaaS Enablement moet zijn om bestaande software met een minimum aan effort aan te passen om deze ook als een dienst te kunnen aanbieden via selfservice, naast het aanbieden van de software onder zijn traditionele vorm. Op die manier kunnen we het aanbod van de private cloud snel en efficiënt verrijken en vervolledigen.

We beseffen dat dit een ”brown field”-operatie is: het feit dat het om bestaande, niet aan de cloud aangepaste, software gaat, kan voor bijkomende problemen zorgen die we niet zouden hebben wanneer we nieuwe software rechtstreeks op de cloud ontwikkelen (een “green field”-operatie).

> ***SaaS Enablement is het opwaarderen van een applicatie zodat het mogelijk wordt om deze in een cloud-context als SaaS aan te bieden.***

In deze nota zullen we onderzoeken hoe SaaS Enablement kan gebeuren voor open source software. Hiermee is de lat in principe iets hoger gelegd dan wanneer we ons enkel zouden toespitsen op software onder eigen controle: we moeten namelijk rekening houden met het feit dat het bestaande softwarepakket buiten onze controle om nog zal evolueren tijdens en na de initiële “SaaS Enablement”-procedure.

1.5. Alternatieven voor SaaS Enablement

Het verrijken van een cloud-aanbod met een veelheid aan aangeboden kant-en-klare applicaties, kan ook op andere manieren. Zo bestaan er al langer oplossingen voor het snel gereedmaken van virtuele servers, die we in een eerste subsectie zullen bespreken. Daarna zoomen we in op Containers, een technologie die reeds op korte termijn de cloud op haar grondvesten heeft doen daveren.

1.5.1. Kant-en-Klare Servers

Indien men een bestaand IaaS-aanbod heeft, en dus virtuele machines aanbiedt via een private cloud, kan men opteren voor het aanbieden van een hele resem aan virtuele machines waarop reeds nuttige software is voorgeïnstalleerd. Verschillende voorbeelden hiervan zijn te vinden in de publieke cloud.

Bitnami[9] is een bedrijf dat images van virtuele machines gratis ter beschikking stelt op zijn website, met ondersteuning voor meer dan 100 populaire webtoepassingen, zoals WordPress en Drupal. Daarnaast biedt het bedrijf ook installatiebestanden aan om op een reeds bestaande machine dezelfde pakketten te installeren, met inbegrip van alle onderliggende middleware. Deze functionaliteit is beschikbaar voor zowel Windows, Linux, als Mac. Een verdere mogelijkheid, die binnenkort beschikbaar wordt, is om de pakketten eveneens via Docker Containers te gaan verdelen (zie volgende subsectie).

Via het aanbod ‘Bitnami Cloud’ laat Bitnami tevens toe om rechtstreeks zijn images naar iemands public cloud account te sturen en dus de gepaste machine met een minimum aan interventie aan te maken en op te starten. Specifiek gaat het over de public cloud van Amazon, Google en Microsoft.

Een concurrent van Bitnami is ***Turnkey***[10]. Ook dit bedrijf biedt images van virtuele machines aan voor een groot aantal populaire toepassingen. Er zijn een aantal verschillen met Bitnami: Turnkey biedt b.v. enkel zijn pakketten aan voor Linux. Daarbij is het wel zo dat Turnkey de standaardmechanismes van software-installatie op machines volgt, terwijl Bitnami een aangepaste manier gebruikt. Hierdoor zijn de virtuele machines van Turnkey achteraf

[9] https://bitnami.com/

[10] http://www.turnkeylinux.org/

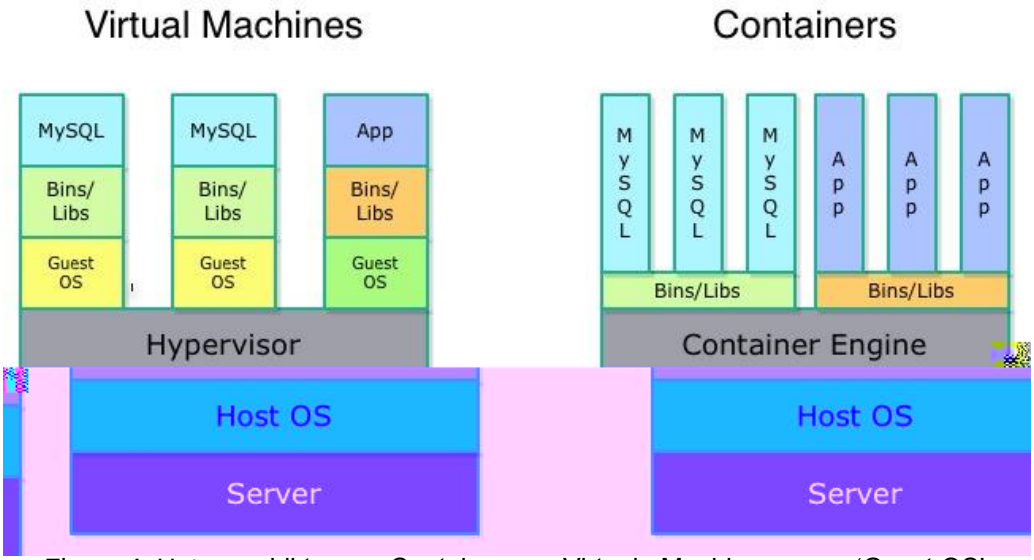
gemakkelijker op de standaardmanier van Linux te onderhouden (via de package managers). Om een major update te doen van een Bitnami-pakket (met inbegrip van de middleware), raadt men daarentegen meestal aan om een nieuwe machine op te zetten en de data van de oude naar de nieuwe te migreren. Kleinere updates kan men typisch uitvoeren via de webinterface van de desbetreffende toepassing.

Bij Turnkey is er ook een iets grotere focus op modulariteit. Zo zou men b.v. bij een WordPress-installatie kunnen opteren om de onderliggende MySQL database op een aparte Turnkey-machine te plaatsen. Bij Bitnami heeft men deze mogelijkheid niet. Uiteraard zorgt dit bij Turnkey voor een grotere complexiteit, waardoor Bitnami een groter gebruiksgemak heeft. Ook Turnkey biedt ondersteuning voor Docker Containers.

Samengevat zou men dus aan IaaS-klanten eveneens zo'n groot aanbod van verschillende virtuele machines kunnen aanbieden. Dezelfde problemen als degene die we verder in de nota zullen zien voor "SaaS Enabled"-applicaties, spelen echter ook hier een rol: onderhoud, updates en integratie. Bovendien heeft deze manier van werken nog een bijkomend nadeel: zolang de applicaties niet te intensief gebruikt worden, passen deze netjes binnen een virtuele machine, maar wanneer er moet worden geschaald, zijn ze hier niet meer voor aangepast.

1.5.2. Containers

Container-technologie is een van die dingen die al een hele tijd bestaan zonder dat er een haan naar kraait, maar dan plots fenomenaal doorbreken als hype. Dankzij Docker[11] staan containers op de roadmap, niet alleen voor Bitnami en Turnkey, maar ook voor vele andere IT-bedrijven.



Figuur 4: Het verschil tussen Containers en Virtuele Machines: geen 'Guest OS' meer nodig

Containers kan men beschouwen als een soort "light weight" virtuele machines, maar het zijn eigenlijk geïsoleerde omgevingen die men binnen een

[11] https://www.docker.com/

machine inneemt (zonder dat daarvoor een volledige virtuele machine nodig is). Via Linux beveiligingstechnologie houdt men ze perfect gescheiden van de rest van de machine en van andere containers. In een container kan men dan alle programmatuur en configuratie stoppen om een applicatie te laten werken. Zie ook Figuur 4 voor het verschil met virtuele machines.

Men kan hier twee richtingen mee uit: zoals bij de virtuele machines kan men in principe een hele software stack in zo'n container stoppen en aldus een werkend en netjes geïsoleerd pakket bekomen dat men kan laten uitvoeren op eender welke machine die de technologie ondersteunt ('build once, run anywhere'). Maar waar containers echt in uitblinken, is in modulariteit: elk deel van een systeem kan men in een andere container stoppen en deze dan met elkaar laten communiceren (b.v. het webgedeelte en de database apart).

Over deze technologie kan nog heel veel gezegd worden, maar om applicaties aan te bieden aan klanten, is het voldoende te weten dat men ook van deze containers er een heel aantal kan verzamelen met allerlei nuttige applicaties erin, net zoals men met de virtuele machines kan. Docker biedt zelfs al zo'n plaats aan waar vele mensen vrijwillig aan bijdragen (de Docker *Hub*). Het volstaat dan om op de private cloud een Containerondersteunend platform aan te bieden en eventueel een GUI om het opstarten van een standaardcontainer nog wat te vergemakkelijken.

Net zoals bij de kant-en-klare servers, vergt het gebruik van containers onderhoud en werk aan updates en integratie. Het schaalbaarheidsprobleem kan daarentegen deels worden verholpen door slim gebruik te maken van de modularisatie.

Container-technologie is voorlopig nog in volle evolutie, maar kan op korte termijn een valabele manier worden om aan SaaS Enablement bij te dragen. Hierover meer in sectie 3.4.

2. VEREISTEN VOOR SAAS ENABLEMENT

In dit hoofdstuk lijsten we op wat er allemaal nodig is om een software-applicatie te kunnen SaaS-Enablen. We beginnen met algemene zaken die nodig zijn in de context van Software as a Service. Daarna zoomen we kort in op *Enterprise* Enablement, iets wat niet strikt nodig is voor SaaS Enablement, maar wel een belangrijke factor zal zijn voor de kwaliteit van het uiteindelijk aangeboden product. In de derde sectie definiëren we dan wat een minimale SaaS Enablement, zonder andere bijkomstigheden, moet inhouden. We ronden het hoofdstuk af met een woordje over de rol van Platform as a Service binnen SaaS Enablement.

2.1. Algemene Vereisten voor SaaS Software

SaaS-software moet minimum voldoen aan de volgende vereisten van cloud-technologie: **selfservice, multi-tenancy, pay as you go**, elastische schaal-baarheid en het gebruik van internet technologie. De laatste voorwaarde is typisch reeds voldaan indien het over een webtoepassing gaat, en dit zijn dan ook de enige soort toepassingen die we zullen beschouwen (niet-webtoepassingen vormen tegenwoordig slechts een kleine minderheid).

De voorwaarde van elastische schaalbaarheid zullen we nader bespreken in de sectie rond PaaS. Rest dus nog te bekijken hoe de 3 eerste voorwaarden (selfservice, multi-tenancy, pay as you go) kunnen worden vervuld.

In de eerste subsectie bekijken we de basisvoorwaarden die bij elke SaaS Enablement minstens zullen moeten worden vervuld. In de tweede beschouwen we de extra benodigdheden die er zijn wanneer men een uitgebreid aanbod van verschillende SaaS-diensten wenst aan te bieden.

Bij SaaS Enablement gaan we op zoek naar de beste manier om sofware aan de belangrijkste SaaS-eigenschappen te laten voldoen:

Selfservice, multi-tenancy, pay as you go

2.1.1. Basisvereisten

Indien we vertrekken van een ‘gewone’ webtoepassing, zal er typisch heel wat aan moeten worden toegevoegd en veranderd om deze als SaaS te kunnen aanbieden. Onder een ‘gewone’ webtoepassing verstaan we, ruim genomen: een toepassing die werkt via de browser en die het inloggen van gebruikers ondersteunt. Deze laatste voorwaarde is er omdat we typisch in een afgeschermde omgeving willen kunnen werken, waarin meerdere gebruikers gelijkaardige zaken kunnen doen, en uitwisselen van informatie tussen de gebruikers enkel op een gecontroleerde manier mag gebeuren.

Laten we dan oplijsten wat in principe nodig is:

a) Selfservice en Multi-tenancy

De tenant moet zich via een beheersmodule een eigen instantie van de dienst kunnen aanmaken. Let erop dat dit niet hetzelfde is als een eenvoudige zelfregistratie van een eindgebruiker op een reeds bestaande instantie van de dienst. Het beheer van de eindgebruikers zal per tenant gebeuren.

b) Pay as you go en Monitoring

De tenant moet via de beheersmodule kunnen zien hoe en in welke mate de instantie wordt gebruikt, en hoeveel dit kost. Hij of zij kan ook de facturatie raadplegen.

c) Configuratie van de toepassing

Net zoals een gewone webtoepassing allerlei configuratie-instellingen aanbiedt, zal de SaaS-toepassing deze opties ook aanbieden. Deze instellingen zullen echter enkel van toepassing zijn op de welbepaalde instantie van de tenant, en dus geen invloed hebben op het SaaS-systeem als geheel. Bovendien kunnen er verschillen zijn tussen wat de toepassing standaard aan opties zou aanbieden, en als SaaS. Zo kunnen er zaken verborgen worden voor de tenant, omdat enkel de provider deze kan beheren (b.v. instellingen die met eventuele backend-databases te maken hebben). Het is evengoed mogelijk dat er extra opties worden voorzien (b.v. de mogelijkheid om een logo aan te passen). Om deze reden zal men typisch niet klakkeloos de beheersmodule van de oorspronkelijke webtoepassing kunnen overnemen.

2.1.2. Extra Vereisten i.v.m. Multi-SaaS offerings

Een leverancier of ontwikkelaar van private of community SaaS-diensten zal zich typisch niet beperken tot één enkele SaaS. Wanneer men echter een uitgebreid SaaS-aanbod voor ogen heeft, zal men nog extra zaken in acht moeten nemen tegenover het geval van één enkel SaaS-product. Er zullen extra ondersteunende diensten moeten worden opgezet waarmee de verschillende SaaS-producten integreren.

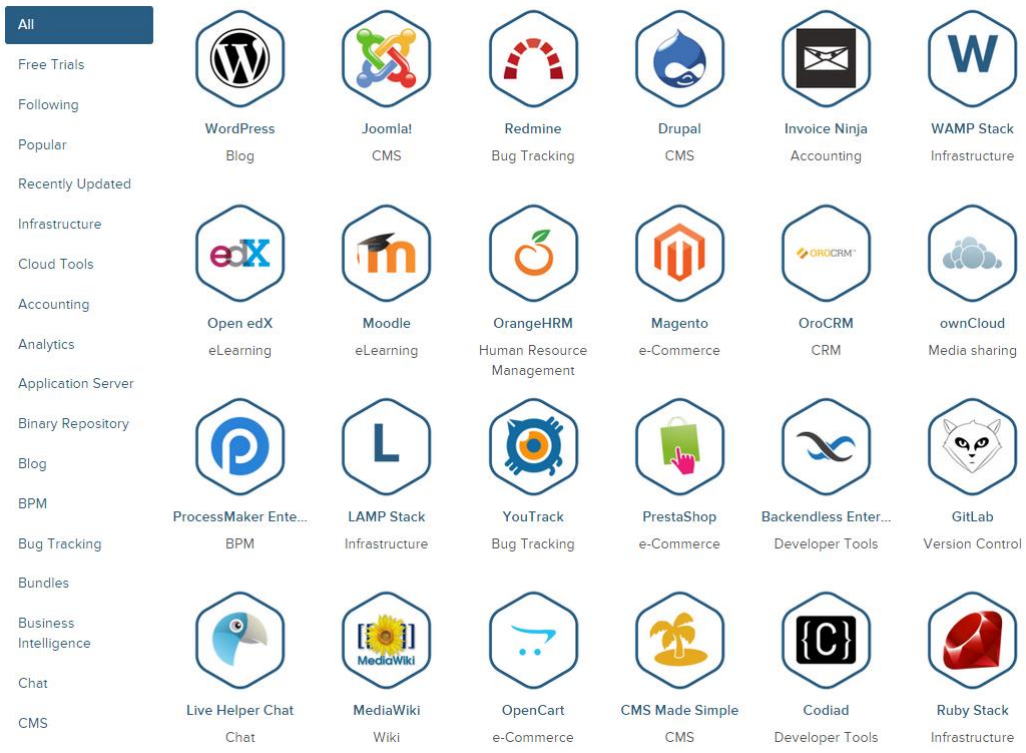
a) Catalogus

Wanneer men een multi-SaaS-aanbod voor ogen heeft, maakt men best gebruik van een catalogus, waar de verschillende tenants de mogelijkheid hebben te kiezen uit de verschillende producten en er zich voor in te schrijven.

Vanuit de catalogus moet het voor de tenant gemakkelijk zijn om zich in te schrijven en om een portefeuille aan SaaS-producten te beheren. Elk softwareproduct dat als SaaS wordt aangeboden, moet vanuit deze catalogus te benaderen zijn voor instantiatie en configuratie.

b) Geïntegreerd User Management

Voor wie zich als tenant inschrijft op meerdere SaaS-toepassingen, is het quasi onontbeerlijk dat het gebruikersbeheer over de verschillende toepassingen heen goed geïntegreerd is. De specifieke rollen/permissies van gebruikers binnen een bepaalde toepassing kunnen eventueel apart worden



Figuur 5: Een voorbeeld van een cataloguspagina: de ‘stacks’ van Bitnami, te vinden op de site

beheerd, maar het beheer van gebruikers en groepen zelf moet voor alle toepassingen één zijn.

c) Klantenbeheer

Facturatie in een multi-SaaS omgeving gaat verder dan het meteren en factureren van één toepassing. De tenants krijgen allicht graag een overzicht van hun gehele verbruik over verschillende toepassingen heen, en verwachten een single point of contact.

2.2. Enterprise Enablement

In deze sectie geven we een aantal bijkomende vereisten waaraan SaaS-software zal moeten voldoen om effectief bruikbaar en van goede kwaliteit te zijn op Enterpriseniveau. Volgens de definitie betekent dit eigenlijk gewoon 'technologie ontwikkeld op maat van bedrijven i.p.v. individuele gebruikers'. Concreet staat 'Enterprise ready' voor matuur, kwalitatief, goed integreerbaar, 'full-blown' (uitgebreide en volledige functionaliteit), etc.

Samengevat zijn in een enterprisecontext twee zaken extra belangrijk: een hoge mate van integratie, en een hoge graad van kwaliteit. Deze kunnen op de volgende manieren worden bereikt:

- **Kwaliteit van de originele toepassing**. Deze dient uiteraard voldoende hoog te zijn, anders zal er sowieso niet kunnen worden voldaan aan de kwaliteitseisen. We komen hier op terug in sectie 3.1.
- **Elastische Schaalbaarheid op elk niveau van de toepassing**. Wanneer we efficiënt van de onderliggende resources van het systeem gebruik willen maken, dan moeten deze zoveel mogelijk gezamenlijk gebruikt worden door de SaaS-toepassing. Dit wil bijvoorbeeld zeggen dat alle instanties van alle tenants gebruikmaken van dezelfde pool van webservers, of zelfs eventueel van dezelfde database, waarbij er dan enkel een logische, en geen fysieke scheiding is tussen de door de verschillende tenants gebruikte onderdelen. Op die manier vermijden we dat we deze zaken voor elke instantie opnieuw moeten instantiëren. Dat is één van de basisprincipes van Cloud Computing. Om dit te kunnen, moet een toepassing echter op alle niveaus apart schaalbaar gemaakt zijn. Voor een typische toepassing, bestaande uit drie lagen (presentatielaag, voorzien door browsers/webservers; applicatielaag, voorzien door webservers/applicatieservers en datalaag, voorzien door databases), wil dit dan bijvoorbeeld zeggen dat de cluster webservers kan groeien wanneer het webverkeer toeneemt, zonder daarbij automatisch de andere lagen mee te schalen, tenzij ook het gebruik van deze lagen mee zou toenemen (wat niet altijd het geval is, en zeker niet in dezelfde mate). De schaalbaarheid moet ook elastisch zijn: dit wil zeggen dat we het aantal beschikbare resources ook kunnen verminderen wanneer de belasting terug zou afnemen.
- De kwaliteit op zich is ook afhankelijk van **integratiemogelijkheden op een meer technisch niveau**. In een enterprisecontext kunnen er o.a. reeds veralgemeende diensten bestaan voor monitoring en voor het beheer van logs. Hoe goed de applicatie hiermee kan integreren is mede bepalend voor de kwaliteit van de uiteindelijk opgezette dienst.

Aan de andere kant willen we natuurlijk wel een zo groot mogelijk gedeeld gebruik van de onderliggende computing resources bereiken; er zal dus een zeker evenwicht moeten worden gevonden tussen zoveel mogelijk multi-tenancy en resource sharing bereiken enerzijds, en de effort zo klein mogelijk houden anderzijds.

2.3.2. Selfservice

Om van SaaS te kunnen spreken, moeten klanten zelf de controle hebben over het in- en uitschakelen van de software. Er moet een interface worden voorzien waarin ze hun gebruik kunnen aan- en uitschakelen, hun andere instellingen betreffende de toepassing kunnen beheren, en een overzicht kunnen krijgen van verbruik en facturatie.

Dit betekent dus dat we op zijn minst een soort catalogus zullen moeten aanbieden. Gelukkig kunnen we de effort beperken tot een eenmalige ontwikkeling van zo'n catalogus. Het volstaat daarna dat elke nieuwe SaaS Enablement ermee compatibel is.

Daarnaast zullen we ons moeten beperken bij het configureerbaar maken van de toepassing. De meeste software biedt aan administrator accounts al heel wat mogelijkheden om via de eigen interface heel wat zaken te veranderen; deze zullen we uiteraard automatisch ondersteunen indien we de toepassing als SaaS uitrollen. Sommige applicaties hebben echter ook nog extra configuratiemogelijkheden via het wijzigen van onderliggende bestanden etc. Deze 'offline' mogelijkheden zullen we enkel kunnen aanbieden aan tenants die het aandurven hun handen vuil te maken achter de schermen van de toepassing: het zou te veel ad hoc implementatiewerk vergen om hiervoor aparte gebruikersinterfaces voor het web te moeten ontwikkelen.

2.3.3. Elastische Schaalbaarheid

Ook voor de elasticiteit van de toepassing zullen we een zekere mate van pragmatisme moeten aanvaarden. We kunnen hier het onderscheid maken tussen twee niveaus van schaalbaarheid.

Op het niveau van de als SaaS aangeboden toepassing, en het onderliggende platform, moeten we uiteraard voorzien in een hoge mate van schaalbaarheid: we moeten vele tenants tegelijkertijd kunnen ondersteunen, en dit voor meerdere SaaS-toepassingen.

Anderzijds is het op het niveau van één tenant, binnen één toepassing niet altijd even hard nodig om extreme schaalbaarheid te voorzien. Hier kunnen we een afwachtende houding aannemen en eventueel slechts stappen ondernemen om bepaalde tenants voor bepaalde toepassingen van een hogere schaalbaarheid te voorzien, wanneer ze aangeven deze nodig te hebben.

Vaak is het ook zo dat de webtoepassingen op te splitsen zullen zijn in meerdere tiers (b.v. web tier & data tier). Ook hier is het mogelijk dat niet alle tiers even schaalbaar zullen moeten worden gemaakt.

2.3.4. Enterprise Enablement

Ondanks het grote belang van Enterprise Enablement voor de toepassingen, kunnen we hier niet te veel aandacht aan besteden voor elke software apart. We zullen moeten op zoek gaan naar manieren waarop we de vereiste integraties tot op zeker niveau kunnen ondersteunen ter hoogte van een onderliggend platform. Zoals we in de volgende sectie zullen zien, kan een PaaS-platform daarvoor oplossingen bieden.

Verder zal ook de selectie van kandidaat-software voor SaaS Enablement nauwgezet moeten gebeuren: veel mogelijkheden tot integratie, en de moeite die hiervoor moet worden gedaan, zal afhangen van de opties daartoe voorzien in de bronsoftware.

In hoofdstuk 3, bij het bespreken van onze strategie voor SaaS Enablement, zullen we Enterprise Enablement verder buiten beschouwing laten.

2.4. De Rol van aPaaS

Platform as a Service, meer bepaald *Application* platform as a Service, zal een belangrijke rol spelen voor SaaS Enablement van bestaande software. Aangezien een SaaS Enablement in zekere zin een ontwikkelings- en uitroleffort zal betreffen, en aPaaS-platformen zich toespitsen op ondersteuning voor deze zaken, zullen deze een niet te onderschatten deel van het werk voor ons kunnen verlichten. Aldus vormen ze een noodzakelijke vereiste voor SaaS Enablement.

2.4.1. Schaalbaarheid en Elasticiteit

Een aPaaS-systeem ondersteunt typisch de schaalbaarheid van de erop ontwikkelde toepassingen, en zal zelf ook erg elastisch kunnen schalen, aangezien erop uitgerolde applicaties kunnen komen en gaan. APaaS-platformen nemen dus al een groot deel van de effort om efficiënt gebruik te maken van de onderliggende Computing resources voor hun rekening, of het nu om een onderliggend IaaS-platform gaat, of om rechtstreeks gebruikte hardware.

Op die manier krijgen we dus de schaalbaarheid en elasticiteit op het niveau van het platform cadeau: we zullen zoveel SaaS Enablements kunnen doen als de schaalbaarheid van de aPaaS toelaat, en ook binnen één SaaS Enablement zullen we, zoals te zien zal zijn in volgend hoofdstuk, een manier kunnen vinden om zoveel versies van de software voor aparte tenants uit te rollen en eventueel terug te verwijderen, als nodig.

Het aanbieden van applicaties bovenop een aPaaS laat ons toe een groot deel van de elastische schaalbaarheid inherent in dit patform aanwezig, te hergebuiken

2.4.2. Integratie met Ondersteunende Diensten

Een onderliggend platform voor applicaties kan een aantal zaken standaard voorzien waarvan elke applicatie gebruik zal kunnen maken.

Een eerste, relatief eenvoudig, voorbeeld is logging functionaliteit. De meeste applicaties, alsook hun onderliggende middleware, genereren logs. De manieren waarop deze logs worden opgeslagen zijn beperkt (in de meeste gevallen via log files, heel af en toe in een database). Hierdoor kan het platform in de meeste gevallen gemakkelijk alle nuttige logs binnen een systeem vinden en verzamelen.

De meerwaarde in functionaliteit kan dan zijn dat men een handige view biedt op de logging van een applicatie, via b.v. een website. Dit alles kan goed gestandaardiseerd en geautomatiseerd worden door het aPaaS-systeem en aldus krijgt men dus de integratie met een algemeen centraal systeem voor het beheer van logs voor elkaar. Een gelijkaardig voorbeeld kan men uitwerken voor monitoring.

Iets moeilijker ligt het voor de integratie van een Identity en Access Management systeem (IAM), of kortweg het gebruikersbeheer. Dit is namelijk voor vele toepassingen net iets anders geïmplementeerd of geconfigureerd. Nochtans is het vrij belangrijk dat we nieuw gelanceerde toepassingen eenvoudig kunnen koppelen aan bestaande systemen voor gebruikersbeheer.

Applicaties die één of ander standaardmechanisme ondersteunen, zoals b.v. LDAP, hebben hier een streepje voor, maar dan nog is de manier waarop men zo'n systeem aan een applicatie koppelt niet gestandaardiseerd, dus er zal altijd nog enige ad hoc configuratie nodig zijn.

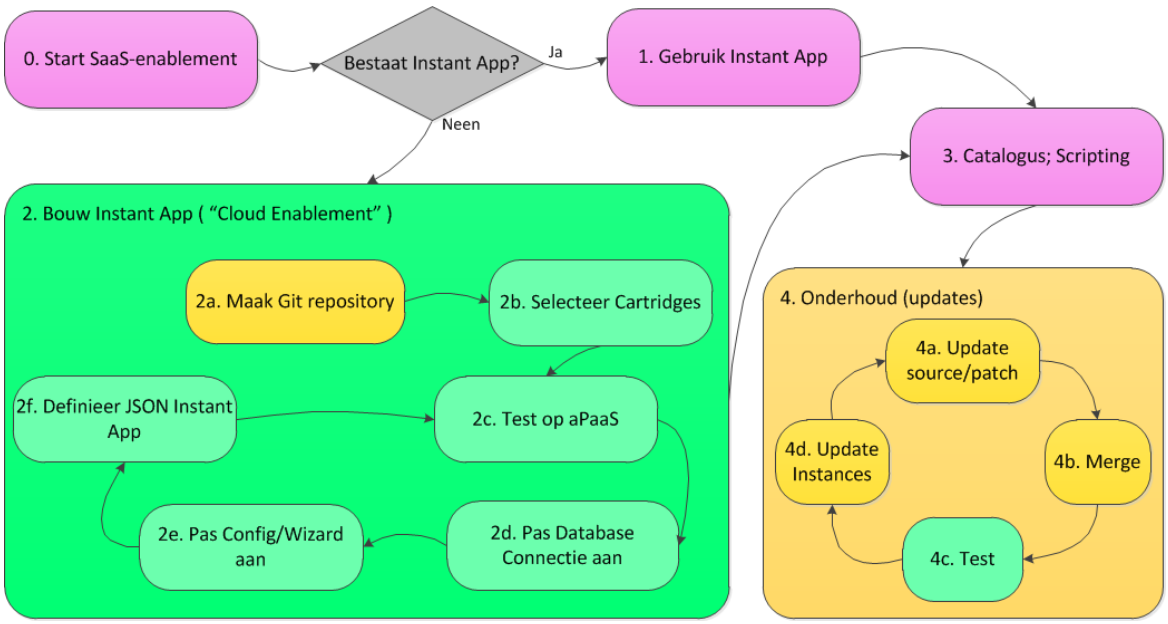
Hoe het aPaaS hier kan helpen is door sowieso al de nodige verbindingen te voorzien die de applicatie nodig zal hebben om met een bestaand IAM-systeem te kunnen connecteren. Hoe automatisch deze connectie dan effectief geconfigureerd kan worden, zal afhangen van de mogelijkheden van de brontoepassing.

In de loop van de studie maakten we gebruik van het Red Hat OpenShift Online aPaaS-platform[14], waarnaar we in het komend hoofdstuk een aantal keer zullen verwijzen. Dit platform bespraken we reeds in de eerder vermelde studie aPaaS.

3. EEN MINIMALE SAAS ENABLEMENT STRATEGIE

In dit deel bespreken we een strategie voor SaaS Enablement die we in de loop van de studie ontwikkelden en testten. Ze werd uitgedacht met als belangrijkste opzet een lage kost, wat dus neerkomt op een zo slank mogelijke procedure, met zo weinig mogelijk manuele interventie.

[14] https://www.openshift.com/products/online



Figuur 6: Hoog-niveau stappendiagram van de "SaaS Enablement"-strategie *Instance as a Service*

In Figuur 6 zien we een schematische voorstelling van de "SaaS Enablement"-procedure, met daarin 3 belangrijke onderdelen, die we zullen bespreken in aparte secties van dit hoofdstuk. Deze drie fasen hebben in de figuur aparte kleuren (roos, geel en groen); twee stappen binnen een fase kregen de kleur van een andere fase, om het belang ervan voor deze andere fase te benadrukken.

- Het eerste en overkoepelende onderdeel dat we zullen bespreken is *Instance as a Service*: de achterliggende strategie die we gebruiken voor Saas Enablement (roze in de figuur).
- Een daaropvolgend deel behandelt Cloud Enablement (groen): dit is het compatibel maken van de bronsoftware met een PaaS-platform, hetgeen ons een aantal belangrijke voordelen oplevert voor weinig moeite.
- Vervolgens gaan we dieper in op het verhaal van updates (geel): zowel de werkende SaaS-dienst als de originele bronsoftware kunnen namelijk wijzigingen ondergaan en we moeten deze op kunnen nemen.

We beginnen het hoofdstuk echter met de voorwaarden waaraan de software op basis van dewelke we een SaaS-dienst willen ontwikkelen (= de *bronsoftware*), moet voldoen. Dit wordt onmiddellijk gevolgd door een sectie die de infrastructuur beschrijft die we slechts één keer hoeven opzetten (i.t.t. het werk dat we bij elke SaaS Enablement zullen moeten herhalen).

Na de beschrijving van de strategie gaan we ook nog dieper in op de case studies die tijdens de studie werden gedaan, en in welke mate onze methode multi-tenancy ondersteunt. We eindigen dit hoofdstuk met een reflectie over hoe we meerwaarde zouden kunnen creëren bovenop deze eenvoudige "SaaS Enablement"-strategie.

3.1. Vereisten voor de bronsoftware

Het zal niet verbazen dat de vereisten voor de originele softwarepakketten focussen op kwaliteit en een goede functionaliteit en integratiemogelijkheden. We sommen ze in detail op:

- De belangrijkste voorwaarde voor bronsoftware is uiteraard dat dit pakket ook een nuttige functionaliteit biedt die populair is bij potentiële klanten.
- Ook zeer belangrijk is dat het om een goed onderhouden en kwalitatief sterk softwarepakket gaat. Dit zal de effort om te SaaS-Enablen een stuk kleiner maken.
- Daarnaast is ook de update policy belangrijk: hoe compatibel de nieuwe versies zijn met de vorige, en hoe pijnloos men een update of eventuele migratie kan uitvoeren, zullen doorslaggevend zijn bij de hoeveelheid onderhoud die men aan het SaaS-Enablede product zal hebben. We komen hier uitgebreid op terug in sectie 3.5.
- Ten slotte zijn ook de integratiemogelijkheden van groot belang, met voorop de ondersteuning van een standaardtechnologie voor autorisatie, zoals b.v. LDAP.

Goede bronsoftware is nuttig, matuur, goed onderhouden en gemakkelijk integreerbaar

3.2. Extra Infrastructuur in functie van een Multi-SaaS-context

Buiten het werk dat aan elke toepassing apart zal moeten gebeuren, zijn er ook een aantal concrete zaken te voorzien in een Multi-SaaS-context, die we slechts éénmaal moeten opzetten:

- **De catalogusapplicatie**. Deze zal, naast haar voor de hand liggende functionaliteit, gebruik moeten maken van de api van het onderliggende aPaaS-platform om instanties van toepassingen te lanceren voor de tenants. Van belang is dus ook dat de nodige aandacht geschonken wordt aan Capacity Management.
- **De facturatie-backend**. Deze zal, naast uiteraard integratie met de catalogusapplicatie voor het beheer door de tenants, ook moeten integreren met de verschillende lagen aan monitoring voorzien in de onderliggende aPaaS en de erop gedeployde applicaties, en dit omwille van de noodzaak om aan metering te kunnen doen in de context van pay-per-use. Een andere mogelijkheid is om enkel de facturatie van de onderliggende aPaaS te gebruiken en de gebruikers dus rechtstreeks voor het verbruik van de onderliggende aPaaS resources te factureren.
- **Logging- en Monitoringinfrastructuur**. Ook hiervoor kan men een aantal zaken klaarzetten op niveau van het platform, al zal er voor elke applicatie apart ook nog wat integratiewerk aan te pas komen. De grafische interfaces die zullen toelaten deze bronnen te raadplegen zijn een goed voorbeeld.

3.3. InaaS: Instance as a Service

De algemene strategie achter SaaS Enablement is dus Instance as a Service, wat erop neerkomt dat we een versie klaarzetten van de bronsoftware die gemakkelijk herhaaldelijk is te instantiëren door de tenants. Bovendien moeten updates aan deze versie ook gemakkelijk kunnen worden doorgevoerd naar de instanties die reeds in productie zijn. Waarom we specifiek voor een strategie met aparte instanties kiezen (i.t.t. een strategie met slechts één multi-tenant instantie voor iedereen), bespreken we verderop in de subsectie.

Zoals reeds vermeld, maken we in de studie gebruik van OpenShift als aPaaS-platform. Dit platform maakt gebruik van zogenaamde 'Instant Apps', waarmee snel volledige software stacks op een schaalbare manier kunnen worden ontplooid. Het zijn deze Instant Apps die we zullen creëren, gebruiken en up-to-date houden binnen Instance as a Service.

Zoals in Figuur 6 is te zien, is het mogelijk dat er reeds een Instant App bestaat voor de software die we willen SaaS-Enablen. In dit geval volstaat het om deze te gebruiken. Soms kan het zijn dat bestaande Instant Apps niet alle kenmerken vertonen die we wenselijk achten (sommige ondersteunen b.v. het schalen van de applicatie niet). In dat geval moeten we een afweging maken of we een eigen variant zullen gaan ondersteunen of niet. Het spreekt voor zich dat het gebruik van een reeds bestaande versie een stuk goedkoper zal zijn.

Voorbeelden van bestaande Instant Apps zijn WordPress, Drupal of de JBoss BPM Suite.

> ***Ook binnen onze strategie Instance as a Service, kunnen we vaak gebruikmaken van reeds bestaande en vaak gratis aangeboden 'Instant Apps'***

Indien geen Instant App bestaat, dan moeten we deze zelf voorzien. Instantieerbaar maken dient voor elke te SaaS-Enablen toepassing opnieuw te gebeuren. Gelukkig zal dit echter weinig meer inhouden dan de volgende stap, Cloud Enablen.

Waarom InaaS?

Om de voorwaarde van multi-tenancy te vervullen, zullen we er bij SaaS Enablement voor zorgen dat de toepassing door meerdere tenants tegelijk kan worden gebruikt terwijl deze geïsoleerd zijn van elkaar, en dat terwijl we toch een zekere mate van gedeeld gebruik van resources trachten te behouden.

In theorie kan dit op drie manieren:

- Ofwel is de toepassing voldoende gemakkelijk aanpasbaar, zodat we ervoor kunnen zorgen dat ze op zichzelf voldoende ondersteuning gaat bieden voor meerdere van elkaar geïsoleerde tenants.
- Indien de eerste manier geen optie is, kan het zijn dat we ervoor kunnen zorgen dat sommige tiers van de toepassing gemakkelijk te isoleren zijn van andere, en dat we een aantal van deze tiers multitenant maken, terwijl we andere delen zullen repliceren per tenant. Indien de te repliceren tiers daarbovenop nog toestandsloos zijn, bekomen we een goed schaalbaar en

nagenoeg perfect multitenant geheel. Mogelijk kunnen we b.v. de database die de toepassing gebruikt, delen onder alle instanties, door voor elke tenant een andere username te gebruiken in de database, of door op een andere manier een logisch onderscheid tussen tenants te definiëren. Of op applicatief vlak kunnen we eventueel, afhankelijk van wie er is ingelogd, andere configuratie-instellingen raadplegen (waarbij we de ingelogde user mappen op de tenant). In de front-end kunnen we eventueel gebruikmaken van één cluster front-endmodules per tenant, waarbij elke cluster zijn eigen configuratiebestanden kan krijgen.

- Ofwel, ten slotte, is de toepassing erg moeilijk aanpasbaar aan het bestaan van meerdere tenants en is het bovendien ook moeilijk om één of meerdere tiers af te splitsen en multitenant te maken. In dit geval zullen we verplicht zijn om de hele stack van de toepassing volledig opnieuw uit te rollen telkens wanneer een tenant om een instantiëring ervan vraagt, en om deze instanties op het niveau van de onderliggende aPaaS logisch van elkaar gescheiden te houden. Het niveau van multi-tenancy zal hierdoor lager zijn, maar de TCO (Total Cost of Ownership) kan toch nog lager uitvallen omdat deze methode heel wat ontwikkelingswerk kan besparen tegenover de andere methodes.

In de praktijk zullen we bij een minimale “SaaS Enablement”-strategie meestal kiezen voor de **derde manier** van werken: voor elk soort toepassing zal dit het minste moeite kosten. Het verlies aan efficiëntie in het gebruik van onderliggende resources (die op die manier minder gedeeld worden tussen tenants) is meestal te verwaarlozen, doordat het voor een groot stuk teniet wordt gedaan door het aPaaS-platform dat we zullen inzetten (zie 3.4).

> ***Instance as a Service is de meest economische manier om te SaaS-Enablen, dankzij handig gebruik van de elasticiteit van het onderliggende aPaaS-platform***

3.4. Cloud Enablement

We vermeldden reeds een aantal keer dat het SaaS-Enablen van een toepassing bovenop een aPaaS-platform zal gebeuren, vanwege de vele inherente voordelen aan die aanpak.

Het is echter zo dat niet alle softwarepakketten ertoe zijn ontworpen om snel en gemakkelijk op dit platform te kunnen worden geplaatst. M.a.w. niet alle software wordt aangeboden als Instant App. Welnu, wanneer we bestaande software in die mate aanpassen dat ze wel degelijk op een aPaaS-platform kan worden uitgerold, spreken we van *Cloud Enablement*.

In Figuur 6 komt Cloud Enablement overeen met de groene fase. De naam Cloud Enablement kan dan wel vrij uitdagend klinken, gelukkig is het echter zo dat de meeste toepassingen vrij eenvoudig kunnen worden aangepast om op een aPaaS te draaien, zeker indien ze gemakkelijk configureerbaar zijn. Meestal komt het er vooral op neer dat men op zijn minst de configuratie van de databaseconnectie automatisch moet kunnen aanpassen, gezien de aPaaS-platformen typisch de database dynamisch zullen toewijzen aan de

applicatie. Daarnaast kunnen er nog andere kleine configuratie-instellingen zijn.

Meestal zal dit dus voor een developer bekend met de programmeertaal van de broncode van de toepassing een bescheiden hoeveelheid werk zijn. Spijtig genoeg zal dit werk, hoewel er dus wel een rode draad in is te zien, ad hoc blijven per te SaaS-Enablen toepassing; aangezien deze zaken toch altijd net iets anders geïmplementeerd zijn. Het zal dus niet automatisch kunnen.

Indien de applicatie uit meerdere tiers (b.v. web – app – db) bestaat, die makkelijk afzonderlijk uit te rollen zijn, kunnen we eventueel opteren om elk van deze tiers afzonderlijk te Cloud-Enablen. Dit zal het apart schalen van de tiers, en dus de flexibiliteit van de gehele toepassing, ten goede komen.

> ***Cloud Enablement noemen we het aanpassen van een toepassing zodat ze op het gewenste aPaaS platform kan werken***

Zo klein mogelijke apart schaalbare componenten zijn altijd efficiënter, maar indien het niet mogelijk is de tiers apart uit te rollen, is het nog geen ramp: doordat het aPaaS-systeem reeds vrij efficiënt gebruikmaakt van de onderliggende hardware resources, krijgen we op die manier ook al een vrij goed gedeeld resourcegebruik over de verschillende instanties van alle tenants heen.

Tegen het einde van dit jaar zal OpenShift 3 Containertechnologie ondersteunen[15]. Dit zal als bijkomend voordeel hebben dat we applicaties nog makkelijker moduleerbaar zullen kunnen maken bij het Cloud-Enablen.

Ten slotte is het belangrijk dat Cloud Enablement goed wordt opgevolgd wat betreft de broncode. Zoals we zullen zien in de volgende sectie, zal er, met elke nieuwe release van de originele toepassing, opnieuw een ‘merge’ van de originele code en de gedane aanpassingen moeten gebeuren, en zullen een aantal integratietesten moeten worden uitgevoerd.

3.5. Omgaan met Updates

Eens een bestaand opensourcepakket SaaS Enabled werd, is dit niet het einde van ons werk. Goede opensourcesoftware zal regelmatig worden geüpdatet, en het is van belang om onze SaaS Enablede instanties mede up-to-date te houden. Dit doen we in de laatste fase in figuur 6 (geel); deze fase blijft uiteraard lopen gedurende de volledige levensduur van de applicatie.

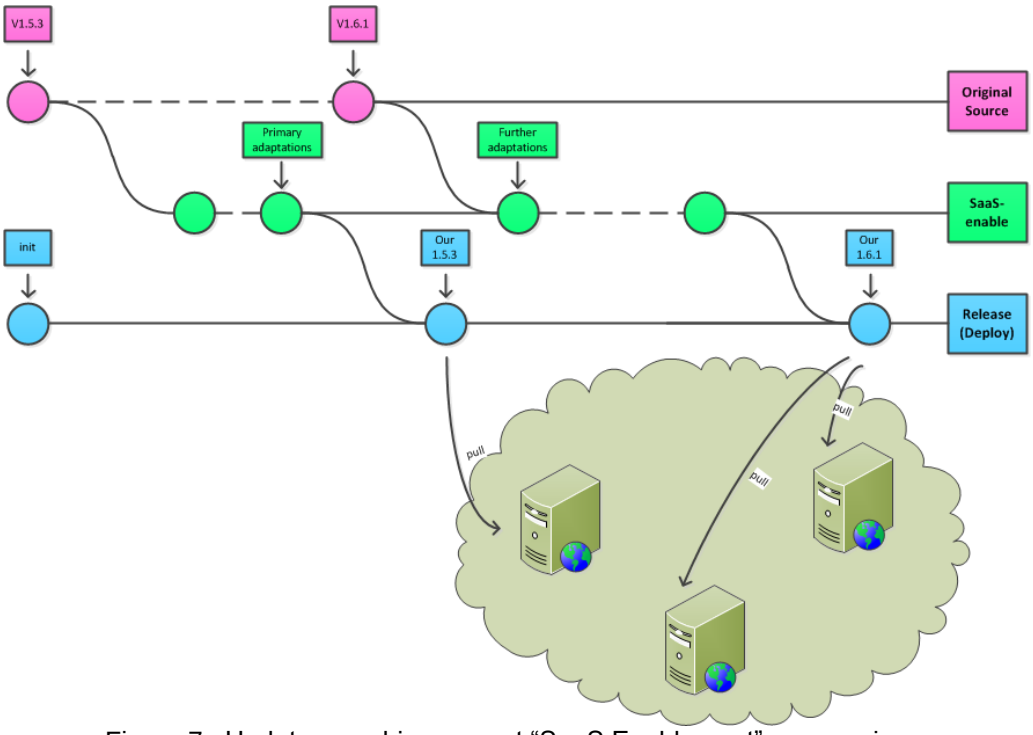
Updates van een softwarepakket zullen typisch de volgende complicaties voor ons meebrengen:

- Zo kan het zijn dat bepaalde updates in de broncode van het pakket conflicteren met wijzigingen die we hebben aangebracht bij het SaaS-Enablen
- Verder is het ook mogelijk dat we reeds ontplooide instanties hebben van een oudere versie wanneer we een update uitvoeren

[15] https://blog.openshift.com/the-future-of-openshift-and-docker-containers/

- Bovendien kan het zijn dat in de ontplooide instanties wijzigingen ontstaan tijdens het gebruik door de tenants (b.v. in de www of config folder van een eenvoudige webapplicatie)

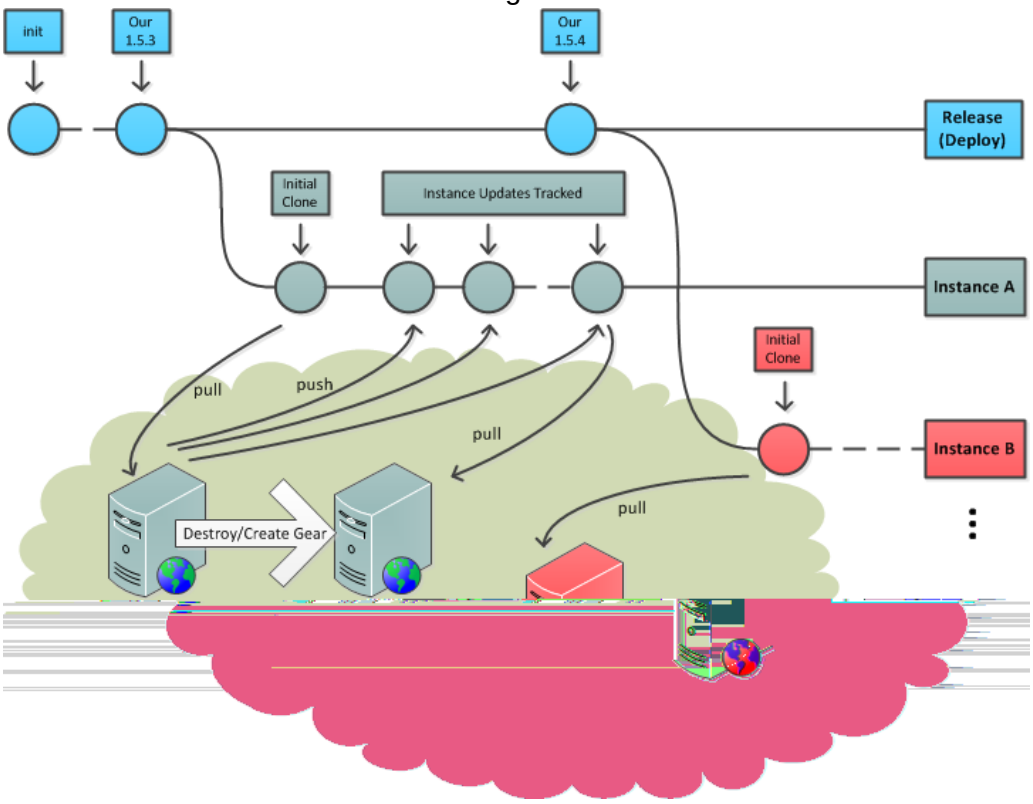
Deze problemen zullen we moeten aanpakken d.m.v. een goede updatestrategie. Versiecontrole van de broncode speelt daarin een cruciale rol. We zullen een git repository onderhouden waarin we de externe broncode importeren in een ***Original Source branch***. Onze aanpassingen om te SaaS-Enablen zullen we uitvoeren op een ***SaaS-Enable branch*** en zullen we af en toe op een ***Release branch*** overnemen. Deze Release branch zal de bron zijn voor nieuwe instantiaties van de software door tenants.



Figuur 7 : Updates combineren met “SaaS Enablement”-aanpassingen

Wanneer er nu broncode-updates komen, wil dat zeggen dat we evolutie zien op de Original Source branch. Deze kunnen we dan (her-)mergen met de SaaS Enable branch en eventueel de ontstane conflicten terug manueel herstellen. Via diffs en een aantal eenvoudige testen zal dit meestal vrij gemakkelijk verlopen, al kan het af en toe zo zijn, als net dat mechanisme in de originele broncode waarop we ingrijpen om te Saas-Enablen b.v. volledig wordt omgegooid of vervangen, dat we quasi opnieuw zullen moeten Saas-Enablen. Er moet uiteraard altijd getest worden; dit kan door een instantie te bouwen op basis van de SaaS Enable branch op een testserver. Eens deze testen met zekerheid lukken, kunnen we het nieuwe resultaat opnieuw overnemen op de Release branch. Dit proces wordt geïllustreerd in Figuur 7. Op deze manier kunnen we, althans wat nieuwe instantiaties betreft, de software goed onderhouden.

We voeren vervolgens ook in dat alle bestaande instantiaties hun eigen branch krijgen vanaf het moment dat ze voor de eerste keer gedeployd worden. Deze branch zal gekoppeld worden aan de branch die op het aPaaS-platform lokaal actief is voor de instantie (dit kunnen er meerdere zijn indien de applicatie meerdere web-tier-nodes inneemt, maar deze worden door het aPaaS-platform synchroon gehouden). Het is namelijk zo dat sommige applicaties d.m.v. runtime-gebeurtenissen (zoals configuraties en uploads) wijzigingen kunnen aanbrengen in deze folders. Er bestaan mogelijkheden om deze wijzigingen te committen in een git repository[16] en om ze dan te pushen naar de ***Instance branch*** op de centrale repository. Dit principe kunnen we zien op figuur 8. Een bijkomend voordeel van deze manier van werken is dat, wanneer een node om één of andere reden door het platform vernietigd en achteraf terug gecreëerd zou worden, deze eveneens de wijzigingen die hij reeds onderging kan terugkrijgen uit de git repository. Uiteraard zijn we er ons van bewust dat dit eigenlijk geen algemeen goed ontwerpsprincipe is: software die geschreven is om als SaaS te functioneren zou dit soort wijzigingen niet op een node zelf mogen doen, maar enkel in een aan de toepassing gekoppelde database; op die manier krijg je een “stateless” runtime, die je gemakkelijker kan laten schalen door het platform. Spijtig genoeg zijn vele opensourceproducten niet met deze methodologie indachtig geschreven en hebben we dus deze work-around nodig.



Figuur 8: Wijzigingen in bestaande instanties volgen

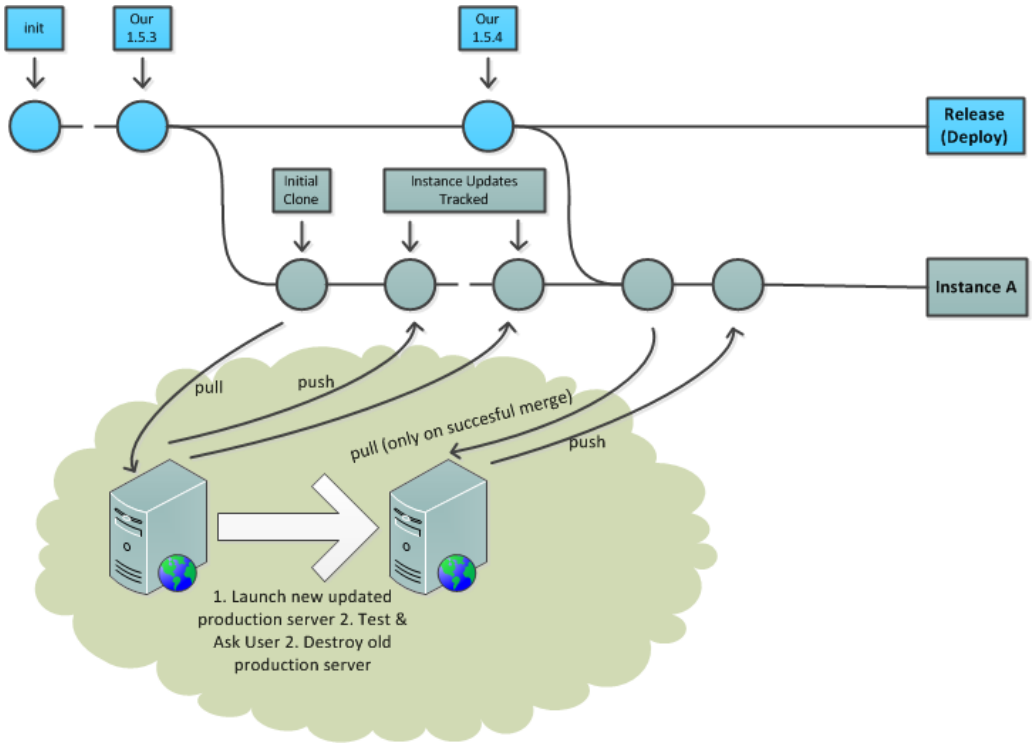
We kunnen op dit punt dus zowel met wijzigingen in de broncode omgaan als met wijzigingen door gebruik; zij het apart. Er rest ons dus nog deze beide soorten wijzigingen met elkaar te kunnen verenigen. Dit zal gebeuren in de

[16] https://forums.openshift.com/one-newbie-question (kan gescript worden)

Instance branch van de toepassing. Indien we dit doen, kunnen ook bestaande instanties worden geüpdatet naar een volgende versie van de originele software, met behoud van hun eventuele eigen wijzigingen. Om dit te kunnen doen, is weliswaar een goede procedure nodig, om conflicten en crashes te vermijden (we zien dit in Figuur 9, doch slechts de belangrijkste stappen). Deze versie van het updateproces is geldig wanneer de leverancier van de opensourcesoftware geen specifieke update of patch scripts levert, die in de plaats van het in-place updaten van de code kunnen worden uitgevoerd.

Stapsgewijs gaat dit als volgt:

1. (automatisch) Merge de nieuwe versie van de Release branch op de Instance branch
2. (door de platformbeheerders) Los eventuele merge-conflicten op
3. (automatisch) Lanceer een nieuwe instantie op basis van de merge.
4. Deze zaken worden getoond op de dashboards van zowel de gebruiker als de platformbeheerder
5. (door de platformbeheerders of door de gebruiker) Kijk de nieuwe instantie na.
6. (door de gebruiker) Duid aan op dashboard dat update mag gebeuren
7. (automatisch) Alle gebruik wordt naar de nieuwe instantie gerouteerd en de oude instantie wordt offline gehaald

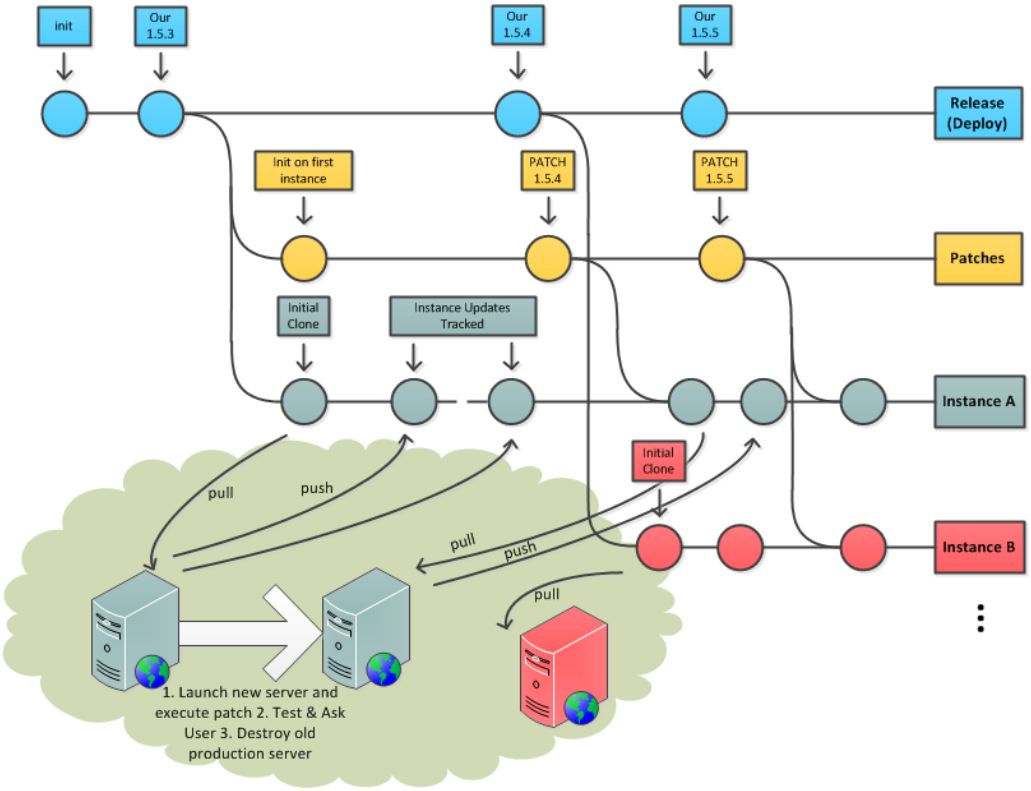


Figuur 9: Nieuwe versie combineren met bestaande wijzigingen in instantie

Sommige softwarepakketten zullen update of patch scripts aanleveren om in de plaats te gebruiken van een eenvoudige update van de code. Het gebruik hiervan is meestal aangeraden. In zo'n geval kunnen we echter geen eenvoudige merge doen met de Release branch, omdat deze met de standaard, doorheen de tijd wijzigende broncode werkt. In zo'n geval kan de Release branch enkel worden gebruikt voor het lanceren van nieuwe instanties. Wat we dan wel zullen doen, is een speciale branch voorzien

waarop we telkens de vorige patch verwijderen en de volgende gereed zetten, waarmee dan alle instanties kunnen worden geüpdatet. Dit kan je zien in Figuur 10: na de initiële instantiatie, zal de instance branch verdere updates verkrijgen van de patches branch i.p.v. de release branch. De procedure voor het updaten verloopt verder zoals in de vorige figuur. (In Figuur 10 is enkel de eerste update van instance A volledig uitgewerkt.)

Uiteraard is het de bedoeling dat alle instances zo goed mogelijk aansluiten bij de laatste update (anders zouden we meerdere patch en release branches moeten onderhouden voor de verschillende versies). Klanten die een update niet laten uitvoeren, zullen achteraf geconfronteerd worden met een hogere kost voor een custom upgrade, omdat er dan ook extra manueel werk aan te pas zal komen en het testen sowieso meer tijd in beslag zal nemen, wegens het grotere versieverschil.



Figuur 10: Patches integreren in bestaande instanties

3.6. Case Studies

In de loop van de studie gingen we aan de slag met twee opensourceapplicaties die nog geen bestaande 'Instant App' op het OpenShift-platform aanbieden. Daarnaast bouwden we een minimale versie van een catalogus van toepassingen.

3.6.1. GLPI

GLPI (Gestionnaire Libre de Parc Informatique)[17] is een webapplicatie voor het beheer van informaticaparken, inclusief workflows, takenbeheer en ticketing (ITIL). Het pakket is geschreven in PHP.

We deden een Cloud Enablement van GLPI, door in de broncode een aantal aanpassingen door te voeren in de wizard voor de eerste ingebruikname van het pakket; dit om ervoor te zorgen dat de database connectie automatisch zou verlopen. Dit bleek voldoende om het pakket te laten werken op het aPaaS-platform. Een ervaren PHP-programmeur zou hiervoor allicht minder dan een dag nodig hebben.

Vervolgens voerden we één code-update door volgens de eerste manier (mergen van de Release en Instance branches). Doordat we alles manueel deden, was dit ook een effort van ongeveer een dag, maar een aantal zaken zijn hierbij te automatiseren (b.v. het volgen van de originele source code via een branch, alsook het volgen van de instances, zodat er uiteindelijk enkel nog het mergen en testen overblijft).

3.6.2. SugarCRM

SugarCRM[18] is een bekend Customer Relationship Management (CRM) pakket met een gratis opensourcebasisversie. Net als GLPI, is het het geschreven in PHP.

Hier lag de Cloud Enablement iets moeilijker, omdat de database-initialisatie te sterk verweven was met andere codes (vermoedelijk zou een ervaren PHP-programmeur hier wel mee overweg hebben gekund). We kozen voor een work-around: uiteindelijk hebben we het creëren en intitialiseren van de database in een opstartscript afgezonderd. Dit script wordt uitgevoerd nog voor het lanceren van de webapplicatie.

Door de complexiteit van SugarCRM en de nood aan een work-around duurde deze case study wat langer dan de vorige, maar we maken ons wederom sterk dat iemand met een grondige PHP-kennis dit op minder dan een week tot een goed einde zou kunnen brengen.

De updates van SugarCRM werken op de tweede manier: via een patch file. Deze file kon gelukkig wel de updates aan een reeds in productie zijnde database doorvoeren zonder verdere work-arounds nodig te hebben. Dankzij de kwaliteit van de patch kostte dit uiteindelijk zelfs minder tijd dan we nodig hadden voor de andere methode bij de eerste case study.

3.6.3. Catalogus

Via een democatalogusapplicatie (niet meer dan een lijst van applicaties en een knopje om een nieuwe instantie te lanceren), testten we of we de functionaliteit van het OpenShift-platform voor het creëren van applicaties konden oproepen vanuit een klantvriendelijke omgeving. Hierbij werden de technische details van het aPaaS-platform verborgen.

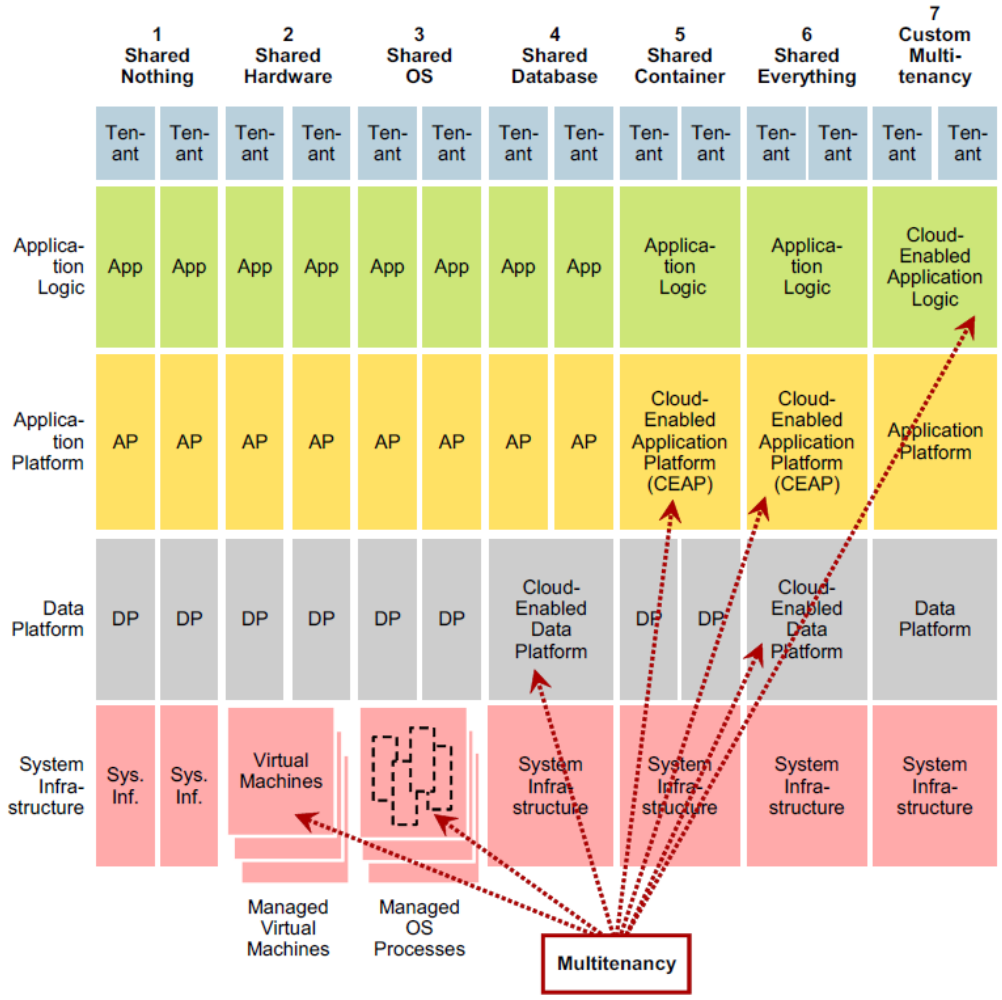
[17] http://www.glpi-project.org/

[18] http://www.sugarcrm.com/

Dit lukt vrij goed doordat OpenShift een vrij volledige RESTful api aanbiedt, waarbij wordt gebruikgemaakt van JSON-objecten om met het platform te interageren. Bovendien was het mogelijk om het starten van een 'instant app' te simuleren, door in één commando alle nodige informatie om een applicatie te initialiseren door te geven. Een instant app is eigenlijk niet meer dan een JSON-object dat die informatie bevat (althans in de huidige versie van OpenShift, die nog niet via Docker werkt).

3.7. Multi-Tenancy

We bespraken reeds een aantal keer de noodzaak voor een zekere multi-tenancy in onze oplossing, terwijl we deze dienden te balanceren met het minimaliseren van de effort. In deze sectie bespreken we welk niveau van multi-tenancy we uiteindelijk kunnen behalen met onze aanpak.



Figuur 11: Gartner Multitenancy Referentiemodel

Gartner[19] biedt een model aan om multi-tenancy te meten, gebaseerd op een aantal niveaus, gaande van volledige isolatie van de tenants op hardwareniveau (*Shared Nothing*; dit is dus eigenlijk nog geen multi-tenancy),

[19] Gartner Reference Model for Elasticity and Multitenancy; 22 juni 2012

tot *Shared Everything,* waarbij alles, van dat databases tot zelfs de applicatielogica, gedeeld gebruikt wordt.

Gezien via onze methode telkens volledig nieuwe instanties worden gebouwd van een applicatie, zou men verwachten dat het niveau van multi-tenancy vrij laag valt. Maar dankzij het gebruik van het onderliggende aPaaS platform, hebben we niet alleen een gedeeld gebruik van onderliggende hardware, maar ook nog eens het gebruik van een gedeeld Cloud-Enabled Application Platform (CEAP)!

Een CEAP vinden we pas terug vanaf niveau 5 in het model (waarbij databases, zoals bij onze methodologie, nog apart staan per tenant). We hebben echter het probleem dat, vanaf dit niveau ook de applicatielogica gedeeld wordt, hetgeen bij ons niet het geval is.

Daarom voeren we een niveau 5bis in, een niveau gelijkend op niveau 5, maar met aparte applicatielogica per tenant. We kunnen dus stellen dat onze methode dit nieuwe niveau 5bis haalt op de multi-tenancy schaal.

3.8. Meerwaarde Toevoegen

De cloud-markt is nog niet zo heel oud, maar we zien toch dat een zekere consolidatie is begonnen. Wat standaard cloud-diensten, zoals IaaS en Storage, betreft, zullen kleinere spelers nooit op kunnen tegen kolossen als Amazon, Google en Microsoft. Ze kunnen enkel overleven door extra dienstverlening aan te bieden en zich op bepaalde niches te concentreren.

Ook private clouds wacht, zoals we reeds in de inleiding aangaven, hetzelfde lot. Daarom moeten we dus op zoek naar meerwaarde die in de public cloud niet te vinden is. SaaS Enablement op zich is zeker waardevol, maar nog onvoldoende. Door de sterke opkomst van steeds gebruiksvriendelijkere en goedkopere publieke SaaS-diensten, en nu ook kant-en-klare servers en containers, kunnen onze potentiële gebruikers al snel elders beginnen kijken voor een catalogus aan bruikbare applicaties.

Het komt er dus op neer om niet alleen een goede samenstelling van nuttige applicaties aan te bieden, maar daarbovenop nog iets extra te voorzien. Op dat moment gaan we verder dan SaaS-Enablen, en dus weg van het opzet van deze studie. Dessalniettemin sommen we hier een aantal mogelijkheden op.

- Integratie met goede technische randdiensten, zoals logging en monitoring, en security audit logging.
- Integratie met het “Identity en Access Management”-systeem van de klanten, en/of een goed aanbod van een geïntegreerd IAM.
- Koppeling aan een bestaand aanbod van reeds aan de klant geleverde diensten en custom applicaties, alsook de daarachterliggende gegevensbronnen.
- Koppeling aan bestaande middleware-infrastructuren, zoals b.v. een algemene Message Bus[20].

Daarnaast is het ook van belang, naast kostprijs, een andere zeer belangrijke selectiefactor niet uit het oog te verliezen: gebruiksgemak! Hoe goed een

[20] Een Message Bus is een infrastructuur die onafhankelijke systemen toelaat met elkaar te communiceren via een gemeenschappelijke interface.

bepaalde dienstverlening te gebruiken is, en de kwaliteit van de achterliggende diensten, bepalen naast de kosten het meeste de voorkeur van een klant. En uiteraard dient men ook aandacht te besteden aan andere belangrijke niet-functionele vereisten, zoals een sterke security.

We sluiten af met de volgende metafoor: *de kledingindustrie*. Aanbieders van cloud-diensten, gericht op een specifiek doelpubliek, zijn een beetje als kleermakers die kleren op maat maken. Daarnaast heb je de grote, onpersoonlijke public cloud spelers: deze zijn een beetje als de grote confectieketens die 'one size fits all' aanbieden. Vroeger waren er enkel kleermakers, maar velen van deze zijn weggeconcurreerd door de opkomst van de grote ketens. Sommige hebben zich echter kunnen aanpassen en blijven tot op vandaag rendabel. En dat is dus de uitdaging voor een kleine cloud-speler of private cloud-aanbieder.

4. CONCLUSIE & AANBEVELINGEN

We noteren de belangrijkste conclusies die we uit de tekst kunnen halen en geven een aantal aanbevelingen.

4.1. Conclusie

SaaS Enablement past binnen een grotere strategie van diversificatie van het eigen cloud-aanbod, en dit om competief te kunnen blijven, gegeven de de facto kleinere schaal en de concurrentie van de public cloud.

SaaS Enablement via Instance as a Service kan relatief goedkoop gedaan worden gebruikmakende van de in deze studie ontwikkelde methode. Deze maakt handig gebruik van een onderliggend aPaaS-platform (waardoor SaaS Enablement ≈ Cloud Enablement). Instance as a Service kan echter niet volledig automatisch gebeuren, en is dus niet gratis. Bovendien is SaaS Enablement slechts een stukje van de puzzel: echte meerwaarde zal moeten komen uit verregaande integratie en extra dienstverlening (SaaS Enablement ≠ Enterprise Enablement).

Binnen Saas Enablement zelf, zijn het updates en onderhoud die uiteindelijk het meeste werk zullen vragen, en mogelijk het meeste problemen kunnen geven. Een goede selectie van de bronsoftware kan dit grotendeels voorkomen.

Het grote voordeel van onze methode is dat we schaalbaarheid cadeau krijgen, dankzij het onderliggende aPaaS-platform. Bovendien zorgt ze voor een efficiënter gebruik van resources, zeker tegenover oplossingen die gebruikmaken van één of meerdere virtuele machines per instantie. Daarnaast kan de aanwezigheid van een aPaaS-platform ook nog zorgen voor een gemakkelijker beheer van ondersteunende diensten, zoals logging en monitoring van de applicaties.

Cloud Competitiviteit kan worden verhoogd via SaaS Enablement en via een groot aantal integraties, zoals met ondersteunende diensten, Identity en Access Management, Middleware en bestaande op maat gemaakte software. De focus moet op service liggen, en niet zozeer op prijs.

4.2. Aanbevelingen

Een minimale SaaS Enablement kan relatief goedkoop gebeuren, maar is niet gratis. Daarom raden we af om zomaar willekeurige sofwarepakketten te beginnen SaaS-Enablen en via een catalogus aan te bieden, in de hoop dat tenants ervan gebruik zullen maken. Dit is iets wat kosteneffectief kan zijn op de schaal van grote public cloud-spelers, maar niet in een private cloud voor een beperkt doelpubliek.

De beste strategie zal zijn om af te wachten tot wanneer een klant een bepaalde functionaliteit nodig heeft. Is er op dat moment een goed opensourcealternatief voorhanden, dan kan men gaan SaaS-Enablen, zodat ook andere klanten gemakkelijk van de oplossing gebruik kunnen maken. Dit zorgt voor schaalvoordelen en synergieën.