# Ontwikkelen van een schaalbaar en modulair PaaS voor Big Data oplossingen

Merlijn Sebrechts

Promotoren: prof. dr. ir. Filip De Turck, dr. ir. Tim Wauters
Begeleiders: ir. Thomas Vanhove, dr. Gregory Van Seghbroeck, dr. Marleen Denert

Masterproef ingediend tot het behalen van de academische graad van
Master of Science in de industriële wetenschappen: informatica

Vakgroep Informatietechnologie
Voorzitter: prof. dr. ir. Daniël De Zutter
Faculteit Ingenieurswetenschappen en Architectuur
Academiejaar 2014-2015

UNIVERSITEIT GENT

# Ontwikkelen van een schaalbaar en modulair PaaS voor Big Data oplossingen

Merlijn Sebrechts

Promotoren: prof. dr. ir. Filip De Turck, dr. ir. Tim Wauters
Begeleiders: ir. Thomas Vanhove, dr. Gregory Van Seghbroeck, dr. Marleen Denert

Masterproef ingediend tot het behalen van de academische graad van
Master of Science in de industriële wetenschappen: informatica

Vakgroep Informatietechnologie
Voorzitter: prof. dr. ir. Daniël De Zutter
Faculteit Ingenieurswetenschappen en Architectuur
Academiejaar 2014-2015

UNIVERSITEIT GENT

# Inhoudsopgave

# Toelating tot bruikleen

De auteur geeft de toelating deze scriptie voor consultatie beschikbaar te stellen en delen van de scriptie te kopiëren voor persoonlijk gebruik. Elk ander gebruik valt onder de beperkingen van het auteursrecht, in het bijzonder met betrekking tot de verplichting de bron uitdrukkelijk te vermelden bij het aanhalen van resultaten uit deze scriptie.

The author gives permission to make this master dissertation available for consultation and to copy parts of this master dissertation for personal use. In the case of any other use, the limitations of the copyright have to be respected, in particular with regard to the obligation to state expressly the source when quoting results from this master dissertation.

Merlijn Sebrechts, juni 2015

# Woord vooraf

Een lagereschoolopdracht om een opstel te schrijven over "later" resulteerde in een verhaal over hoe ik zou werken in mijn eigen computerbedrijf. Omstandigheden zorgden er echter voor dat ik als 18-jarige jongen met een diploma *Biotechnische Wetenschappen* en een GIP "Productie van Bio-marsepein" in handen stond. "Wat nu?" was een echt dilemma, want zowat alles interesseerde mij.

Uiteindelijk heb ik dan toch beslist voor de bachelor *Electronica-ICT* met afstudeerrichting ICT. Dit bleek al snel een uitstekende keuze te zijn. Ik had mijn passie gevonden. Nu zijn we drie succesvolle bachelorjaren, een schakel- en een masterjaar later. Ik sta op het punt om af te studeren als *Industrieel Ingenieur Informatica*, en deze masterproef is de afsluiter. Deze masterproef was ontzettend leerrijk. Echte problemen oplossen verplicht je om enorm veel bij te leren. Na een jaar zwoegen voor deze masterproef moet ik toch wel zeggen dat ik *het ontwikkelen van een schaalbaar en modulair PaaS voor Big Data oplossingen* veel interessanter en uitdagender vind dan Bio-marsepein.

Op mijn eentje zou het mij niet gelukt zijn. Daarom wil ik iedereen die ik onderweg ben tegengekomen bedanken. Met het risico om mensen te vergeten: Eerst en vooral bedank ik Universiteit Gent, de onderzoeksgroep Internet Based Communication Networks and Services (IBCN), iMinds, en de promotoren prof. dr. ir. Filip De Turck en dr. ir. Tim Wauters voor de unieke kans die deze masterproef is. Voor hun uitgebreide feedback en begeleiding bedank ik de begeleiders dr. Gregory Van Seghbroeck, ir. Thomas Vanhove en dr. Marleen Denert. Bedankt aan de online communities zoals Askubuntu en Stackoverflow om mij te helpen bij problemen, en bedankt aan de Juju ontwikkelaars voor het snel oplossen van bugs. Als laatste bedank ik mijn vrienden, familie en klasgenoten voor het nalezen van en de steun tijdens deze masterproef.

Merlijn Sebrechts, juni 2015

# Abstract

Deze masterproef stelt een architectuur van een schaalbaar en modulair Big Data Platform as a Service (PaaS) voor. Dit platform geeft gebruikers de mogelijkheid om met beperkte kennis van systeembeheer een platform op te zetten om Big Data applicaties voor te ontwikkelen. Gebruikers kunnen kiezen voor een standaardopstelling gebaseerd op de Lambda Architectuur of om zelf een opstelling te bouwen door middel van basisblokken. Deze modulaire functionaliteit wordt verkregen door gebruik van de Service Orchestration Tool Juju. De voorgestelde architectuur wordt geïmplementeerd door Tengu, een implementatie van de Lambda architectuur, om te zetten in een modulair en schaalbaar PaaS. Verder wordt onderzocht wat de beste manier is om een Juju Charm te implementeren.

# Development of a scalable and modular PaaS for Big Data solutions

Merlijn Sebrechts

Supervisor(s): prof. dr. ir. Filip De Turck, dr. ir. Tim Wauters

*Abstract*— **This article summarizes the master thesis "Development of a scalable and modular PaaS for Big Data Solutions". This master thesis proposes an architecture for such a platform using the Service Orchestration tool Juju. The proposed architecture is then implemented by converting the Tengu platform, an implementation of the Lambda infrastructure made by IBCN, into a scalable and modular PaaS.**

*Keywords*— **Big Data, Configuration Management, Service Orchestration, Juju, Chef, PaaS**

## I. Introduction

REALTIME Big Data analytics is a field where a lot of research and innovation is happening. The sudden popularity resulted in an explosion of Big Data related tools. These tools have the ability to work together in complex ways to solve varying types of problems. Because of this, there are three problems that have to be solved before the development of a Big Data application can begin. The first problem is finding an architecture for the platform to develop upon. The second problem is finding the right software to implement that architecture. Lastly, there is the problem of actually setting up and configuring the platform. Solving these problems is time-consuming and knowledge-intensive.

Nathan Marz proposes a solution to the first problem: the Lambda architecture, a best-practice general-purpose architecture for Big Data solutions. However, there is no easy way to check this claim, since there doesn't exist a platform to setup and test different Big Data solutions.

It is the goal of this masters thesis to develop such a scalable and modular Big Data PaaS. It must be modular, so it is easy for a user to test different solutions. It must be scalable, so it can handle real-world Big Data workloads. This platform must enable developers and researchers to develop and implement different Big Data solutions in an easy way. This platform must be modular, so it is arbitrarily easy to test different solutions.

## II. Functional analysis

Since the platform needs to be rebuild constantly, the value is in the code to setup the platform, and not in the deployed platform itself. Therefore, the Infrastructure as Code paradigm needs to be used. The created code needs to be of high quality. It needs to be maintainable and extendable so that it is easy to evolve the platform to meet future needs. In order to achieve this, the code has to be modular and use abstraction. These two features need to be present in the language and framework used to create the platform.

It must be arbitrarily easy to test different solutions on the platform. To enable this, the platform must present basic components, a single service, that can be combined, using relations, to create a Big Data solution. It must be easy for the user to replace a service with one that has a similar function. Because of this requirement, conventional policy-based configuration management tools, as first described first by Mark Burgess, will not suffice. This requirement needs a service orchestration tool that manages services and their relations. A Service Orchestration operates at a layer above conventional configuration management[1]. Because of this, there is still a need for a policy-based configuration management tool to manage the individual servers.

Each service must be scalable to handle Big Data workloads. Services also need to do simple disaster recovery themselves. When a service crashes, it needs to be restarted automatically.

The platform must support multi-tenancy and roles. Every customer needs to have an instance that is not affected by instances from other customers. Each customer must be able to assign different responsibilities to different users of the platform.

Finally, it is very important that the platform has long term viability. The platform must be able to adapt to the changing Big Data needs, and the platform must be built on technologies that will stay relevant and supported for a long time.

## III. Proposed Architecture

The core of the architecture is Juju, an Open Source Service Orchestration Tool made by Canonical. Juju handles the orchestration of the services and their relations and presents an easy to use interface to the user: the juju-gui. Services are encapsulated into Charms that have relations with other Charms. A Charm is a bundle of installation and configuration scripts. Each configuration script subscribes to a hook that triggered by Juju. The first hook that is triggered by Juju is the *install* hook. This hook installs and configures the standalone service. When the service gets a relation with another service, *relation-changed* hooks get called, and the Charm configures that relation. When a service needs to be scaled over multiple servers, the Charm handles the installation and configuration of the complete cluster. In this architecture, the configuration scripts are Chef scripts, so each time a hook is called, a Chef script is run. Chef runs in *chef-zero* mode. This is done so a Charm has no outside dependencies. Juju has an interface to request nodes from different IaaS providers like Ubuntu MAAS, Digitalocean or Amazon EC2. The *manual provider* enables you to use existing infrastructure by giving Juju a static pool of nodes. Juju uses these nodes to deploy the Charms.

Charms enable the user to build a Big Data solution from scratch. Charm bundles enable the user to deploy a default solution based on the best practice principles of the Lambda archi-

[1] http://askubuntu.com/a/66287/172367

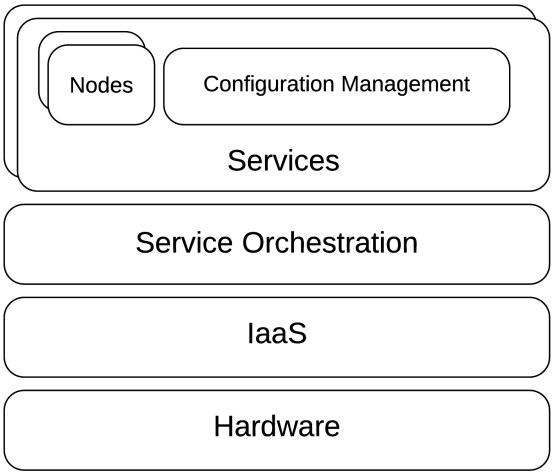



Fig. 1. The architecture of the Scalable and Modular PaaS

tecture. A Juju Bundle defines a list of Charms, relations and configuration parameters. After deployment of the bundle, the user can still change the solution to better fit his needs.

This architecture does not provide a way to implement multi-tenancy. multi-tenancy can only be supported when Juju supports different users with different roles. Although this feature is planned to be implemented in Juju, there is no release date known.

In order to make sure that Juju is a stable platform and a good long-term decision, Canonical was contacted with questions about the future of Juju. From this communication, it became clear that Juju is a good choice. Canonical uses Juju internally for every project, and it is devoted to keep supporting Juju for a very long time. Some of the biggest industry players build their products on top of Juju. Finally, Mark Shuttleworth responded with a statement that he is interested in Juju personally and that it has a bright future under all the scenarios he can influence.

## IV. Implementation

The platform requires communication between JFed, Juju and the virtual wall. This communication is done by the tengu script. The tengu script is written in Python. It installs the required dependencies, requests nodes from JFed and deploys a Juju environment with a Juju GUI on top of these nodes.

Most of the required services were already implemented in Charms. However, not all the Charms implemented all the functionality needed for Tengu. For this thesis, it was decided to modify the Charm Store Charms instead of developing new ones. Using Store Charms has a lot of benefits. Store Charms have already a large community around them that maintain and update the Charms. The community also implements and maintains relations to other Charms. A lot of Store Charms are maintained by the developer of the service that is encapsulated and the Charm store provides a test infrastructure that automatically tests all submitted Charms.

In order to decide how to implement the Charms that were not available in the Charm Store, the Hadoop Charm was implemented in three different ways: Using Bash, Python and a combination of Python and Chef. The use-case concluded the following: Bash is a good language to get to know Juju, but it isn't robust enough, doesn't provide builtin idempotence, is very platform-dependent and the code is hard to maintain. The code from Python Charms is more robust, but it is still very platform dependent, and it also doesn't provide builtin idempotence. The combination of Python and Chef yielded the best result. The code is very robust, it is less platform dependent, and Chef provides builtin idempotence.

The communication between Juju and Chef happens using the Python Services framework and databags. When Juju executes a hook, the Services framework checks if a configuration Change is needed. When it is needed, the configuration values are put into a databag and the Chef Cookbook is executed. The Chef Cookbook uses the databag values to make the necessary configuration changes.

## V. Conclusions

During the thesis, a scalable and modular Big Data PaaS was created. It was clear from the beginning that it would not be possible to create such a platform using conventional policy-based Configuration Management tools. The proposed architecture used Juju, a service orchestration tool, to enable the platform to be built in a modular and highly scalable way. By encapsulating services and their relations in Juju Charms, modularity was achieved. However, conventional policy-based configuration management tools are still useful to manage individual servers. Juju Bundles proved to be a great way to deploy a default Big Data platform based on the Lambda architecture.

Using Juju, all but one requirement was implemented. Because of limitations in Juju, it was not possible to implement multi-tenancy support.

# Verklarende woordenlijst

| | |
|---|---|
| Big Data | Een collectie van data die zo groot is dat ze niet kan verwerkt worden door een conventioneel dataverwerkingssysteem. |
| ESB | Enterprise Service Bus |
| Service | Een geheel van programma's, processen en bestanden die een specifieke taak uitvoeren. |
| Container | Een afgeschermd deel van een besturingssysteem. |
| Node | Een eenheid waar services op kunnen draaien. Dit kan een fysieke computer, een virtuele machine of een container zijn. |
| Multi-tenant | Een methode om meerdere gebruikers dezelfde applicatie te laten gebruiken. Multi-tenant duidt specifiek op het geval waar meerdere gebruikers éénzelfde instantie van een applicatie gebruiken. |
| Peer-to-peer | Een netwerk waarbij nodes gelijkwaardig zijn. Communicatie gebeurt rechtstreeks tussen twee nodes. Elke node kan communicatie starten en stoppen. |
| Platform as a Service (PaaS) | Een applicatie die over het internet geleverd wordt. Deze applicatie dient als platform om software op te maken. |
| Upstream | Het open source project dat de bron is van de gebruikte code. |

# Hoofdstuk 1

# Inleiding

Big Data, een term die de laatste jaren ontzettend populair geworden is, [1] is eigenlijk een nieuwe term voor een oud probleem. Data wordt steeds complexer waardoor traditionele dataverwerkingstechnieken niet meer toereikend zijn. Douglas Laney[7] beschrijft de uitdagingen door middel van drie V's.

- **Volume**: De datasets worden steeds groter.
- **Velocity**: Data groeit sneller aan en analyses moeten sneller beschikbaar zijn.
- **Variety**: Steeds meer datasets van verschillende bronnen, met verschillende structuur en semantiek moeten samengevoegd en samen geanalyseerd worden.

Deze drie V's werden uiteindelijk de basis van de consensusgebaseerde definitie van Big Data: *"Information assets characterized by such a High Volume, Velocity and Variety to require specific Technology and Analytical Methods for its transformation into Value"*[16].

## 1.1 Probleemstelling

De grote populariteit van Big Data heeft zich vertaald in een explosie van software gerelateerd aan data analyse en opslag. Dit gaat van producten die grote datasets opslaan en analyseren tot producten die het samenwerken van software in een cluster coördineren. Veel producten specificeren zich op specifieke scenario's en specifieke delen van een Big Data workflow. Er is veel keuze uit tools die op een complexe manier samenwerken waardoor het als ontwikkelaar

[1] www.google.com/trends/explore#q=big%20data

of onderzoeker moeilijk is om door het bos de bomen te zien. Voor de ontwikkeling van een Big Data applicatie kan beginnen moeten er drie problemen opgelost worden.

Het eerste probleem is het vinden van een geschikte **architectuur** voor het platform waar de eigenlijke applicatie op zal draaien. Hoe zal de data opgeslagen worden? Hoe wordt de data verwerkt? Hoe gebeurt de communicatie naar andere services? Er bestaan verschillende manieren om Big Data problemen op te lossen en om beschikbare softwarepakketten te combineren.

Eenmaal beslist is hoe de architectuur van het platform er uit gaat zien, moet beslist worden met **welke softwarepakketten** er specifiek gewerkt zal worden. Verschillende softwarepakketten lossen hetzelfde probleem op, en het zoeken van het geschikte softwarepakket kan veel tijd in beslag nemen.

Als laatste probleem stelt zich **het eigenlijke opzetten van het platform** waarop de Big Data applicatie zal draaien. Om een Big Data platform op te zetten is er veel kennis nodig van systeembeheer en de interne werking van de gekozen softwarepakketten. Deze kennis heeft weinig relevantie voor de kerntaak van een Big Data ontwikkelaar.

Het oplossen van deze problemen neemt veel **tijd** in beslag. Kostbare tijd die beter kan gebruikt worden voor het ontwikkelen en onderzoeken van Big Data applicaties. Daarbovenop vereist het oplossen van deze problemen ook veel **kennis** van de ontwikkelaar of onderzoeker. Deze kennis is niet altijd even relevant voor de kerntaak, en deze kennis vergroot de drempel om te starten als Big Data ontwikkelaar of onderzoeker.

In het boek *Big Data: Principles and best practices of scalable realtime data systems*[6] beschrijft Nathan Marz de Lambda architectuur, een *best practice* architectuur om veelzijdige Big Data problemen op te lossen. Deze architectuur zou een oplossing zijn voor het eerste probleem, namelijk de architectuur. Er is echter geen eenvoudige manier om dit uit te testen, doordat er geen platform is om verschillende Big Data opstellingen op te zetten en te testen.

## 1.2 Doelstelling

Het doel van deze masterproef is om een schaalbaar en modulair Big Data Platform-as-a-Service (PaaS) te maken dat onderzoekers en ontwikkelaars kunnen gebruiken om Big Data applicaties te bouwen. Dit platform moet schaalbaar zijn om real-world Big Dataproblemen aan te kunnen. Dit platform moet ook modulair zijn zodat het eenvoudig is om verschillende opstellingen en softwarepakketten uit te testen. Kort gezegd: Gebruikers van het platform

kunnen zich focussen op de Big Data problemen in plaats van op de onderliggende infrastructuur.

Dit platform wordt echter niet vanaf nul opgebouwd. Voor deze masterproef wordt er vertrokken vanaf Tengu, een implementatie van de Lambda architectuur[17][2] die ontwikkeld wordt bij IBCN. Deze implementatie moet omgezet worden in een schaalbaar en modulair PaaS.

## 1.3 Overzicht

In Hoofdstuk 2 wordt de Lambda architectuur theoretisch onderzocht door middel van een literatuurstudie. Hoofdstuk 3 gaat dieper in op de componenten waar Tengu, een implementatie van die Lambda architectuur, uit is opgebouwd.

De vereiste functionaliteit van een Big Data PaaS wordt besproken in Hoofdstuk 4. In dat Hoofdstuk wordt ook besproken waarom het niet mogelijk is om een dergelijk platform te ontwikkelen door middel van conventionele policy-based configuratiebeheer tools zoals eerst beschreven door Mark Burgess [18].

Hoofdstuk 5 stelt een architectuur voor om Tengu om te zetten naar een schaalbaar en modulair Big Data PaaS dat voldoet aan de vereisten. Dit door middel van een Service Orchestration tool. De eigenlijke implementatiedetails worden verder uitgediept in Hoofdstuk 6. Hoofdstuk 7 onderzoekt de voor- en nadelen van verschillende manieren om één service te implementeren.

Hoofdstuk 8 geeft een algemene conclusie en enkele mogelijke uitbreidingen.

[2] tengu.intec.ugent.be

# Hoofdstuk 2

# Literatuurstudie

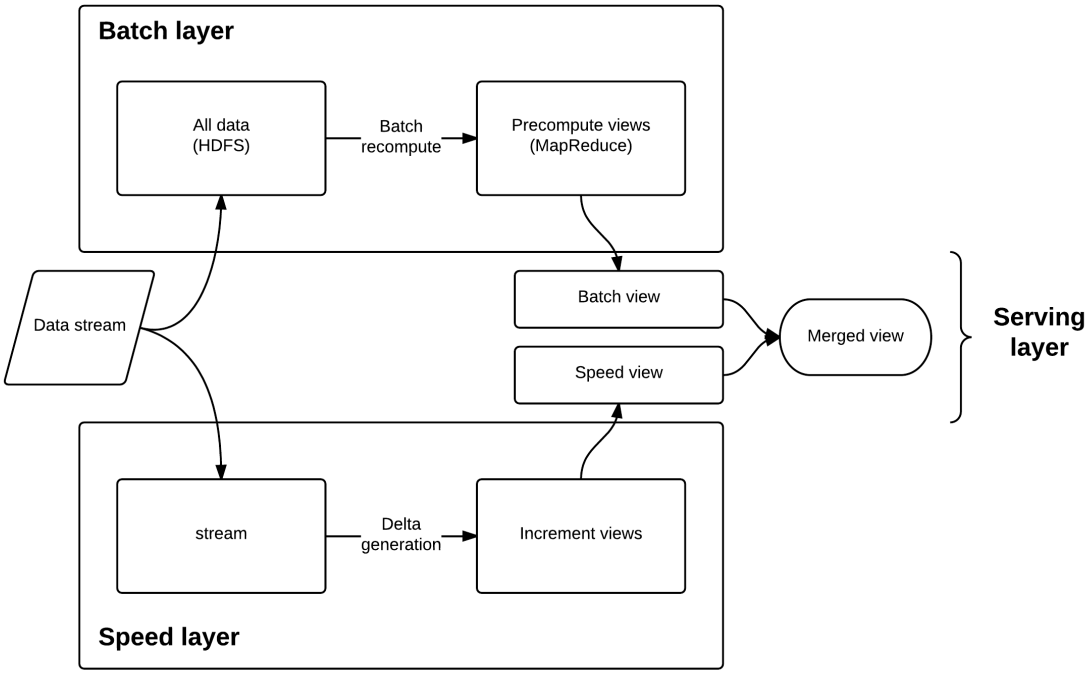
Om de uitdagingen bij Big Data applicaties uit te leggen wordt het voorbeeld van een Big Data applicatie die Twitter afzoekt op cyberpesten bekeken. Deze applicatie moet in korte tijd veel data verwerken. Op die data moet er ook een zware analyse gebeuren. Cyberpesten kan in één kleine post zitten, die enorme gevolgen kan hebben. Een voorbeeld hiervan zijn doodsbedreigingen. Hierop moet onmiddellijk gereageerd worden. Cyberpesten kan echter ook subtieler zijn, maar dan over een lange periode. Als iemand maandenlang negatieve berichten stuurt naar een andere persoon, dan zal dit ook actie vereisen.

Er bestaat software om realtime grote stromen data te analyseren. Deze software kan echter niet kijken naar de context van een bericht en de geschiedenis van betrokken personen. Er bestaat ook software om heel veel data, met context en geschiedenis te analyseren, maar de verwerking van data gaat dan heel traag, en moet in blokken gebeuren. Een softwarepakket dat beide problemen oplost bestaat dus niet.

## 2.1 De lambda architectuur

De lambda architectuur is een dataverwerkingsarchitectuur gemaakt om realtime big data te verwerken. De lambda architectuur lost het probleem van de Twitter cyberpesten applicatie op door een speed-(snelheid) en een batchlaag te voorzien. Data wordt realtime en snel geanalyseerd in de speedlaag, maar wordt ondertussen ook traag en grondig verwerkt in de batch-laag. Het resultaat van beide lagen wordt samengevoegd en aan de gebruiker gegeven. Zie Figuur 2.1.

Alle onverwerkte data die het systeem binnenkomt wordt in de master dataset van de batch-laag gestoken. De records van deze dataset zijn *immutable*. Dit wil zeggen dat eenmaal data



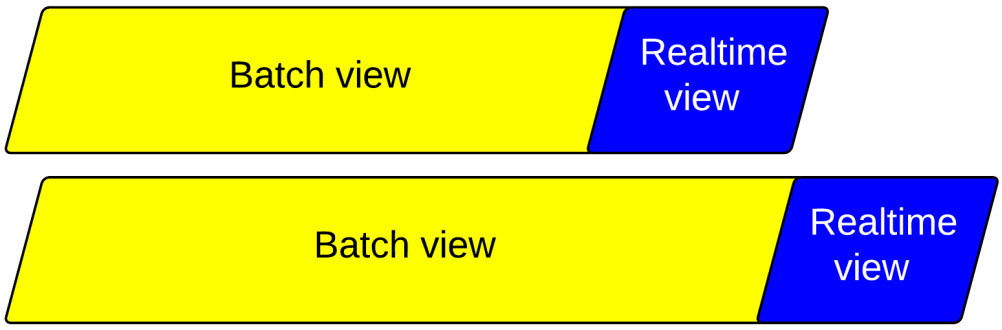

**Figuur 2.1:** De lambda architectuur

in de set zit, deze nooit meer gewijzigd wordt. Om een record te wijzigen wordt er dus een nieuwe versie van die record toegevoegd. Uit deze dataset worden constant de batch views berekend. Deze dataset blijft echter groeien, en het berekenen van deze views is zeer intensief, waardoor dit veel tijd in beslag neemt. Het resultaat van deze berekeningen zal pas na lange tijd beschikbaar zijn.

Er is dus een lange periode tussen het binnenkomen van de data en het moment waarop die data zichtbaar is in de views. Die tijdspanne wordt overbrugd door de berekeningen van de speed laag.

In plaats van de volledige dataset opnieuw te analyseren, berekent de speed laag enkel de delta's, de verschillen. Tijdens de deltaberekeningen wordt de nieuwe speed view gemaakt door analyse van de vorige speed view en de nieuwe binnengekomen data. Enkel het verschil tussen de oude en de nieuwe speed view wordt berekend. De analyse gaat veel sneller doordat de te analyseren dataset veel kleiner is. Deltaberekeningen zijn echter veel complexer dan batchberekeningen. Ook zijn sommige berekeningen heel moeilijk om te zetten naar hun delta vorm, of geven ze een heel onnauwkeurig resultaat.

Het resultaat van de speed en de batch laag wordt op een slimme manier samengevoegd tot de *merged view*. Zoals eerder beschreven bevatten de batch view en de speed view de resultaten van analyse van dezelfde data. De speed view bevat echter analyse van recentere data. In de merged view zijn de resultaten afkomstig van de speed view doorzichtig. Wanneer er een batch view klaar is met analyse overschrijft die de analyse van de speed view. Geanalyseerde data afkomstig uit de speed view wordt dus zeer snel vervangen door geanalyseerde data afkomstig uit de batch view. Zie Figuur 2.2.




**Figuur 2.2:** De merged view is een combinatie van de realtime view en de batch view.

Deze manier van werken heeft verschillende voordelen:

- Een combinatie van real-time views en diepgaande analyse.
- *Complexity isolation*: Het meest complexe deel van de architectuur, het berekenen van de delta's in de speed laag, heeft maar een tijdelijk effect op de data. Door het snel berekenen van delta's wordt veel informatie overgeslagen tijdens de analyse. Het resultaat van de speed laag is er snel, maar dit gaat ten koste van de complexiteit. Grotere complexiteit geeft een grotere kans op fouten. Deze resultaten zijn echter maar voor korte tijd zichtbaar in de merged view. Indien bepaalde data al meegenomen is in de berekeningen van de batch view, dan is enkel het resultaat van de batch berekeningen van die data zichtbaar in de merged view. Mogelijke fouten van de speed laag worden dus constant weggewerkt door de batch laag.
- *Human Fault tolerant analysis*: De batch views worden steeds op de volledige onverwerkte dataset berekend. Een fout in het analysealgoritme heeft daardoor enkel tijde-

lijke gevolgen. Het algoritme wordt geschreven door mensen, dus kan fouten bevatten. Na het herstellen van het algoritme is de fout opgelost in de volgende batch view.

- *Human Fault tolerant data storage*: De master dataset, waaruit de batch views worden berekend, is immutable. Bestaande records kunnen niet gewijzigd worden, er kan enkel nieuwe data aan toegevoegd worden. Een bestaand record updaten gebeurt door een nieuw record met de update toe te voegen. De dataset kan vervuild worden met foute data, bijvoorbeeld door een menselijke fout in het programma dat de data levert. In dit geval kan men de dataset eenvoudig terugzetten naar een vorige staat door alle records toegevoegd na een bepaald tijdstip te verwijderen. Fouten in de input van data hebben dus enkel tijdelijke gevolgen.

- Onzichtbaar: De eigenlijke werking van de architectuur is onzichtbaar bij het uitvoeren van query's op de view. Applicaties moeten niet aangepast worden, deze architectuur kan een *drop-in* vervanging zijn voor een bestaande oplossing.

# Hoofdstuk 3

# Technologiestudie

## 3.1 Tengu

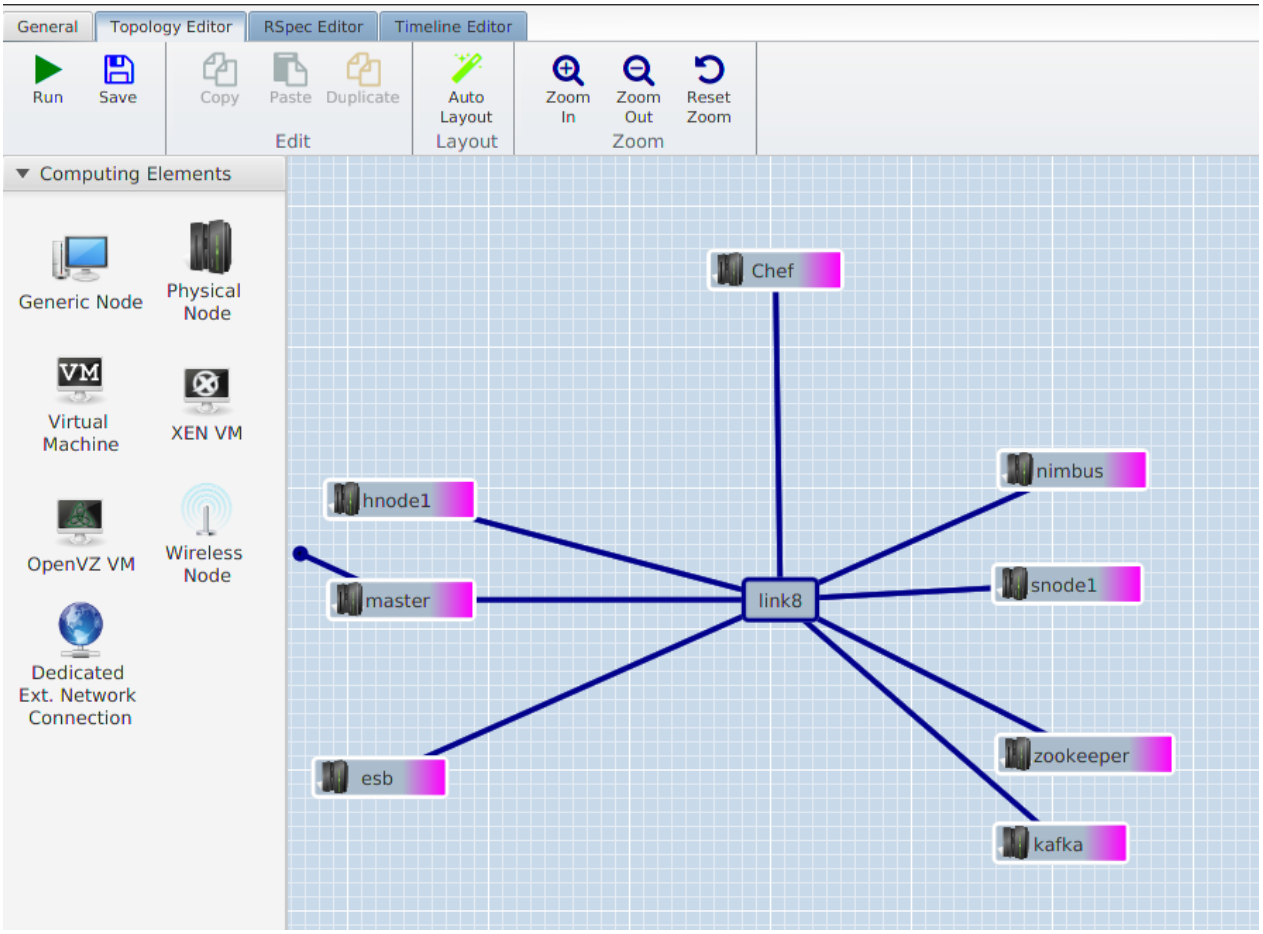
Tengu[17][1] is een implementatie van de lambda architectuur die ontwikkeld wordt bij IBCN. Tengu gebruikt Apache Hadoop als batchlaag en Apache Storm als speedlaag. De verschillende services communiceren door middel van de Enterprise Service Bus (ESB) WSO2. De services zijn verspreid over verschillende clusters op verschillende nodes. De installatie en configuratie van Tengu is al deels geautomatiseerd door middel van Chef cookbooks. Dit voldoet echter nog niet aan alle vereisten. Meer hierover in Hoofdstuk 4.

Tengu draait op de virtual wall[2]. De virtual wall is een *Infrastructure as a Service* (IaaS) project waar je fysieke machines en netwerken kan aanmaken alsof je in een virtuele omgeving aan het werken bent. Het beheer van de virtual wall gebeurt door middel van de jFed tool.

Het Tengu platform bestaat uit verschillende technologieën die met elkaar samenwerken zoals te zien op Figuur 3.1. De WSO2 Enterprise Service Bus verzorgt de communicatie tussen verschillende componenten. De speed laag bestaat uit Apache Storm, Kafka en Zookeeper. De batch laag bestaat uit Apache Hadoop. De technologieën worden uitgebreid uitgelegd in dit hoofdstuk.

[1] tengu.intec.ugent.be

[2] ilabt.iminds.be/iminds-virtualwall-overview




**Figuur 3.1:** De default Tengu installatie

## 3.2 WSO2 ESB

De WSO2 *Enterprise Service Bus*[3] is het middelpunt van het Tengu platform. Alle communicatie tussen de verschillende delen van het Tengu platform gebeurt via de ESB. De ESB vertaalt berichten van de ene service naar een formaat dat de andere service begrijpt. De ESB ondersteunt dan ook verschillende protocollen zoals HTTP, REST en SOAP en kan berichten van het ene protocol omzetten naar het andere protocol. Ook kan de ESB berichten doorsturen naar verschillende services op basis van hun inhoud of metadata.

[3] www.wso2.com/esb

## 3.3 Apache Hadoop

Apache Hadoop[4] is de batch laag in het Tengu platform. Hadoop is een Java framework gemaakt om hele grote blokken data op te slaan en in hun context te analyseren. Twee belangrijke delen van Hadoop zijn HDFS en MapReduce. Hadoop is gemaakt met het idee dat ieder individueel component op ieder moment kan uitvallen, en dat dit automatisch opgevangen moet worden.

### 3.3.1 Hadoop HDFS

Hadoop Distributed File System is, zoals de naam zegt, een gedistribueerd bestandssysteem. Hier wordt de master dataset opgeslagen zoals besproken in Hoofdstuk 2.1. Bestanden op het HDFS spreek je op dezelfde manier aan als bestanden op een Unix bestandssysteem. HDFS kan op Linux zelfs gemount worden door middel van *Filesystem In Userspace* (FUSE). Door deze low level aanpak is het heel eenvoudig om HDFS te integreren in bestaande toepassingen. Deze interface is echter minder abstract dan wat we gewoon zijn van een database. Het is dan ook aangewezen om een abstractielaag bovenop de HDFS interface te gebruiken.

### 3.3.2 Hadoop MapReduce

Hadoop MapReduce maakt het mogelijk om op een heel gedistribueerde manier grote datasets te analyseren. MapReduce slaat op de twee grote fases waarin de data verwerkt wordt: Map en Reduce. De functionaliteit is gemakkelijk uit te leggen met een voorbeeld: wordcount, het bepalen van de frequentie waarmee ieder woord voorkomt in een tekst.

Tijdens de **Map** fase wordt de Map functie parallel uitgevoerd op kleinere subsets van de data. Voor een wordcount zal de map functie iedere lijn apart analyseren en de frequentie van ieder woord bepalen. Iedere map functie geeft een verzameling van key-value pairs terug, met als key het woord en als value de frequentie van dat woord op de lijn. Het MapReduce framework verzamelt alle resultaten van alle Map functies en groepeert waarden met dezelfde key.

Wanneer alle Map functies hun resultaat hebben teruggegeven, begint de **Reduce** fase. Hierin wordt de Reduce functie parallel uitgevoerd op iedere verzameling van waarden met dezelfde key. Bij het wordcount voorbeeld worden alle individuele frequenties opgeteld om een globale frequentie van dat woord in de tekst te geven.

[4] hadoop.apache.org

Het is mogelijk om meerdere MapReduce fases te hebben. Het is belangrijk om zoveel mogelijk bewerkingen tijdens de Map fase uit te voeren, aangezien die veel gedistribueerder uitgevoerd kan worden. De Reduce functie heeft een apart parallel proces voor iedere key terwijl de Map fase een apart parallel proces heeft voor iedere subset van de data.

## 3.4 Apache Storm

Apache Storm[5] is de speed laag in het Tengu platform. Storm zorgt voor snelle, real-time analyse van data.

Apache Storm analyseert real-time. Storm heeft enkele abstracties om het programmeren van analysesoftware eenvoudiger te maken. Een **spout** is een bron van een datastroom. Een spout kan bijvoorbeeld data ophalen uit Kafka. De eigenlijke analyse van de data gebeurt in een **bolt**. Een bolt kan meerdere invoerstromen analyseren en meerdere uitvoerstromen produceren. Een bolt kan functies uitvoeren op één of meerdere datastromen, communiceren met een database en meer. Spouts en bolts kunnen in een netwerk gezet worden om zo geavanceerde functionaliteit te krijgen. Dergelijk netwerk wordt een topology genoemd.

## 3.5 Kafka

Apache Kafka[6] is een *message broker*. Kafka stelt een interface open om berichten te publiceren en berichten af te halen. Kafka kan heel veel berichten verwerken in korte tijd. Kafka kan als cluster opgezet worden, waarbij het werk verspreid wordt over verschillende *nodes*. Berichten worden gepubliceerd op *topics* en clients abonneren op die topics. Iedere client loopt met zijn eigen iterator over de topic. Clients kunnen zo op hun eigen tempo berichten verwerken waardoor Kafka kan werken als cache.

Kafka wordt gebruikt in Tengu als cache voor de communicatie tussen de ESB en Storm. De ESB schrijft data naar Kafka op het moment dat de ESB die binnenkrijgt. Storm leest de data van Kafka op zijn eigen tempo. Kleine verschillen tussen de snelheid waarmee berichten binnenkomen en de snelheid waarmee Storm die kan verwerken worden op deze manier opgevangen.

[5] storm.apache.org
[6] kafka.apache.org

## 3.6 Apache Zookeeper

Om services in een cluster te laten draaien is er heel veel communicatie nodig en moeten de verschillende nodes op een complexe manier samenwerken. Bij het programmeren van een gedistribueerde service is er veel extra programmeerwerk nodig dat eigenlijk niets te maken heeft met de kerntaak van de service. Naast de kennis over de kerntaak van de service hebben programmeurs dan ook een grote kennis over gedistribueerde applicaties nodig. Apache Zookeeper[7] biedt deze functionaliteit als service aan.

De essentie van gedistribueerde services is de communicatie tussen de verschillende componenten. Zookeeper voorziet een eenvoudige interface om informatie uit te wisselen. Je kan Zookeeper het best vergelijken met een gedistribueerd gesynchroniseerd bestandssysteem. Hierop kunnen de verschillende componenten van een gedistribueerde service informatie zetten en afhalen.

Zookeeper is geoptimaliseerd voor leesoperaties. Zookeeper werkt dus het best voor opdrachten met een hoog aantal leesoperaties en een laag aantal schrijfoperaties. Zookeeper wordt gebruikt door Storm en Kafka en neemt de communicatie en de synchronisatie tussen nodes in individuele clusters op zich.

## 3.7 Chef

Chef[8] is een configuratiebeheer tool. Chef beheert de configuratie van individuele servers. Chef kan ook verbinden met verschillende *cloud* platformen zoals Amazon EC2, OpenStack en Microsoft Azure om servers dynamisch aan te vragen en te verwijderen.

Chef werkt door middel van Cookbooks. Dit zijn Ruby scripts die verschillende acties uitvoeren zoals software installeren en configuratiefiles aanpassen. Chef Cookbooks zijn idempotent. Dit wil zeggen dat het tweemaal uitvoeren van hetzelfde Cookbook hetzelfde resultaat moet geven. Cookbooks worden regelmatig heruitgevoerd om te verzekeren dat de configuratie van de server nog steeds juist is.

Standaard werkt Chef in client/server modus. Hierbij biedt een server Cookbooks en configuratie-informatie aan één of meerdere clients aan. Er is ook de mogelijkheid om in *local-mode* te werken. Hierbij zijn client en server dezelfde machine. De client houdt dan zijn eigen Cookbooks en configuratie informatie bij.

[7] zookeeper.apache.org
[8] chef.io

## 3.8 De Virtual Wall

Als test- en ontwikkelomgeving wordt de Virtual Wall[9] gebruikt. Dit is een *Infrastucture as a Service* project waarbij fysieke servers en netwerkverbindingen aangevraagd kunnen worden alsof het virtuele machines zijn.

### 3.8.1 jFed

Het beheer van de nodes op de virtual wall gebeurt door middel van jFed[10]. JFed is ontwikkeld binnen het Europese Fed4FIRE project en maakt het mogelijk om op een gestandaardiseerde manier verschillende servers automatisch op te zetten. jFed geeft je een gebruiksvriendelijke interface om experimenten aan te maken bestaande uit verschillende nodes met netwerkverbindingen ertussen. Met behulp van jFed kan je ook:

- experimenten exporteren naar een `rspec` bestand om te delen met andere gebruikers
- images maken van bestaande nodes, en die op andere nodes zetten
- opstartscripts meegeven die uitgevoerd worden bij het initialiseren van een node

## 3.9 Conclusie

Dit hoofdstuk gaf een kort overzicht van de technologieën waaruit Tengu bestond bij de start van deze masterproef. Aan de hand van deze technologieën biedt Tengu een standaardimplementatie van de Lambda architectuur aan. Het doel van deze masterproef is om Tengu om te zetten naar een schaalbaar en modulair PaaS. In Hoofdstuk 4 wordt de vereiste functionaliteit van dit PaaS verder beschreven.

[9] www.iminds.be/en/succeed-with-digital-research/technical-testing
[10] jfed.iminds.be

# Hoofdstuk 4

# Functionele analyse

Het doel van deze masterproef is de initiële versie van Tengu omzetten in een modulair en schaalbaar PaaS. Voor er gekeken kan worden hoe deze omzetting het best gebeurt, moet gedefinieerd worden wat nu net de vereiste functionaliteit is.

Dit hoofdstuk maakt een functionele analyse en bespreekt hoe sommige eisen verschillen van conventionele eisen.

## 4.1 Infrastructure as Code

Om het platform als een service te kunnen aanbieden moet de installatie en configuratie volledig automatisch gebeuren. De waarde van de infrastructuur zit niet meer in de opgezette infrastructuur, maar in de code om de infrastructuur automatisch op te zetten. Dit is het *Infrastructure as code* paradigma. Aangezien de waarde nu in de codebase zit, is de kwaliteit van de codebase heel belangrijk. Concepten die gebruikt worden in de softwareontwikkeling moeten toegepast worden op de ontwikkeling van de infrastructuur. Infrastructuurcode moet van hoge kwaliteit, onderhoudbaar, uitbreidbaar en betrouwbaar zijn. In de wereld van Big Data evolueren technologieën snel en komen er veel nieuwe technologieën bij. Het platform moet deze ontwikkelingen kunnen bijhouden. Het is dus nodig dat ontwikkelaars het platform eenvoudig kunnen onderhouden en uitbreiden.

Modulariteit is hier van groot belang. Modulariteit zorgt ervoor dat verschillende componenten aangepast kunnen worden zonder invloed op andere componenten. Dit wordt verder besproken in Sectie 4.3. Abstractie is ook belangrijk. Abstractie zorgt voor eenvoudig onderhoud en hergebruik. De programmeeromgeving moet concepten die te maken hebben met het besturingssysteem abstraheren. Het moet eenvoudig zijn om code om te zetten naar een

ander besturingssysteem en details moeten zoveel mogelijk weggewerkt worden.

## 4.2 Ontplooien applicaties

Het Big Data PaaS bestaat uit verschillende services waarvan er veel zelf een platform zijn om code op te draaien. Gebruikers willen hun applicatie snel kunnen testen. Ieder stukje code handmatig deployen op iedere service neemt veel tijd in beslag en vereist enige kennis over het platform zelf. Er moet een interface komen om eenvoudig applicaties bovenop het platform te ontplooien.

## 4.3 Modulariteit

Een modulair platform is een platform dat samengesteld is uit eenheden die op verschillende manieren gecombineerd kunnen worden [1]. Toegepast op een Big Data PaaS wil dit zeggen dat individuele services kunnen toegevoegd, verwijderd en vervangen worden. Dit is belangrijk om twee redenen:

1. Ieder Big Data probleem is verschillend, en vraagt een unieke aanpak. Het platform moet flexibel genoeg zijn om zich aan te passen aan de noden.

2. Er bestaan veel verschillende softwarepakketten die een vergelijkbare functionaliteit hebben. Het platform moet gebruikers de mogelijkheid geven om verschillende pakketten uit te testen. Dit in het kader van onderzoek of om te kijken op welke manier een probleem het best opgelost wordt.

Een **service** moet gedefinieerd kunnen worden als een onafhankelijk geheel dat zich dynamisch aanpast aan **verbindingen** met andere services. Adam Jacob, de maker van de configuratiebeheer tool Chef, beschrijft de basis voor een dergelijke aanpak in Hoofdstuk 5 van het boek *Web Operations* [9]. Hij gebruikt de term *Service-oriented Architecture*:

> *You break each part of the applications down to its most primitive parts, and make each of those available over a network as a service, enabling you to create ever more flexible, powerful infrastructure with less overhead and time.*

[1] http://www.almaany.com/en/dict/en-nl/modulariteit/

Deze modulaire en abstracte kijk op infrastructuur wordt echter niet ver genoeg doorgetrokken bij conventionele configuratiebeheer tools zoals Puppet en Chef. Deze tools gaan uit van een statisch model waar een **server** een **rol** krijgt. Om twee servers te verbinden krijgt iedere server een nieuwe rol. In het conventionele model wordt een service dus gedefinieerd door alle mogelijke staten die een server kan hebben. Alle mogelijke staten, en dus ook relaties, moeten hierbij op voorhand gekend zijn. Bemerk ook het verschil tussen servers en services. Dit wordt verder besproken in Sectie 4.4.

Er moet ook een *peer-to-peer* netwerk zijn dat uitwisseling van informatie tussen verbonden services mogelijk maakt. Bij conventionele configuratiebeheer tools gebeurt dit op een top-down manier. Indien twee verbonden servers informatie van elkaar nodig hebben, dan moet die door de master aangeboden worden. Dit zorgt voor minder flexibiliteit.

## 4.4 Schaalbaarheid

Schaalbaarheid is ook een zeer belangrijke functionaliteit van een Big Data Paas. Het moet triviaal zijn om services die draaien op één node uit te breiden naar verschillende nodes. Een service mag dus niet vasthangen aan één server. Een service die als cluster draait over meerdere servers moet op dezelfde manier beheerd kunnen worden als een service die op één enkele node staat. Conventionele configuratiebeheer tools werken echter op een andere manier. Zij beheren individuele servers in plaats van volledige services. Om het met de woorden van Adam Jacob [9] te zeggen:

> *Configuration management means keeping track of all the stuff you have to do to take a system from "bare metal" to "doing its job."*

Configuratiebeheer tools zetten dus servers in een bepaalde staat. Een cluster wordt gemaakt door verschillende **servers** verschillende **rollen** te geven. Dit is duidelijk een abstractieniveau lager dan **nodes** toevoegen aan **services**.

## 4.5 Ontplooien standaardopstelling

Het moet mogelijk zijn om als gebruiker tussen verschillende standaardopstellingen te kiezen. Het moet dus mogelijk zijn om een groep van services en verbindingen in één keer te ontplooien.

## 4.6 Gebruiksgemak

Het gebruiksgemak is belangrijk voor iedere PaaS. Dit is een van de grootste redenen om te kiezen voor een PaaS oplossing en het gebrek aan gebruiksvriendelijkheid is een van de grootste problemen die in het kader van deze masterproef opgelost moet worden. Gebruiksgemak is een heel brede term. Volgende zaken zijn heel interessant in het geval van een Big Data PaaS:

1. **Abstractie tegenover gebruiker:** De juiste vorm van abstractie kan het gebruiksgemak van een platform sterk verbeteren. De abstractie moet het mogelijk maken om het idee van een gebruiker om te zetten in praktijk zonder dat de gebruiker zich moet bezighouden met onbelangrijke details. De ideeën van de gebruiker moeten eenvoudig vertaalbaar zijn naar componenten en handelingen op het platform. Secties 4.3 en 4.4 werken dit verder uit.

2. **Functionele interface:** Een functionele interface doet twee dingen: Hij toont de gebruiker informatie en functionaliteit die op dat moment relevant zijn en hij geeft een beeld van welke extra informatie en functionaliteit er beschikbaar is.

3. **Aantrekkelijke interface:** Naast functioneel moet de interface ook aantrekkelijk zijn. Dit is om nieuwe gebruikers over te halen om het platform uit te testen.

## 4.7 Multi-tenancy

Multi-tenancy slaat op het ondersteunen van meerdere gebruikers op dezelfde instantie van een applicatie. Multi-tenancy is belangrijk op twee niveaus. Langs de ene kant wil de provider van het Platform dit aan meerdere gebruikers kunnen aanbieden op zo een efficiënt mogelijke manier. Langs de andere kant is de klant meestal niet één individuele persoon. Het moet dan mogelijk zijn dat verschillende leden van een team, elk met hun eigen rechten en verantwoordelijkheden het platform gebruiken.

## 4.8 Stabiliteit

Een Big Data PaaS moet bestand zijn tegen het uitvallen van een service. Het uitvallen van een service kan verschillende oorzaken hebben:

1. De node waarop de service draait valt uit. Dit kan te wijten zijn aan hardwareproblemen op de node zelf of op het netwerk, ernstige softwareproblemen zoals een kernel panic, of fouten bij het onderhoud, zoals een kabel die per ongeluk is uitgetrokken.

2. De processen die de service voorzien crashen. Dit kan komen door bijvoorbeeld een fout in de software van de service of een fout in het programma dat op de service draait.

3. De configuratie van een service is fout waardoor de processen van de service niet meer uitgevoerd kunnen worden. Dit kan gebeuren als een administrator of een programma de configuratie aanpast.

4. De processen draaien zonder foutmeldingen maar de service levert zijn diensten niet correct af. Ook hier kan dit liggen aan een foute configuratie, een fout in de software die op de service draait, of een fout in de configuratie van een andere service.

Op verschillende niveaus moet de staat van de server dus gecontroleerd worden.

## 4.9 Relevantie

Het PaaS moet natuurlijk op lange termijn relevant blijven. Langs de ene kant moeten de aangeboden services voldoen aan de noden van de klanten. Anderzijds moet het platform op een stevige basis gebouwd worden. Als het platform sterk afhankelijk is van bepaalde projecten, dan is het belangrijk dat die projecten op lange termijn ondersteund worden.

# Hoofdstuk 5

# Architectuur

Om aan alle vereisten te kunnen voldoen wordt gebruik gemaakt van Juju en Chef. In dit hoofdstuk wordt eerst de rol van Juju en Chef toegelicht. Daarna wordt er een algemeen overzicht van de architectuur gegeven.

## 5.1 Juju

Juju is het middelpunt van het Big Data PaaS. Juju is heel geschikt om een Big Data platform op een modulaire en schaalbare manier te automatiseren. Dit hoofdstuk legt uit hoe Juju werkt en waarom het voldoet aan de functionaliteitseisen beschreven in Hoofdstuk 4.

Juju is een *service orchestration tool*. Juju werkt een niveau hoger dan *configuration management* tools zoals Chef en Puppet. Conventionele configuratiebeheer tools zetten een server in een gevraagde staat. Juju ontplooit services en beheert hoe deze services samenwerken, ongeacht op welke server of over hoeveel servers die services verspreid staan.

Een service wordt beschreven door een Juju Charm.

### 5.1.1 Juju charms

Een Charm is een bundel van scripts, configuratiebestanden en installatiebestanden. Elke Charm is verantwoordelijk voor één service. Een Charm automatiseert volgende zaken:

- het installeren van een service
- het uitbreiden van de service over verschillende nodes

- het configureren van een verbinding tussen verschillende services

De eigenlijke functionaliteit zit in de *hooks*. Dit zijn scripts die worden aangeroepen wanneer er iets moet gebeuren. Enkele mogelijke hooks zijn:

- **install**: installeert de service op een node
- **relation-changed**: configureert een relatie tussen twee services
- **stop**: stopt de service, maakt een back-up en verwijdert de applicatie van de node

Net zoals Chef cookbooks, moeten deze taken idempotent uitgevoerd worden. Een `install` hook zou dus niets mogen doen indien de service al geïnstalleerd is.

Hooks kunnen geschreven worden in een programmeertaal naar keuze. Er zijn verschillende hulpfuncties om de hooks in Bash of Python te schrijven. Hooks kunnen ook gebruik maken van bestaande configuration management tools zoals Puppet en Chef.

Iedere Charm krijgt standaard zijn eigen nodes. Standaard Charms kunnen niet aan nodes van een andere Charm. Een Charm kan echter van het type *subordinate* zijn. Een subordinate Charm heeft toegang tot de nodes van de Charm waar hij een relatie mee heeft. Dit is handig voor bijvoorbeeld logging services of Charms die een applicatie ontplooien op een service van een andere Charm.

### 5.1.2 Relaties

Een van de sterkste punten van Juju is het systeem van relaties tussen verschillende Charms. Een relatie koppelt verschillende services aan elkaar om zo een groot geheel te maken. Relaties werken door middel van losse interfaces. Door enkele eenvoudige regels te volgen kunnen deze losse interfaces heel complexe verbindingen maken.

Een interface is gedefinieerd door een collectie van parameters die uitgewisseld worden en een reeks van acties die beide services uitvoeren. Het uitwisselen van parameters en het uitvoeren van acties gebeurt in hooks. Telkens er een event gebeurt, wordt de bijhorende hook uitgevoerd.

- Bij het opzetten van een relatie wordt initieel de `relation-joined` hook uitgevoerd.
- Telkens er een parameter wijzigt wordt de `relation-changed` hook uitgevoerd.

- Bij het beëindigen van een relatie worden `relation-departed` en `relation-broken` hook uitgevoerd.

Een interface valt te beschrijven door de acties die er gebeuren bij de verschillende hooks. Ter verduidelijking wordt hier de `client` interface van de message broker Kafka besproken. Deze werd origineel gebruikt om informatie over de Zookeeper door te geven aan mogelijke clients.

**Listing 5.1:** client interface van kafka

```
# kafka
client-relation-changed
  relation-set port
  relation-set zookeepers

# client
kafka-relation-changed
  relation-get port zookeepers
    # pas configuratiebestanden aan
```

Wanneer de `client-relation-changed` hook wordt uitgevoerd, geeft Kafka zijn eigen poortnummer en de URL van de zookeeper server door. Dit zorgt ervoor dat bij de client de `relation-changed` hook wordt uitgevoerd. De client kan nu aan de hand van de waarden de verbinding configureren. Het voordeel van deze event-based benadering is dat de client niets moet doen indien nog niet alle parameters ingesteld zijn. Wanneer Kafka die parameters doorgeeft zal de `relation-changed` hook van de client nogmaals opgeroepen worden.

In Tengu moet de ESB communiceren met Storm via Kafka. Hiervoor moeten er enkele topics aangemaakt worden in Kafka. Ook moeten de clients weten welke topics er aanwezig zijn op Kafka. Het is nu heel eenvoudig om de client interface aan te passen met deze nieuwe functionaliteit zonder de compatibiliteit met oude clients te verliezen.

**Listing 5.2:** uitgebreide client interface

```
# kafka
client-relation-changed:
  relation-set port
  relation-set zookeepers
  relation-get requested_topics
    # kijk of topics bestaan, zoniet: maak topics aan
  relation-set available_topics
```

```
# client
kafka-relation-joined:
   relation-set requested_topics

kafka-relation-changed
  relation-get port zookeepers
  # pas configuratiebestanden aan
```

Oude clients kunnen nog steeds perfect werken. Kafka krijgt dan geen aan te maken topics door. Nieuwe clients kunnen echter van de extra functionaliteit gebruik maken. Door het aanmaken van topics in de `relation-changed` hook te steken wordt dit direct dynamisch. Telkens de gevraagde topics aangepast worden door de client wordt de relation-changed hook van Kafka uitgevoerd, met nieuwe topics tot gevolg.

### 5.1.3 Providers

Juju zorgt ook voor het monitoren van de nodes en kan dynamisch nieuwe nodes aanvragen aan verschillende soorten providers. Volgende providers worden ondersteund:

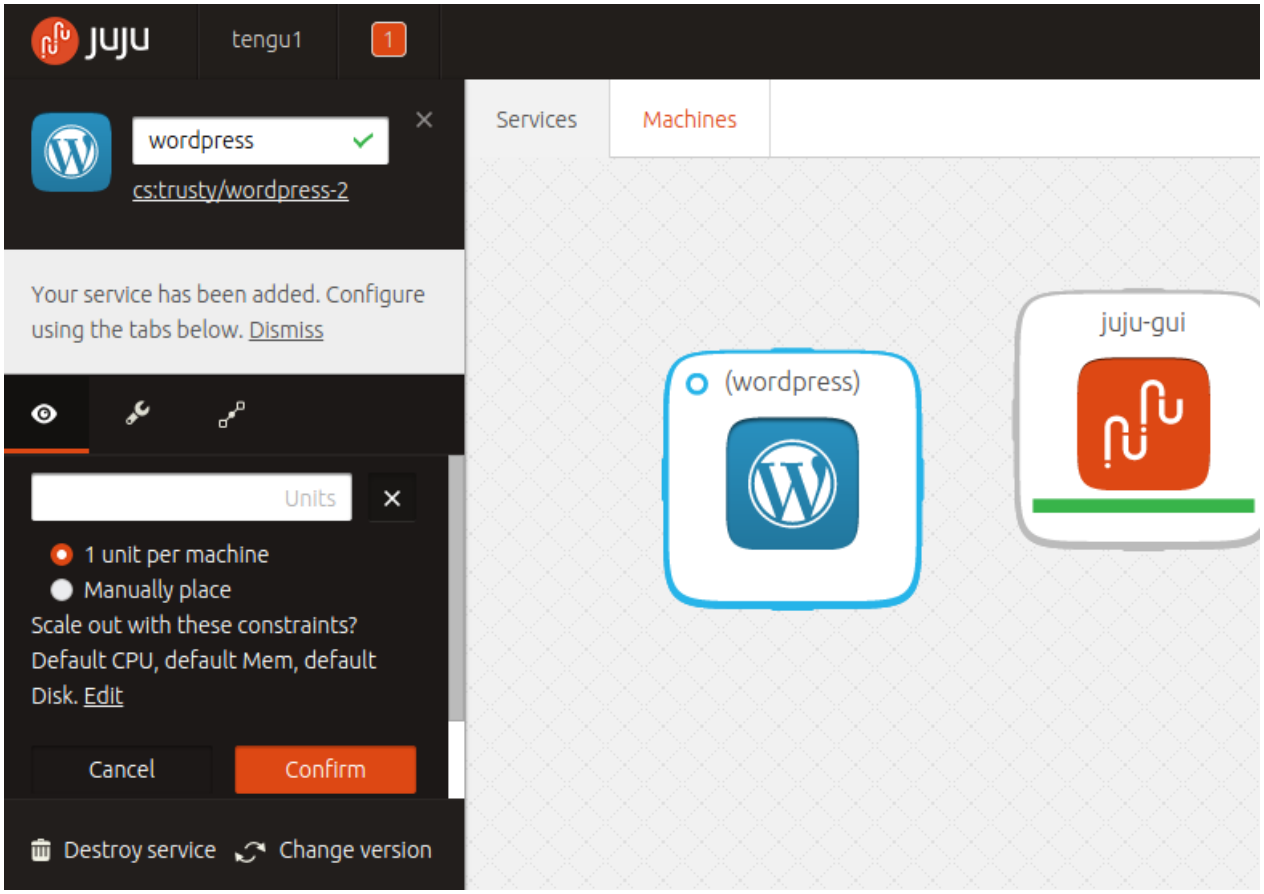
- public cloud providers zoals Amazon EC2, Digitalocean en Azure Cloud
- private cloud oplossingen zoals Openstack en MaaS
- container providers zoals LXC en Vagrant

Andere providers kunnen ondersteund worden indien er een connector voor geschreven wordt.

Naast dynamische providers, kan Juju ook werken met een statische pool van nodes. Hierbij worden nodes manueel toegevoegd aan Juju. Bij het deployen van een Charm kiest Juju dan zelf een node uit de pool. Tengu maakt gebruik van deze manual provider.

### 5.1.4 Juju GUI

Juju heeft een gebruiksvriendelijke en intuïtieve interface om services te beheren. Deze interface is zelf een Charm die als eerste ontplooid wordt op de juju-bootstrap node. Op het *Services* tabblad ziet de gebruiker een overzicht van de services die er draaien en op welke manier die verbonden zijn. Op het *Machines* tabblad ziet de gebruiker op welke nodes die services staan. In afbeelding 5.1 zie je de interface om nodes toe te voegen aan een service.




**Figuur 5.1:** De interface om nodes toe te voegen aan een cluster

### 5.1.5 Bundels

Juju heeft ook de mogelijkheid om services, relaties en configuratieparameters te groeperen in een bundel. Een bestaande opstelling kan zo opnieuw ontplooid worden.

### 5.1.6 Relevantie Juju

Het is belangrijk dat Juju op lange termijn nog steeds ondersteund wordt en relevant blijft. Volgende vragen zijn interessant om een beeld te geven van hoe relevant Juju nog zal zijn over enkele jaren:

- Zijn er grote bedrijven die diensten bouwen bovenop Juju?
- Zijn er grote bedrijven die Juju verder ontwikkelen en onderhouden?
- Wat zijn de gebruikersstatistieken?

- Worden er verschillende besturingssystemen ondersteund?

Het grootste deel van het ontwikkel- en onderhoudswerk aan Juju gebeurt door Canonical, het bedrijf achter Ubuntu. In het kader van deze masterproef is contact opgenomen met Canonical. Dit is het antwoord van Nate Finch, een van de core Juju developers bij Canonical[4]:

> *Juju is used extensively internally at Canonical. We have a mandate that all internal services be deployed via Juju.*
>
> *[...]*
>
> *IBM for example is delivering quite a few charms and has committed multiple full time development resources to working with juju.*
>
> *[...]*
>
> *We actually do support Windows, right now [...] Cloudbase is also currently tackling CentOS support [...] it should be available for testing in a few weeks.*

Mark Ramm-Christensen[1], Engineering Manager bij Canonical geeft meer informatie over diensten van Canonical die Juju gebruiken:

> *We also have some big plans for products built on top of juju. The first of which is the OpenStack Autopilot which automates the deployment, scale-out, and management of OpenStack clouds. But, we are also building more products on top of Juju right now, and it is core to our future plans in the cloud.*

In een latere e-mail[10] vernoemt hij enkele grote bedrijven die producten bouwen bovenop Juju:

> *Partners making use of juju as part of the OpenStack Interoperability Lab (OIL) include: IBM, Microsoft, Intel, AMD, HP, Juniper, Lenovo, Melanox, Metaswitch, OCP (Open Compute Project), SanDisk and VMWare.*

Verder staan de Juju developers heel open voor feedback over Juju. Na het doorgeven van alle *bugs* en *feature-requests* die invloed hebben op Tengu reageren Nate Finch[13] en Alexis Bruemmer[11], Juju Core Manager bij Canonical zeer positief:

> *I wanted to specifically thank you for pointing out the bugs that affect you. It's a huge help in prioritizing what we work on.*

Als laatste reageert ook Mark Shuttleworth, oprichter van Ubuntu.

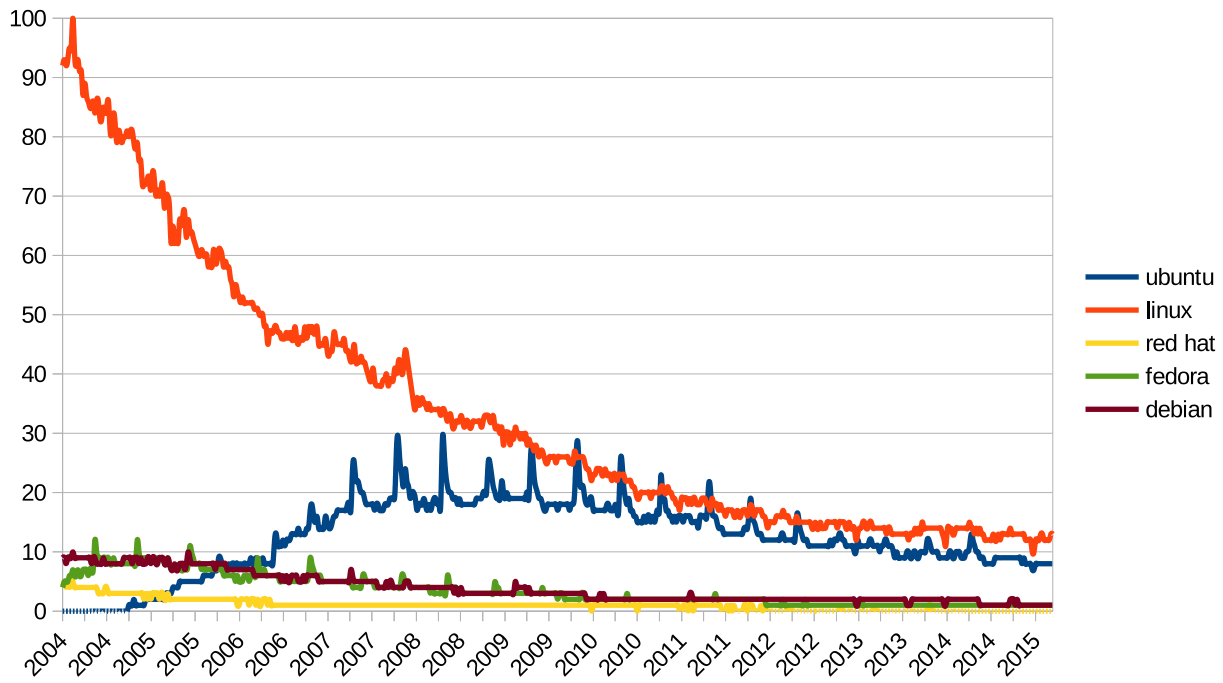
> - *Pretty much the biggest telco's in the world, and many of the companies that supply them, are writing charms.*
> - *Pretty much the biggest investment banks in the world, are writing charms.*
> - *Pretty much the biggest media companies in the world, are writing charms.*
> - *Pretty much the biggest big data and machine learning companies, are writing charms.*
>
> *[...]*
>
> *You heard it from me. I'm personally interested in Juju and it has a future, for sure, under all the scenarios I can influence. I think the cloud world needs something like Juju that is cross-platform and cross-cloud, and I find the project personally challenging and interesting.*

### 5.1.7 Relevantie Charms

Juju gaat op termijn de mogelijkheid hebben om servers met andere besturingssystemen te beheren. De Charms zelf zijn echter deels afhankelijk van Ubuntu. Hierdoor hangt de lange termijn relevantie van Tengu samen met de lange termijn relevantie van Ubuntu.

Of dit een ernstig nadeel is, hangt af van hoe populair Ubuntu is, nu en in de toekomst. Google Trends is een goed platform om objectieve data te vinden over de populariteit van woorden. Google trends maakt het mogelijk om een CSV bestand te downloaden met de relatieve populariteit van zoektermen.

Als je kijkt naar de populariteit van de zoekterm *Ubuntu* dan lijkt het alsof de populariteit van Ubuntu aan het dalen is, zie Figuur 5.2. Bij de populariteit van de zoekterm *Linux* is er nog een veel sterkere daling merkbaar. Is de populariteit van Linux aan het dalen? Een logischere verklaring is dat dit effect te wijten valt aan de populariteit van Google. Google was in de beginjaren vooral populair bij een eerder technisch publiek. Deze populariteit is uitgebreid naar een minder technisch publiek, waardoor de relatieve populariteit van technische zoektermen is gedaald. In dezelfde periode zijn de minder technische zoektermen zoals *bakery* en *recipe* gestegen en andere technische zoektermen zoals *Windows* gedaald.



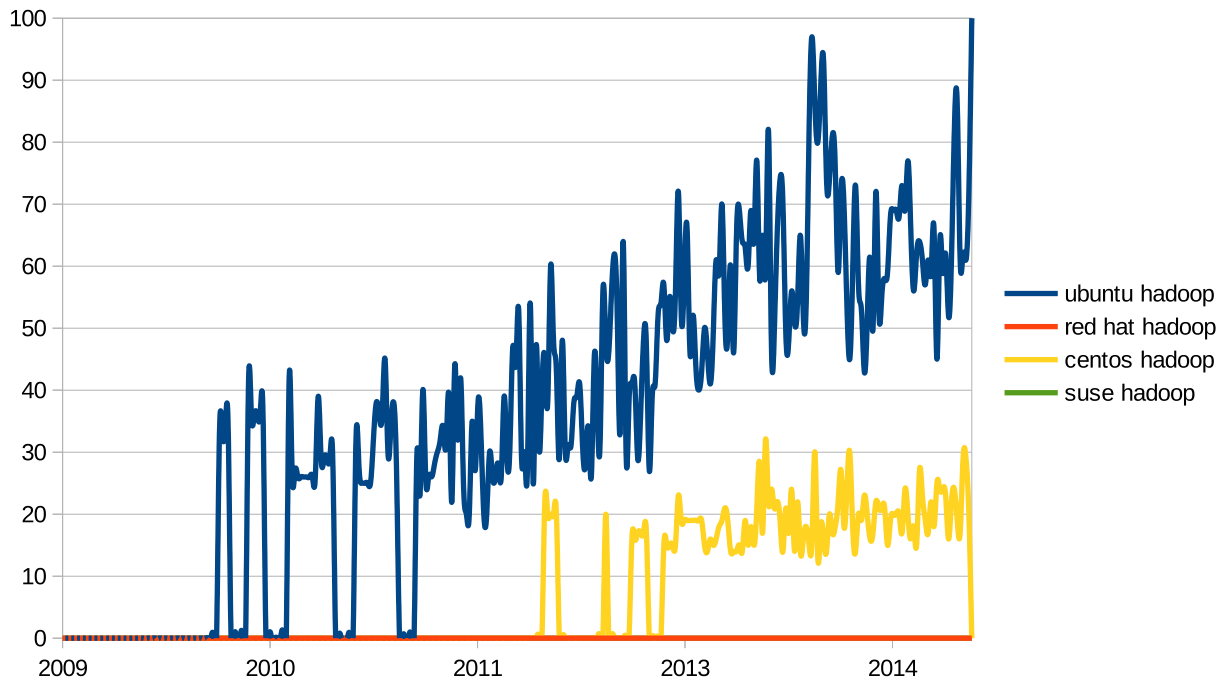

**Figuur 5.2:** Relatieve populariteit Ubuntu zoekterm

Als je de populariteit van de zoekterm Ubuntu vergelijkt met de populariteit van de zoektermen die verwijzen naar andere Linux distributies, dan steekt Ubuntu er duidelijk met kop en schouders bovenuit.

De algemene populariteit van Ubuntu is belangrijk. De populariteit van Ubuntu bij big data oplossingen is echter veel belangrijker. De combinatie van zoektermen *Ubuntu* en *Hadoop* zijn hier zeker relevant, zie Figuur 5.3. Hier is duidelijk een stijging vast te stellen. Bij de vergelijking met andere Linux distributies komt Ubuntu er weer duidelijk bovenuit. De combinatie van zoektermen Ubuntu en Hadoop is populairder en stijgt ook sneller dan die van andere Linux distributies.

Een laatste indicator voor de gezondheid van Ubuntu is het vertrouwen van grote spelers in de IT wereld. Microsoft is een plotse inhaalbeweging begonnen op vlak van Linux. Hoewel de azure cloud verschillende Linux distributies ondersteunt, draait het big data platform van Microsoft, HDInsight, volledig op Ubuntu.

Het lijkt niet alsof Tengu afhankelijk maken van Ubuntu de relevantie van Tengu zal schaden.




**Figuur 5.3:** Relatieve populariteit Ubuntu Hadoop zoekterm

## 5.2 Chef

Waar Juju tekort komt is bij het beheer van individuele nodes. Juju heeft geen eenvoudige abstracte manier om de configuratie van individuele nodes te automatiseren. Juju biedt wel de mogelijkheid om configuratiebeheer tools zoals Puppet en Chef te gebruiken voor de configuratie van individuele servers. Hier wordt dan ook gebruik van gemaakt in deze architectuur. De Charms gebruiken Chef cookbooks voor de eigenlijke installatie en configuratie.

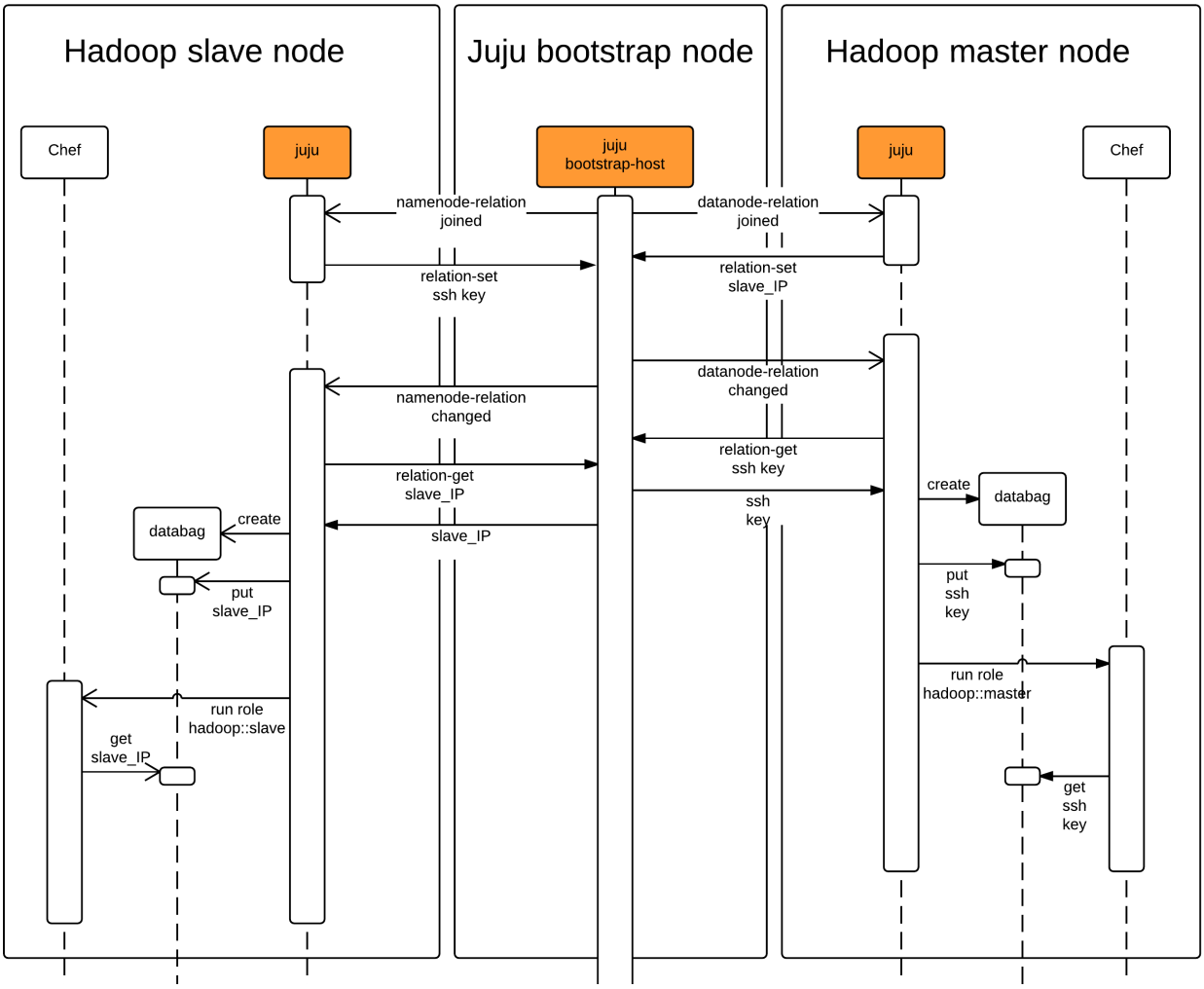
### 5.2.1 Chef Zero

Omdat een Charm geen afhankelijkheden mag hebben die zich buiten de Charm bevinden, kan er niet gewerkt worden in de Client/Server mode zoals beschreven in de technologiestudie. Een Charm mag er niet van uitgaan dat er een Chef Server beschikbaar zal zijn. *Chef-Zero* brengt hier de oplossing. Chef-Zero werkt op net dezelfde manier als Chef in Client/Server mode, met het verschil dat de Chef Client en Chef Server op dezelfde node staan. Cookbooks die afhankelijk zijn van een Chef Server werken nog steeds op Chef-Zero.

### 5.2.2 Communicatie tussen Chef en Juju

De communicatie tussen Chef en Juju kan op verschillende manieren gebeuren. Juju kan bijvoorbeeld zelf cookbooks maken die de standaard configuratieparameters van Chef overschrijven. Deze manier van werken is echter moeilijk te debuggen doordat de interface tussen Juju en Chef op voorhand niet duidelijk gekend is. Juju kan op deze manier alle parameters van het Chef cookbook aanpassen, ook al is dit niet gewenst.

Een betere vorm van communicatie tussen Chef en Juju zijn *databags*. Figuur 5.4 toont hoe dit werkt. In een databag kan informatie opgeslagen worden in het `.json` formaat. Juju steekt de nodige informatie in de databags en roept Chef-Solo op. Chef-Solo haalt de informatie uit de databags en gebruikt die in de cookbooks. Deze manier van werken zorgt ervoor dat de interface tussen Juju en Chef duidelijk gekend is. Parameters in de Chef cookbook worden pas aangepast indien de cookbook ze zelf ophaalt uit de databag. Dit maakt het debuggen van cookbooks veel eenvoudiger.




**Figuur 5.4:** Communicatie tussen Juju en Chef met databag

## 5.3 Algemeen Overzicht

Figuur 5.5 geeft een overzicht van de architectuur. Een Management website laat gebruikers toe om in te loggen en Tengu instanties aan te maken. Iedere instantie bestaat uit één beheersmachine en een pool van nodes uit een jFed experiment.

Op de beheersmachine staan het `tengu` script en de Juju ontwikkelaarshulpmiddelen. Het `tengu` script verzorgt de installatie van Juju, zoekt alle nodes in het jFed experiment en geeft deze door aan Juju. Als laatste wordt de Juju GUI ontplooid op een van de nodes.

Door middel van de Juju GUI kan de gebruiker een Big Data opstelling ontplooien. Hierbij heeft hij de keuze tussen een van de standaard bundels gebruiken of zelf zijn eigen opstelling maken door middel van de basiscomponenten. Basiscomponenten kunnen individuele Charms zijn of bundels van Charms.

De gebruiker kan ook zelf services schrijven, aanpassen en ontplooien. Dit door middel van de management machine. De management machine stelt zijn CLI beschikbaar via een webpagina. De gebruiker kan ook via ssh inloggen op de management machine en via sftp zelfgeschreven Charms op de management machine zetten.

## 5.4 Samenvatting

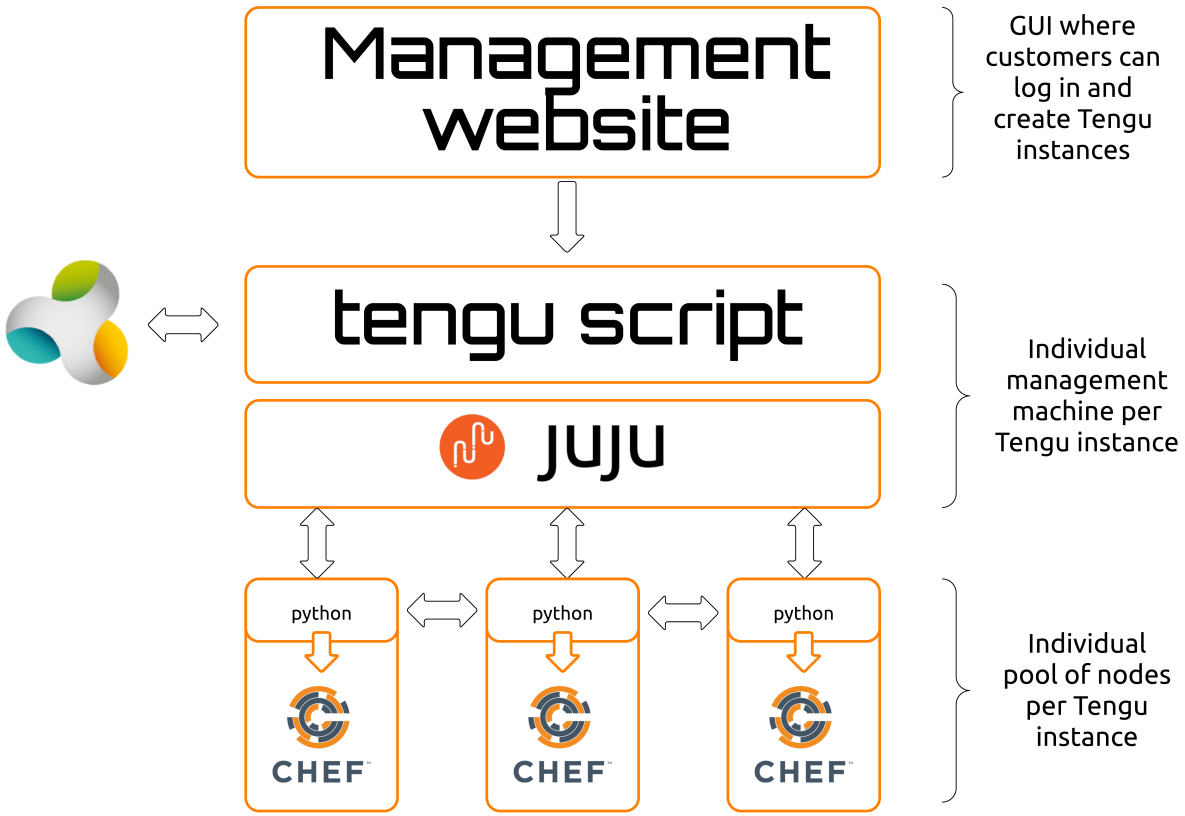
Hierbij een overzicht van de vereiste functionaliteit en hun oplossing:

### 5.4.1 Infrastructure as code

Juju Charms maken het mogelijk om installatie en configuratie van services te automatiseren. Chef cookbooks kunnen de installatie en configuratie van individuele servers automatiseren. Door de abstractie van Chef en de modulariteit van Juju krijg je een codebase die goed onderhoudbaar en gemakkelijk uitbreidbaar is.

### 5.4.2 Ontplooien applicaties

Deze functionaliteit kan verkregen worden door middel van subordenate Charms. Er is voor deze masterproef gekozen om dit niet verder uit te werken aangezien dit meer in de richting van Software as a Service gaat.




**Figuur 5.5:** Overzicht van Tengu architectuur

### 5.4.3 Modulariteit

Door de flexibele interfaces tussen Charms kan een volledig platform opgebouwd worden uit verschillende services. Het is ook mogelijk om in dat platform services te vervangen door vergelijkbare services.

### 5.4.4 Schaalbaarheid

Door de focus van Juju op services in plaats van individuele nodes, is het schalen van een service kinderspel. Een service die bestaat uit een cluster wordt op dezelfde manier aangesproken als een service die op één individuele node staat.

### 5.4.5 Gebruiksgemak

Juju heeft een zeer gebruiksvriendelijke interface. Bovendien maakt Juju slim gebruik van abstractie door te focussen op services en niet op servers.

### 5.4.6 Multitenancy

Meerdere klanten worden ondersteund door iedere klant zijn eigen Tengu instantie te geven. Klanten kunnen zelf ook meerdere Tengu instanties hebben. Meerdere rollen in dezelfde Tengu instantie is in deze architectuur niet mogelijk doordat deze functionaliteit nog niet ondersteund wordt door Juju. Deze functionaliteit staat wel op de roadmap om geïmplementeerd te worden in een van de volgende releases van Juju.

### 5.4.7 Stabiliteit van Services

In Hoofdstuk 4.8 werden de verschillende oorzaken van het uitvallen besproken. Hier een kort overzicht van wat deze architectuur kan doen tegen iedere oorzaak.

1. **Uitvallen nodes:** Juju monitort de connectiviteit naar de verschillende nodes en laat het weten als een node niet meer bereikbaar is. Het PaaS kan hier dan op inspelen en een nieuwe node met dezelfde functionaliteit installeren. Dit deel wordt niet verder uitgewerkt in deze masterproef aangezien jFed nog geen interface openstelt om nodes toe te voegen aan een experiment.

2. **Proces Crasht:** Chef biedt een eenvoudige interface aan om processen als service te definiëren waarbij services moeten heropstarten na een crash.

3. **Configuratie aangepast:** Chef controleert periodiek de configuratie. Alle wijzigingen aan de configuratie die niet via Chef gebeuren worden dus teniet gedaan.

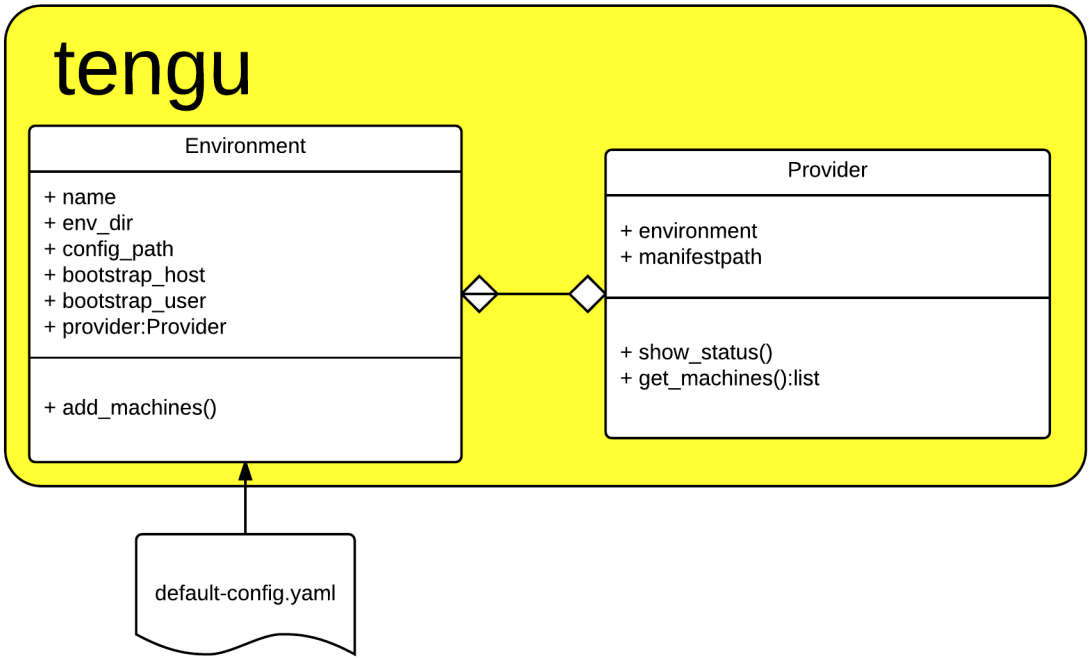
4. **Fouten op applicatieniveau:** Deze fouten kunnen opgevangen worden door een monitoring systeem. Er bestaan al verschillende Charms voor monitoringsystemen zoals nagios en collectd. De mogelijkheden hiervan passen niet in de scope van deze masterproef. Dit zal niet verder besproken worden.

# Hoofdstuk 6

# Implementatie

## 6.1 Het tengu script

Het `tengu` script automatiseert de installatie van een Tengu instantie op Ubuntu 14.04.




**Figuur 6.1:** Het tengu script

### 6.1.1 Functionaliteit

#### config

```
tengu config
```

Dit commando voert de configuratiewizard uit. De nodige configuratieparameters worden gevraagd aan de gebruiker. De gebruiker heeft ook de optie om de standaardinstellingen voor die parameter te gebruiken. De gevraagde parameters zijn:

- **bootstrap-user**: de jFed/Emulab account
- **project-name**: naam van het Emulab project
- **.PEM file location**: het private sleutelbestand om in te loggen
- **.PEM file password**: het wachtwoord om het sleutelbestand te ontgrendelen

#### create-environment

```
tengu create-environment <fqdn>
tengu create-environment <environment-name>
```

Een tengu environment kan op twee manieren aangemaakt worden. Ofwel wordt de environment gemaakt in een bestaand jFed experiment, dan wordt de fqdn van een van de servers in dat experiment meegegeven. Ofwel wordt er een nieuw jFed experiment gemaakt, dan wordt de naam van het nieuwe experiment meegegeven.

Beide versies van het commando installeren dan Juju. Alle nodes in het jFed experiment worden toegevoegd aan Juju, en juju-gui wordt geïnstalleerd op de bootstrap-host. Hierna kan je zelf via de `juju-gui` charms toevoegen of een default Tengu opstelling ontplooien aan de hand van de `default-tengu` bundle.

#### destroy-environment

```
tengu destroy-environment <environment-name>
tengu destroy-environment <environment-name> --complete
```

`destroy-environment` verwijdert het Juju environment en verwijdert alle configuratiebestanden. Dit gebeurt op een robuuste manier zodat indien er iets misloopt, je steeds kan herbeginnen na `destroy-environment`. Indien de vlag `--complete` meegegeven wordt, dan wordt ook het achterliggend jFed experiment verwijderd.

**status**

```
tengu
tengu status
```

Dit commando toont een overzicht van de verschillende Juju machines, Jfed nodes, Charms en beheerdersconsoles. De huidige tengu environment hangt samen met de huidige Juju instantie. Door middel van het commando `juju switch` kan je wisselen tussen instanties.

### 6.1.2 Configuratie

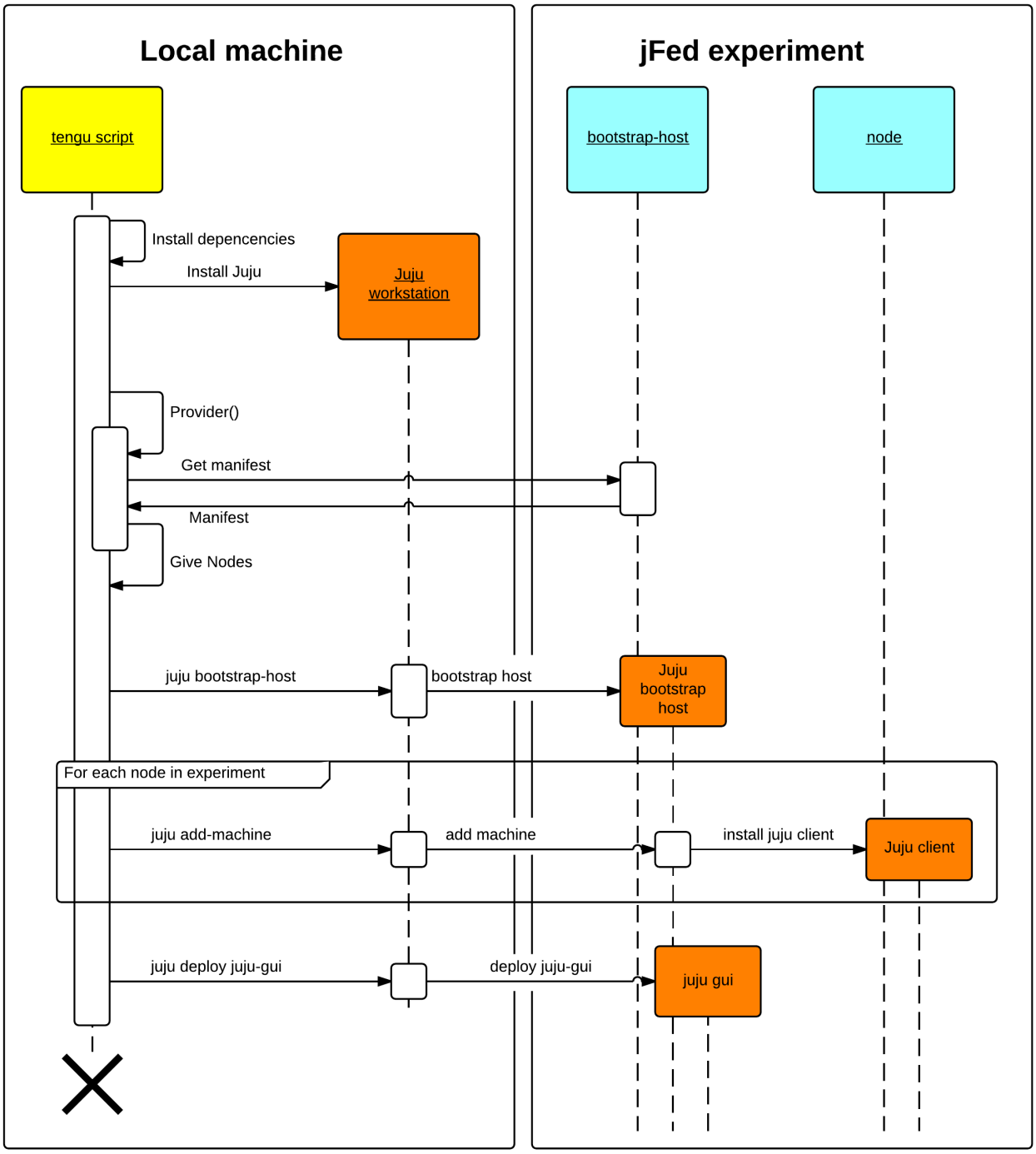
Het `default-config.yaml` configuratiebestand ondersteunt volgende parameters:

- **bootstrap-user**: de gebruikersnaam om in te loggen op de nodes
- **http-proxy**: de http proxy die gebruikt wordt voor de nodes
- **no-proxy**: voor welke hosts de http proxy niet gebruikt moet worden

### 6.1.3 Werking

Het sequentiediagram van Figuur 6.2 toont hoe de installatie in zijn werk gaat. Eerst worden lokaal Juju en de nodige dependencies geïnstalleerd. Dan wordt er verbinding gemaakt met de `bootstrap-host` om de manifest af te halen. Uit de manifest worden alle nodes van het jFed experiment gehaald.

De `bootstrap-host` wordt dan gebootstrappet. Dan worden alle andere nodes in het jFed experiment toegevoegd aan de Juju environment. Als laatste wordt de `juju-gui` Charm ontplooid op de `bootstrap-host` en sluit het `tengu` script af.




**Figuur 6.2:** Sequentiediagram tengu install

## 6.2 Juju Charm Store Charms

In de Juju Charm Store zit voor bijna alle componenten van Tengu een Charm. De enige component die niet aanwezig is, is de WSO2 ESB. Voor de meeste componenten moet er een keuze gemaakt worden: wordt de Charm uit de Store gebruikt of is het beter om zelf een Charm te ontwikkelen. In dit hoofdstuk worden de bevindingen besproken na het werken met de verschillende Charms uit de Charm store.

### 6.2.1 Voordelen van Charms uit Store

Door te werken met Charms uit de Charm store gaat het ontwikkelingsproces sneller, aangezien er minder code geschreven moet worden. Werken met Charms uit de Store heeft echter nog andere voordelen. Deze worden verder besproken in dit hoofdstuk.

#### Meegenieten van wijzigingen van andere ontwikkelaars

Juju evolueert snel en krijgt constant nieuwe functionaliteit. Charms moeten echter aangepast worden om deze functionaliteit te gebruiken. De Charmers gemeenschap implementeert deze nieuwe functionaliteit in Store Charms. Dit haalt veel last weg van Tengu ontwikkelaars[1].

*De actions interface is een functionaliteit die recent toegevoegd is aan Juju. Actions kunnen gebruikt worden om manueel handelingen uit te voeren op een Charm. In het kader van Tengu kan dit gebruikt worden om databases te back-uppen of de iteratieve batch berekening te starten en te stoppen.*

#### Relaties naar nieuwe Charms

Er komen dagelijks nieuwe Charms in de Charms store. Deze nieuwe Charms hebben soms relaties met Charms die in Tengu gebruikt worden. Het implementeren en onderhouden van deze relaties is ook iets dat de Charmers gemeenschap doet.

*Pig is een module bovenop Hadoop. De Pig Charm heeft dus een relatie met de Hadoop Charm. Indien een gebruiker van Tengu wenst te werken met Pig, dan moet deze functionaliteit niet meer geïmplementeerd worden.*

[1] Aangezien Tengu een open source technologie is, kan deze last wel gedragen worden door de Tengu community

#### Onderhouden door producent

Charms die gemaakt worden in samenwerking met de producent van het product worden ook door die producent onderhouden. Dit verzekert dat de Charm alle aanbevelingen van de producent volgt.

*Hortonworks onderhoudt zelf Charms voor hun versie van Hadoop, IBM heeft een team dat charms maakt voor al hun producten, Google onderhoudt de Charms voor Kubernetes en de collectie Openstack Charms wordt gemaakt in samenwerking met de openstack ontwikkelaars. Ook relevant voor Tengu is het Big Data Development Team, dat producten zoals Apache Hadoop, Hive en Pig onderhoudt.*

#### Continue testen

Charms uit de Store worden continu getest. Canonical host een *test runner* die alle Charms automatisch test. Dit verzekert langs de ene kant dat nieuwe versies van Juju achterwaarts compatibel zijn met oude Charms. Dit verzekert ook dat een nieuwe versie van een Charm oude relaties met andere Charms niet breekt.

### 6.2.2 Nadelen

Het is echter belangrijk om niet te vergeten dat er ook nadelen vasthangen aan het gebruik van Charm Store Charms.

#### Vereisen aanpassingen

Een aantal Charms uit de Store vereisen kleine aanpassingen voor ze gebruikt kunnen worden in Tengu. De aanpassingen voor de Hadoop Charm werden besproken in Sectie 7.1. Daarbovenop komt dat Store Charms geschreven zijn in een breed scala van talen, van Bash tot Python en Ansible. Dit maakt het implementeren van kleine aanpassingen complex, zeker indien de programmeur niet gewend is aan de taal.

#### Aanpassingen moeten aanvaard worden

Elke aanpassing in een Store Charm moet aanvaard worden door het project zelf, de *upstream*. Aanpassingen die heel specifiek zijn voor Tengu zullen misschien niet aanvaard worden. Het is ook niet mogelijk om aanpassingen door te voeren die niet achterwaarts compatibel zijn.

### 6.2.3 Conclusie

De meeste Charms uit de Store vereisen enkel kleine aanpassingen. Deze aanpassingen zijn stuk voor stuk aanpassingen van algemeen nut. Aanpassingen waar iedereen voordeel bij heeft worden snel aanvaard bij de *upstream*.

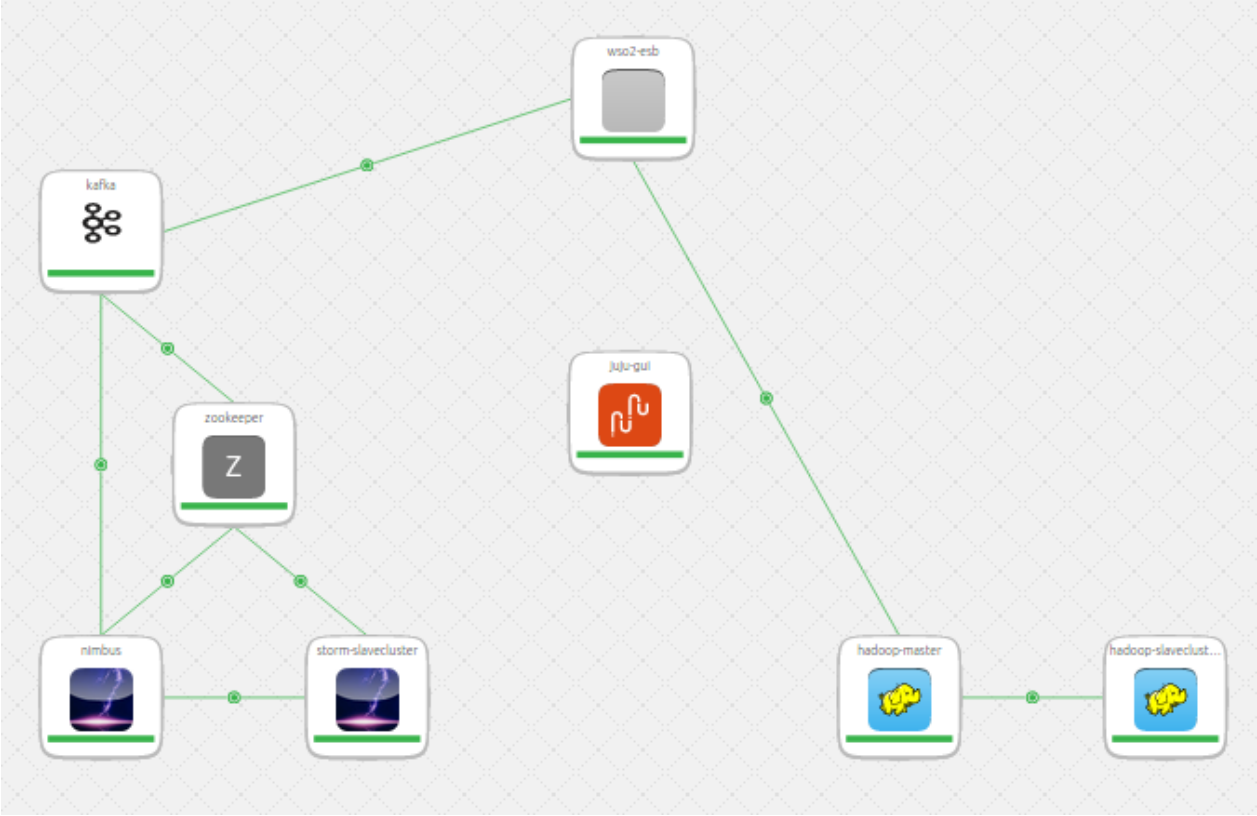
*Iedereen die de Hadoop Charm gebruikt, heeft er voordeel bij dat de services herstart worden na een crash. Het is dus zeker mogelijk om deze aanpassingen door te voeren in de upstream.*

Charms uit de Charm store zijn niet altijd volgens de regels van de kunst geprogrammeerd. Dit zorgt ervoor dat het aanpassen van Charms soms ongewenste effecten kan hebben. Het robuuste testsysteem zorgt er echter voor dat dit geen grote gevolgen kan hebben.

Voor deze masterproef wordt gekozen om Charms die beschikbaar zijn in de Store te gebruiken en aan te passen.

## 6.3 De default-tengu bundel

De default tengu bundel is een Juju Bundle om een standaard Tengu opstelling te ontplooien. Die bundel specifieert een aantal Charms samen met relaties tussen die Charms en configuratieparameters. Figuur 6.3 toont deze standaardopstelling.




**Figuur 6.3:** De standaard Tengu bundel

Juju Bundles hebben verschillende geavanceerde functies zoals de mogelijkheid om verschillende Charms te ontplooien op dezelfde node. Dit gebeurt aan de hand van de `to` parameter.

**Listing 6.1:** Deel van default Tengu bundel

```
kafka:
  charm: "local:trusty/kafka"
  num_units: 1
  options:
    requested_topics: toStorm toESB
  annotations:
    "gui-x": "400"
    "gui-y": "100"
```

```
zookeeper:
  charm: "cs:trusty/zookeeper"
  num_units: 1
  annotations:
    "gui-x": "600"
    "gui-y": "450"
  to: kafka=0
```

## 6.4 Kleine problemen

### 6.4.1 Http Proxy

Op de virtual wall moet gebruik gemaakt worden van een Http Proxy voor alle communicatie met het internet. Dit vraagt extra configuratie voor iedere applicatie apart. Een Http Proxy is niet doorzichtig. Iedere individuele applicatie moet zijn gedrag aanpassen. Dit is niet altijd even evident doordat niet alle applicaties een gestandaardiseerde manier gebruiken om de proxyinstellingen te configureren.

De meest gebruikte manier om proxyinstellingen te configureren zijn de `http_proxy` en de `no_proxy` environment variabelen.

1. `http_proxy` is de URL met poortnummer van de proxy
2. `no_proxy` duidt aan voor welke hosts de http proxy niet gebruikt moet worden. Dit is een string van IP adressen en domeinnamen gescheiden door een komma. Wildcards worden best niet gebruikt aangezien ze niet in alle applicaties ondersteund worden. Bij een domeinnaam geldt dit automatisch ook voor alle subdomeinen.

Sommige applicaties gebruiken de varianten in hoofdletters: `HTTP_PROXY` en de `NO_PROXY`. Door volgend script in `/etc/profile.d/` te zetten en opnieuw in te loggen stel je de variabelen in.

**Listing 6.2:** proxy functie

```
# Variables
PROXY=...
NOPROXY=...

echo "export http_proxy=$PROXY
```

```
export https_proxy=$PROXY
export no_proxy=$NOPROXY
export HTTP_PROXY=$PROXY
export HTTPS_PROXY=$PROXY
export NO_PROXY=$NOPROXY"
```

Deze variabelen worden echter niet doorgegeven aan processen uitgevoerd met `sudo`. Om dit wel te doen kan je `sudo` uitvoeren met de `-E` vlag of het configuratiebestand van `sudo` aanpassen.

**Listing 6.3:** proxy functie

```
Defaults env_keep += "http_proxy https_proxy no_proxy"
```

Dan zijn er nog verschillende programma's die zelf nog extra configuratie nodig hebben. Hieronder een voorbeeld voor de programma's `wget` en `apt`.

**Listing 6.4:** proxy functie

```
# Proxy settings
echo Acquire::http::Proxy\"$PROXY"\;|sudo tee /etc/apt/apt.conf
echo http_proxy = $PROXY | sudo tee -a /etc/wgetrc
echo https_proxy = $PROXY | sudo tee -a /etc/wgetrc
echo no_proxy = $PROXY | sudo tee -a /etc/wgetrc
```

### 6.4.2 Communicatie van Juju master naar nodes

Bij het bootstrappen van Juju en bij het toevoegen van nodes aan Juju wordt er een ssh verbinding opgezet. Gebruikers van de virtual wall ssh'en naar nodes door te authenticeren met hun private key. Gebruikersaccounts op de virtual wall hebben standaard geen wachtwoord. Indien Juju uitgevoerd wordt in een ssh sessie naar de server, dan gebruikt Juju ssh agent forwarding voor de communicatie naar de verschillende nodes. Hier zijn twee dingen belangrijk:

1. Juju heeft de environment variabele `SSH_AUTH_SOCK` nodig. Deze staat standaard ingesteld na login via ssh. Deze variabele wordt echter niet doorgegeven na commando's zoals `sudo` of `su -l`.

2. Dit werkt enkel indien het script uitgevoerd wordt door een gebruiker die zelf ingelogd is via ssh. Een bootscript zal dit dus niet kunnen gebruiken.

Voor deze masterproef werd dit opgelost door de private key die gebuikt wordt om te verbinden met de machines op de management machine waar Juju op draaide te zetten.

### 6.4.3 jFed gebruiker

In het `rspec` configuratiebestand is het mogelijk om bootscripts te definiëren. Deze scripts worden uitgevoerd als de node voor de eerste maal opstart. Scripts worden uitgevoerd onder de `geniuser` gebruiker. Het is aangewezen om van gebruiker te wisselen bij de start van een script.

### 6.4.4 Live vergroten root partitie

De Ubuntu 12.04 images op de virtual wall hebben een heel kleine root partitie. De root partitie kan tijdens het draaien van de server vergroot worden op volgende manier:

Eerst moet de fysieke partitie vergroot worden. De partitie wordt vergroot door er vrije schijfruimte na de partitie aan toe te voegen. Dit gebeurt door middel van de tool `fdisk`. De originele partitie moet verwijderd worden en een nieuwe partitie moet aangemaakt worden. De nieuwe partitie moet hetzelfde startblok hebben. Het is heel belangrijk dat de nieuwe partitie hetzelfde nummer en hetzelfde startblok heeft. Anders is het onmogelijk om nog op te starten van die partitie.

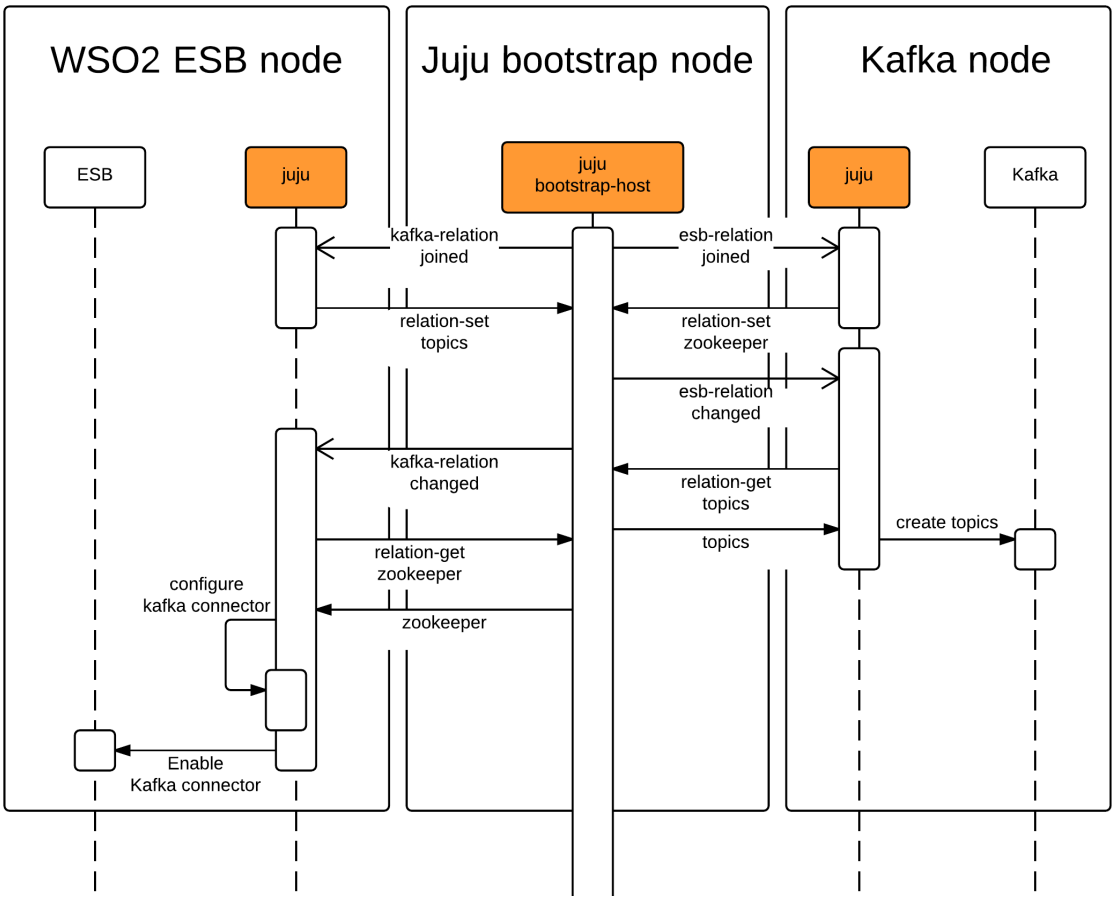
Daarna moet de kernel te weten komen dat de fysieke partitie vergroot is. De gemakkelijkste manier om dit te doen is door de server te herstarten. Na een server herstart is de logische partitie echter nog steeds zijn originele grootte. De logische partitie wordt vergroot door middel van het commando `resize2fs`. Na het uitvoeren van het commando zie je de logische partitie langzaam groter worden.

### 6.4.5 WSO2 ESB Kafka connector aanzetten

Om Kafka te connecteren met de WSO2 ESB moeten er verschillende dingen gebeuren:

- de nodige Kafka topics aanmaken,
- de kafka-esb connector installeren en configureren,
- en de kafka-esb connector activeren.

Het verloop van deze handelingen zie je in figuur 6.4.




**Figuur 6.4:** De relatie tussen Kafka en de ESB

De moeilijkheid is hier om de Kafka connector te activeren. De WSO2 ESB heeft geen commandline interface, alle communicatie moet gebeuren via een SOAP API. De oplossing kwam er dankzij een antwoord op stackoverflow[2]. In het volgend SOAP bericht specifieer je welke connector je wilt aanzetten.

**Listing 6.5:** SOAP bericht om Kafka connector aan te zetten

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
    xmlns:xsd="http://org.apache.synapse/xsd">
<soap:Header/>
<soap:Body>
    <xsd:updateStatus>
```

[2]stackoverflow.com/questions/28946865/is-it-possible-to-manage-wso2-esb-connectors-in-commandline

```
        <xsd:libQName>{org.wso2.carbon.connector}kafka</xsd:libQName>
        <xsd:libName>kafka</xsd:libName>
        <xsd:packageName>org.wso2.carbon.connector</xsd:packageName>
        <xsd:status>enabled</xsd:status>
    </xsd:updateStatus>
</soap:Body>
```

Door dit soap bericht naar de *MediationLibraryAdminService* te sturen wordt de connector aan gezet.

# Hoofdstuk 7

# Hadoop Charm use-case

In het vorige hoofdstuk werd duidelijk dat er voordelen zijn aan het gebruiken van Charms uit de Juju Charm store. De Store Charm van Hadoop implementeert echter niet alle functionaliteit die nodig is voor Tengu.

- De Charm installeert een oude versie van Hadoop.
- Er is geen interface om te verbinden met de WSO2 ESB.
- De processen van de service werden niet herstart na een crash.

Om deze problemen op te lossen werd de originele Store Charm, geschreven in Bash, aangepast. De functionaliteit van deze Charm is daarna ook op twee andere manieren geïmplementeerd. Dit hoofdstuk vergelijkt de voor- en nadelen van de drie implementaties van de Hadoop Charm.

## 7.1 Bash

De Store Charm van Hadoop is geïmplementeerd in puur Bash.

### 7.1.1 Aanpassingen

Om de Charm flexibeler te maken werd er een nieuwe configuratieoptie toegevoegd aan de Charm: de te installeren Hadoop versie. Dit werd gedaan door de code uit Listing 7.1 toe te voegen aan het bestand `config.yaml`.

**Listing 7.1:** config.yaml

```
options:
  hadoop-version:
    type: string
    default: 2.5.2
    description: |
      Version of hadoop to install.
      .
      * 2.5.2 one of the latest hadoop releases
```

Er werd code toegevoegd aan `hadoop-common` om deze versie uit te lezen en de juiste versie te installeren:

**Listing 7.2:** hadoop-common

```
HADOOP_VERSION=‘config-get hadoop-version‘
tar -xzf files/archives/hadoop-$HADOOP_VERSION.tar.gz -C $HADOOP_DIR
chown -R ubuntu:hadoop $HADOOP_DIR
```

Ook werd er een template interface gemaakt tussen Hadoop en de WSO2 ESB. Deze doet nog niets maar voorziet een raamwerk voor het uitwisselen van configuratiegegevens. Er werd gekozen om het herstarten van gecrashte services niet te implementeren in de bash Charm door de complexiteit van deze aanpassing.

### 7.1.2 Voordelen

#### Eenvoudig prototype

Bash Charms zijn zeer geschikt om een eenvoudig prototype te maken en Juju Charms te leren kennen.

Aangezien de Bash de standaard login shell is op de meeste Linux systemen, wordt Bash gebruikt voor de manuele installatie en configuratie van servers door systeembeheerders. Het maken van een Bash script dat de installatie en configuratie automatiseert is niets meer dan het bundelen van de gebruikte commando's.

Het omzetten van een Bash script naar een Juju Charm vereist maar één actie: het loskoppelen van relatie configuratie. De basisconfiguratie wordt vastgehangen aan de `install` hook. De configuratie van relaties wordt vastgehangen aan de respectievelijke `relation` hooks.

### 7.1.3 Nadelen

#### Geen idempotentie

Er is geen ingebouwde manier om idempotentie te verzekeren. Sommige commando's implementeren zelf idempotentie zoals `mkdir -p`. Bij andere commando's moet de idempotentie verzekerd worden door middel van conditionele structuren.

*Hoewel het `mkdir -p` commando idempotentie implementeert, wordt ook dit commando in de conditionele structuur gestoken doordat er geen eenvoudige manier is om te weten of het `tar` commando al uitgevoerd is. Er kan niet gecontroleerd worden of de inhoud van het archief al in de map staat, want de inhoud van het archief is niet op voorhand gekend.*

**Listing 7.3:** Conditionele structuur om idempotentie te verzekeren

```
if [ ! -d $HADOOP_DIR ]; then
    mkdir -p $HADOOP_DIR
    tar -xzf files/archives/hadoop-$HADOOP_VERSION.tar.gz -C $HADOOP_DIR
    chown -R ubuntu:hadoop $HADOOP_DIR
fi
```

#### Moeilijk beheer configuratiebestanden

Veel softwarepakketten worden geconfigureerd door configuratiebestanden aan te passen. Het beheer van deze applicaties vereist het wijzigen van XML, Json, Yaml of andere tekstbestanden of het invullen van templates. Bash heeft hier geen ingebouwde functies voor waardoor het wijzigen van dergelijke bestanden veel complexe code vereist die vatbaar is voor fouten.

*In de Hadoop Bash Charm worden de XML configuratiebestanden beheerd door `dotdee`. Dotdee geeft één configuratiebestand weer als een map met verschillende bestanden. Ieder bestand in de map representeert een regel in het configuratiebestand. Om een regel aan te passen, moet het betreffend bestand in de map aangepast worden. Dit is echter een zeer onnatuurlijke en complexe manier om bestanden te beheren. Dit verlaagt de leesbaarheid en de onderhoudbaarheid van de code.*

#### Weinig modulariteit

Bash heeft weinig mogelijkheden om modulaire programma's te schrijven. Code kan opgesplitst worden in verschillende bestanden, en Bash heeft een basis implementatie van functies.

Complexere technieken zoals Object Geörienteerd Programmeren wordt niet ondersteund. Daardoor worden grote Bash scripts snel onoverzichtelijk en onleesbaar. Een modulair en onderhoudbaar Bash programma schrijven vereist veel zelfbeheersing en kennis van de programmeur.

#### Gevoelige taal

Bash is een gevoelige taal met complexe bijwerkingen. Indien een spatie in een variabele of een padnaam niet op de juiste manier opgevangen wordt, dan is er kans op vreemde bijwerkingen. Ook dit vereist veel kennis van de programmeur en vergroot de kans op fouten.

#### Besturingssysteem afhankelijk

Bash scripts werken enkel op besturingssystemen die de Bash shell hebben. Verder zijn de bash scripts ook afhankelijk van de commando's en programma's die aangeroepen worden.

## 7.2 Python

De Hadoop Charm, origineel geschreven in Bash, werd in het kader van deze masterproef herschreven in Python. Dit met het idee dat Python veel van de nadelen van Bash zou oplossen.

### 7.2.1 Voordelen

#### Eenvoudig beheer configuratiebestanden

Python heeft veel ingebouwde modules om tekstbestanden te verwerken. Python kan heel goed met XML, yaml en json werken.

*Om JSON configuratiebestanden te maken volstaat het om een Python* `dictionary` *object aan de* `json.dump()` *functie mee te geven.*

**Listing 7.4:** JSON configuratiebestand schrijven

```
data = {'id': "1", 'address': "192.168.0.1"}
with open(filename, 'w') as outfile:
    json.dump(data, outfile)
```

### Modulariteit

In Python is het mogelijk om zowel objectgeoriënteerd als functioneel te programmeren. Het Services framework is een Python framework dat het mogelijk maakt om Charms en relaties op een abstracte manier te beschrijven. In het services framework bestaat een Charm uit enkele services die pas kunnen opstarten indien er bepaalde voorwaarden voldaan zijn.

*In de Hadoop Charm zijn Namenode, Datanode, Nodemanager en Resourcemanager aparte services. Om de Datanode service te starten moet er een relatie bestaan met de Namenode en moet de Namenode zijn ssh sleutel doorgegeven hebben. Pas dan zal de Datanode service opstarten. Het eigenlijke starten en stoppen van services gebeurt door het framework zelf. De programmeur moet enkel nog de juiste acties ondernemen om de service te installeren en te configureren.*

**Listing 7.5:** Declaratie van de Datanode service

```
{
    'service': 'datanode',
    'ports': [50010, 50020, 50075], # ports to open after start
    'provided_data': [
        HadoopSlaveSetRelation(name='datanode', interface='dfs')
    ],
    'required_data': [
        HadoopSlaveGetRelation(name='datanode', interface='dfs'),
        {'role' : 'datanode', 'command' : 'hadoop-daemon.sh'}
    ],
    'data_ready': [
        configure_datanode,
        helpers.render_template(
             source='upstart.conf',
             target='/etc/init/datanode.conf')
    ],
},
```

Listing 7.5 toont hoe een Service wordt gedefinieerd door een dictionary in te vullen met de juiste waarden. Hier een korte bespreking van mogelijke velden:

- De naam van de service wordt gespecifieerd in `service`. Deze moet gelijk zijn aan de naam van de upstart service die beheerd wordt.
- De poorten die de service gebruikt worden meegegeven als array in `ports`.

- De klasse toegekend aan `provided_data` geeft de data door aan de verbonden Charms.
- De klasse toegekend aan `required_data` wordt gebruikt om te kijken of aan alle vereisten voldaan is.
- Indien aan alle vereisten voldaan is, worden de acties beschreven in `data_ready` uitgevoerd. Hier wordt ook de upstart service aangemaakt vanaf een template. Deze template wordt ingevuld met de data die gespecifieerd was in `required_data`.

#### Besturingssysteemonafhankelijk

Python zelf is besturingssysteemonafhankelijk. Een Charm geschreven in Python is daardoor gemakkelijker over te zetten naar een ander besturingssysteem. Hier zijn echter nog enkele problemen. Dit wordt besproken bij de nadelen.

### 7.2.2 Nadelen

#### Geen ingebouwde idempotentie

Ook in Python is de idempotentie niet ingebouwd. Voor het uitvoeren van een commando moet er handmatig gecheckt worden of het niet al uitgevoerd is. Dit maakt code snel onleesbaar en vraagt veel extra denkwerk van de programmeur.

#### Laag abstractieniveau

Configuratiebeheer in Python gebeurt nog steeds op een zeer laag abstractieniveau, zeer dicht tegen het besturingssysteem aan. Hierbij het voorbeeld van het declareren van een service.

Hoewel het starten en stoppen van een service transparant gebeurt door het services framework, moet een service toch handmatig gedeclareerd worden. Om een service te declareren moet een upstart template ingevuld worden en op de juiste plaats gezet worden. Dit geeft enkele problemen. Langs de ene kant zorgt dit ervoor dat de programmeur **veel kennis** nodig heeft van het onderliggend *init systeem*, `upstart`. De programmeur moet zelf weten wat de syntax is van de configuratiebestanden en hoe complexe functionaliteit, zoals het opstarten van services als andere gebruiker of het herstarten van services na een crash, ingesteld moet worden. Een tweede probleem is dat deze configuratie zeer afhankelijk is van het besturingssysteem. Zelfs binnen één Linux distributie kunnen er verschillen zijn in hoe dergelijke zaken werken. De nieuwste LTS versie van Ubuntu op het moment van schrijven is 14.04. Ubuntu 14.04 gebruikt `upstart` als init systeem. De nieuwste non-lts versie op het

moment van schrijven is 15.04. Dit gebruikt `systemd` als init systeem. De volgende LTS release, 16.04, zal ook `systemd` gebruiken als init systeem. De template zal in de toekomst dus zeker herschreven moeten worden.

## 7.3 Chef

De werking van een Charm geschreven in Python en Chef is heel gelijkaardig. Ook bij een Charm geschreven in Chef wordt het Python Services framework gebruikt. De eigenlijke configuratieacties die ondernomen worden wanneer de vereiste data beschikbaar is, gebeurt echter door middel van Chef Cookbooks. Om de Chef Hadoop Charm te maken werden de bestaande Chef Cookbooks voor Hadoop gebruikt. Deze Cookbooks werden lichtjes aangepast om flexibeler in het Services framework te passen. De communicatie tussen Python en Chef gebeurt door middel van Databags zoals beschreven in hoofdstuk 5.2.2.

### 7.3.1 Voordelen

Bovenop de voordelen van Python, heeft een Chef Charm een extra voordeel.

#### Idempotentie

Chef heeft veel ingebouwde mechanismen om idempotentie te verzekeren. Er bestaan verschillende resources en providers die configuratietaken op zich nemen en die standaard op een idempotente manier uitvoeren.

*De `user` resource laat het toe om een gebruiker te configureren. In listing 7.6 worden eigenschappen van een gebruiker gedefinieerd. Het achterliggend framework zal kijken of de gebruiker deze eigenschappen heeft. Indien dit niet zo is, zal de gebruiker aangepast of aangemaakt worden.*

**Listing 7.6:** Configuratie van hduser

```
# Setup Hadoop user
user "hduser" do
  comment "Hadoop user"
  gid "hadoop"
  shell "/bin/bash"
  home "#{node['hadoop']['user_dir']}"
  password ""
```

```
  supports :manage_home => true
end
```

*Het is ook mogelijk om externe commando's op een idempotente manier uit te voeren. Het* `create` *argument van de* `execute` *resource zorgt ervoor dat het commando niet meer uitgevoerd wordt indien het meegegeven bestand of de meegegeven map aangemaakt is. Dit wordt bijvoorbeeld gebruikt voor het initiële formatteren van de Namenode.*

**Listing 7.7:** Formatteren van namenode

```
# Format NameNode
execute "Format Namenode" do
    user "hduser"
    group "hadoop"
    cwd "#{node['hadoop']['install_dir']}/current/bin"
    command "./hdfs namenode -format"
    creates "#{node['hadoop']['tmp_dir']}/dfs"
end
```

### 7.3.2 Nadelen

#### complexe architectuur

Chef wordt in de besproken architectuur enkel gebruikt als configuratiebeheerhulpmiddel hoewel Chef eigenlijk veel meer kan. De andere functionaliteit wordt in deze architectuur echter niet gebruikt aangezien Juju dit overneemt. Dit zorgt er echter voor dat Chef een enorm complexe architectuur is, waar maar een klein deel effectief van gebruikt wordt.

### 7.3.3 Platformafhankelijk

Een groot deel van de Chef Cookbooks in de Chef Supermarket zijn werken zowel op Windows als op Linux servers. Hierdoor wordt vaak aangenomen dat Chef Cookbooks standaard platformonafhankelijk zijn. Dit is echter niet zo. Volledige platformonafhankelijkheid is enkel te verkrijgen in Chef Cookbooks door middel van Conditionele constructies die voor ieder platform andere code uitvoeren.

*De* `service` *resource is gemaakt om een platformonafhankelijke manier te zijn om services te definiëren en te beheren in Chef. De* `service` *resource moet dus zelf kijken welk platform*

*en init systeem er gebruikt wordt. Door een bug in Chef*[1] *werkt dit echter niet op Ubuntu. Op Ubuntu moet er expliciet gezegd worden dat het* `upstart` *init systeem gebruikt wordt. Het definiëren van een upstart service wordt ook niet ondersteund waardoor het definiëren van een service uiteindelijk neerkomt op een template invullen en op de juiste plaats zetten, zoals te zien is in listing 7.8. Eerst wordt het doel en bronbestand gespecifieerd, samen met enkele eigenschappen van het doelbestand. Dan wordt in* `variables` *de template variabele gestoken. Deze variabelen zullen in de template vervangen worden. Ook dit template moet zelf gemaakt worden en is verschillend voor ieder init systeem. Dit is dus zeker niet platformonafhankelijk.*

**Listing 7.8:** Declaratie van de Namenode service

```
# Create service namenode
template "/etc/init/namenode.conf" do
  source "hadoop.upstart.conf.erb"
  owner "root"
  group "root"
  variables({
    :install_dir => node['hadoop']['install_dir']+"/current",
    :role => "namenode",
    :script => "hadoop-daemon.sh"
  })
end

service "namenode" do
  provider Chef::Provider::Service::Upstart
  supports :restart => true
  action [:enable,:start]
end
```

### Laag abstractieniveau

Chef resources en providers zijn gemaakt om het besturingssysteem weg te abstraheren. Op sommige plaatsen komt de onderliggende architectuur echter toch naar boven.

*Indien er een waarde uit een niet bestaande databag wordt opgehaald dan wordt er een fout opgeworpen. In de Chef Hadoop Charm wordt deze fout opgevangen, en wordt de standaardwaarde gebruikt. Welke exceptie er wordt opgeworpen hangt echter af van de Chef configuratie. Als Chef in client-server mode of als Chef-Zero draait, dan is de exceptie een*

[1] https://www.chef.io/blog/2014/09/18/chef-where-is-my-ubuntu-14-04-service-support/

*`HTTPServerException` met bericht 404, file not found. In het geval van Chef-Solo wordt de `InvalidDataBagPath` exceptie opgeworpen met het bericht dat een map niet gevonden werd. Dergelijke foutmeldingen en inconsistenties maken het schrijven en vooral het debuggen van een Cookbook moeilijk.*

**Listing 7.9:** Het halen van een waarde uit een databag

```
hdfs_address = begin
        data_bag_item('peers', 'hdfs')['address']
    rescue Net::HTTPServerException, Chef::Exceptions::InvalidDataBagPath
        node['ipaddress'] # default value
    end
```

## 7.4 Conclusie

**Bash** is een goede taal om snel een prototype van een Charm te schrijven. Charms geschreven in Bash zijn echter niet robuust en moeilijk te onderhouden.

Charms geschreven in **Python** zijn robuuster, en gemakkelijker te onderhouden, maar dit is nog steeds op een laag abstractieniveau, dicht tegen het besturingssysteem. De programmeur heeft veel kennis nodig over de onderliggende technologieën en moet zelf handmatig idempotentie implementeren.

**Chef** geeft de programmeur tools om eenvoudig idempotentie te verzekeren. Chef is echter een heel complexe technologie, en werkt voor veel zaken, zoals het declareren van services, nog steeds op een laag abstractieniveau.

Voor deze masterproef worden Charms die niet beschikbaar zijn in de Charm store geïmplementeerd door middel van Python en Chef. Het is naar de toekomst toe wel goed om na te gaan of er geen configuratiebeheer tool bestaat die de taak van Chef kan overnemen.

# Hoofdstuk 8

# Conclusie en verder werk

Het doel van deze masterproef was om een schaalbaar en modulair Big Data PaaS te maken op basis van Tengu. In Hoofdstuk 4 werd een functionele analyse gemaakt. Hoofdstuk 5 beschreef een architectuur die aan de eisen van de functionele analyse voldeed. Deze architectuur werd geïmplementeerd in Hoofdstuk 6, en Hoofdstuk 7 ging dieper in op de implementatie van de Hadoop Charm.

## 8.1 Conclusie

Tijdens de functionele analyse werd het duidelijk dat de traditionele policy based configuratiebeheer tools niet in staat zijn om de gewenste modulariteit te implementeren. Er zou een Service Orchestration tool nodig zijn, waarbij een Big Data opstelling kan opgebouwd worden door verschillende services met onderlinge relaties te ontplooien. De architectuur stelt Juju voor, waarbij services geëncapsuleerd worden als Charms. De combinatie van Juju als Service Orchestration tool en Chef als Configuration Management tool zorgt ervoor dat het platform op een modulaire en abstracte manier opgebouwd is. Door individuele services en hun verbindingen te encapsuleren in Juju Charms is het mogelijk dezelfde componenten te hergebruiken in verschillende Big Data opstellingen. Een Juju Bundle geeft de mogelijkheid om een standaard implementatie van de Lambda architectuur op te zetten. De gebruiker kan via de Juju GUI of de commandline interface services toevoegen, verwijderen en configureren om zo een Big Data opstelling op maat te maken. Doordat Juju services loskoppelt van nodes is het voor de gebruiker eenvoudig om een service die op één node draait te schalen naar een service die op een cluster van nodes draait.

Door de configuratie van individuele servers te doen door middel van Chef, is het eenvoudig om idempotente acties te verkrijgen, en kan een Tengu ontwikkelaar op een abstracte

manier configuratiecode schrijven. Tijdens het implementeren van de Hadoop Charm werd ook duidelijk dat de combinatie van Juju met een configuratiebeheer tool zoals Chef veel voordelen heeft. De grootste voordelen van Chef tegenover puur Python of Bash waren de ingebouwde functionaliteit om idempotentie te verzekeren, de robuustheid en de abstractie op veel plaatsen. Er kwamen wel enkele pijnpunten van Chef naar boven: het gebrek aan abstractie op sommige plaatsen en de complexiteit van Chef. Ook het implementeren van error recovery bij het crashen van processen was moeilijker dan verwacht. Daaruit werd de conclusie getrokken dat het positief kan zijn om te onderzoeken of er geen configuratiebeheer tool bestaat die nog meer aan de noden voldoet.

Na communicatie met Canonical, het bedrijf achter Juju, werd het duidelijk dat Juju een mooie toekomst heeft. Enkele van de grootste spelers uit de sector bouwen hun producten bovenop Juju, en ondersteuning voor het beheer van Windows en CentOS nodes zit momenteel in de ontwikkelingsfase. Het ontwikkelde platform voldoet aan bijna alle eisen. Waar het platform tekort schiet is op het vlak van multi-tenancy. Het huidige platform geeft iedere gebruiker een aparte Tengu instantie. Binnenin die instantie is er maar één gebruiker, deze gebruiker heeft volledige toegang tot alle nodes in die instantie. Aan de vereiste om in één instantie meerdere gebruikers met verschillende rollen te hebben is niet voldaan. Dit hoofdzakelijk door beperkingen in het onderliggend Juju platform. Het implementeren van rollen in Juju staat op de roadmap, maar er is nog geen datum bekend voor wanneer deze functionaliteit geïmplementeerd zal worden.[15]

De eigenlijke implementatie vereiste het samenwerken van JFed, Juju en de Virtual Wall. Hiervoor werd het `tengu` script ontwikkeld. Dit Python script maakt een nieuwe Tengu instantie aan door een JFed experiment aan te vragen, en een Juju environment op de verkregen nodes te ontplooien.

Buiten de multi-tenancy werd dus alle vereiste functionaliteit successvol geïmplementeerd.

## 8.2 Mogelijke uitbreidingen

### 8.2.1 Multi-tenancy

Zoals al eerder besproken ondersteunt één Tengu instantie maar één gebruiker met één rol. Aangezien multi-tenancy een belangrijke feature is, is het aangeraden om ondersteuning voor meerdere rollen te implementeren zodra Juju dit ondersteunt. Het definiëren van rollen zal verschillende mogelijkheden bieden om multi-tenant omgevingen te maken.

### 8.2.2 Automatische Testen

Het is aangeraden om automatische testen te schrijven om te verzekeren dat relaties tussen Charms blijven werken, ook na wijzigingen aan deze Charms. Juju biedt een framework aan om automatisch verschillende configuraties en bundels van Charms te ontplooien en te testen.

### 8.2.3 Gecentraliseerde logging en reporting

Iedere service genereert een hele hoop status- en werkingsinformatie. Ook Juju verzamelt een hoop statusinformatie. Sommige services, zoals Hadoop, hebben een eigen dashboard waar veel informatie op terug te vinden is. Het is echter aangewezen om alle informatie van alle services op een gecentraliseerde plaats op te slaan en beschikbaar te stellen. Deze centrale knowledgebase kan de basis vormen van een intelligent beheer van het volledig platform. Een van de mogelijke toepassingen is het opstellen van een belastingsprofiel om pieken te voorspellen en daar proactief op in te spelen.

### 8.2.4 Intelligentere relaties tussen Services

Relaties tussen Juju Charms kunnen gebruikt worden om complexe functionaliteit te implementeren. Een applicatie die op Storm draait zou kunnen specificeren welke informatie de ESB moet doorsturen naar Storm door middel van een transitieve relatie via Kafka.

### 8.2.5 Diepere integratie Juju met jFed

In de huidige versie gebeurt de communicatie tussen Juju en jFed via het `tengu` script. Een verbetering is om een Juju-naar-jFed connector te schrijven die het mogelijk maakt dat Juju zelf nodes aanvraagt aan jFed. Dat is een veel flexibelere oplossing. Dit was tijdens deze masterproef niet mogelijk omdat jFed de vereiste functionaliteit nog niet openstelde in zijn commandline interface.

### 8.2.6 Ontwikkelaarstools

Er zijn veel tools die enorm handig zijn voor het ontwikkelen van Charms. Indien gebruikers zelf Charms willen schrijven dan moeten ze de tools gebruiken op de management machine van hun Tengu instantie. Het zou beter zijn om alle nodige tools te bundelen in een Docker container zodat gebruikers deze ontwikkelingsomgeving ook lokaal kunnen draaien.

# Bibliografie


[1] Mark Ramm-Christensen. What's the future of Juju?, March 2015.

[2] Stephen Nelson-Smith. *Test-Driven Infrastructure with Chef*.

[3] Chef (software), February 2015. Page Version ID: 647597295.

[4] Nate Finch. What's the future of Juju?, March 2015.

[5] About Cookbooks — Chef Docs.

[6] Nathan Marz and James Warren. *Big Data: Principles and best practices of scalable realtime data systems*. Manning Publications, Place of publication not identified, 1 edition edition, May 2015.

[7] Douglas Laney. 3d data management: Controlling data volume, velocity and variety. *META Group Research Note*, 6, 2001.

[8] Introducing Chef Server - Getting started with Chef.

[9] John Allspaw and Jesse Robins. *Web Operations*.

[10] Mark Ramm-Christensen. What's the future of Juju?, March 2015.

[11] Alexis Bruemmer. What's the future of Juju?, March 2015.

[12] J. Cha, S.-Y. Chou, and J. Stjepandić. *Moving Integrated Product Development to Service Clouds in the Global Economy: Proceedings of the 21st ISPE Inc. International Conference on Concurrent Engineering, September 8–11, 2014*. IOS Press, September 2014.

[13] Nate Finch. What's the future of Juju?, March 2015.

[14] partitioning - How can I resize an ext root partition at runtime? - Ask Ubuntu.

[15] Mark Shuttleworth. Multi-user Juju Questions, March 2015.

[16] Andrea De Mauro, Marco Greco, and Michele Grimaldi. What is big data? A consensual definition and a review of key research topics. In *AIP Conference Proceedings*, volume 1644, pages 97–104. AIP Publishing, February 2015.

[17] T. Vanhove, J. Vandensteen, G. Van Seghbroeck, T. Wauters, and F. De Turck. Kameleo: Design of a new Platform-as-a-Service for flexible data management. In *2014 IEEE Network Operations and Management Symposium (NOMS)*, pages 1–4, May 2014.

[18] Mark Burgess. On the theory of system administration. *Science of Computer Programming*, 49(1–3):1–46, December 2003.

# Bijlage A

# Email conversatie: What's the future of Juju?

Gmail by Google

Merlijn Sebrechts <merlijn.sebrechts@gmail.com>

# What's the future of Juju?

17 berichten

**Merlijn Sebrechts** <merlijn.sebrechts@gmail.com> 25 maart 2015 21:01
Aan: juju@lists.ubuntu.com

Hi

I'm interested in what the future of Juju is. From the small experience I've had with it, it seems like a product with a lot of potential. It fills a gap in our project that no other technology can fill. Its biggest strength is how relations between services are managed. This is something that, to my knowledge, does not exist in any tool today. It enables a very modular approach and solves a lot of problems we would have with other tools.

However, I've also seen some things that worry me. Even after three years, there are still a lot of bugs in the project. The documentation is lacking, especially in the parts of Juju that are the most competitive. The community is also very small. The fact that it can still only manage Ubuntu servers worries me too. I could go more into detail here, but I don't think it is relevant to this question.

I'm considering starting a big long-term project on top of Juju. The project would be very dependent on Juju, so I don't want to do this if there is a chance that Juju will be abandoned in 5 years...

What can you tell me about the future of Juju? Things I'm interested in:

- Big companies building services on top of Juju

- Statements of long-term commitment from Canonical

- Usage statistics

- Statements of commitment to support other distro's

- .. or else, signs that Juju doesn't have a bright future.

Thanks

**Nate Finch** <nate.finch@canonical.com> 25 maart 2015 21:32
Aan: Merlijn Sebrechts <merlijn.sebrechts@gmail.com>
Cc: juju <juju@lists.ubuntu.com>

I'm a core dev on Juju, I can answer some, but not all of these questions.

First off, as far as long term commitment for Juju - Juju is a huge part of Canonical's long term strategy... right up there with the Ubuntu Phone and Ubuntu itself. The Juju team has been expanding hugely in the last couple years... I forget exactly the numbers we're at now, but it's an order of magnitude more people working on Juju than there were just a couple years ago.

Juju is used *extensively* internally at Canonical. We have a mandate that all internal services be deployed via Juju.

As far as supporting other operating systems, we actually do support Windows, right now (though it can be a little tricky to set up, and generally only works on private clouds, due to licensing restrictions on distributing Windows images). See here: http://www.cloudbase.it/windows-with-juju-and-maas/ (Cloudbase partnered

with us to get Juju working with Windows)

Cloudbase is also currently tackling CentOS support. It currently works and is just being cleaned up, it should be available for testing in a few weeks.

The number of features that have landed in the last year is tremendous - high availability, networking, storage, major improvements in the GUI, support for more clouds (Google Cloud Compute support is coming out with 1.23, which is due any day now), Windows support, backup and restore....

As for bugs, there are bugs in every product, especially new and rapidly expanding products, like Juju. If there are particular bugs that concern you, we'd be happy to look into them. We try to make sure that we fix anything that is a regression or would majorly hinder usage.... we do use this internally after all, so believe me, we hear about it when things aren't working well! :)

I'm sorry you find the documentation lacking. We have been putting effort into that recently. I, personally, am a big fan of extensive documentation, and I know our documentation is not nearly as extensive as it could be.

I can't personally talk about big companies using Juju... I know we have several very large companies doing very large installations, but I don't think anything is public about that. Hopefully someone else can bring up a list of people using Juju.

Hope that answers at least some of your questions.

[Tekst uit oorspronkelijke bericht is verborgen]

> [Tekst uit oorspronkelijke bericht is verborgen]
> --
> Juju mailing list
> Juju@lists.ubuntu.com
> Modify settings or unsubscribe at: https://lists.ubuntu.com/mailman/listinfo/juju

---

**Mark Ramm-Christensen (Canonical.com)** <mark.ramm-christensen@canonical.com> 25 maart 2015 21:33
Aan: Merlijn Sebrechts <merlijn.sebrechts@gmail.com>
Cc: "juju@lists.ubuntu.com" <juju@lists.ubuntu.com>

Well, I can provide a few things off the top of my head that should help.

- Canonical is fully committed to Juju as the way we deploy software internally, the way we deploy Open Stack clouds for our largest clients
- Windows workloads are supported in the current beta version of Juju, and should after a bit of real-world testing be fully supported in one of the next (bi-monthly) production ready releases.
- CentOS support is nearly feature complete, and should enter a beta release of Juju for testing within the next month. Like windows it will flow to a production release after it's had some real-world tests.

There are quite a few big companies working on juju charms. IBM for example is delivering quite a few charms and has committed multiple full time development resources to working with juju.

There are also quite a few other big names working on juju charms -- many of them in the OpenStack space. I'll get a list of folks who are already public about being part of our juju based openstack integration labs for you as soon as I can.

We also have some big plans for products built on top of juju. The first of which is the OpenStack Autopilot which automates the deployment, scale-out, and management of OpenStack clouds. But, we are also building more products on top of Juju right now, and it is core to our future plans in the cloud.

So, to make a long story short, I think juju is gaining traction with some big enterprise players, Canonical is fully committed to Juju, and we are seeing momentum pick up in the marketplace. So, I personally would definitely bet on a bright future for Juju.

--Mark Ramm

> On Wed, Mar 25, 2015 at 4:01 PM, Merlijn Sebrechts <merlijn.sebrechts@gmail.com> wrote:
>
> [Tekst uit oorspronkelijke bericht is verborgen]
> --
> Juju mailing list
> Juju@lists.ubuntu.com
> Modify settings or unsubscribe at: https://lists.ubuntu.com/mailman/listinfo/juju

---

**Merlijn Sebrechts** <merlijn.sebrechts@gmail.com> 25 maart 2015 22:24
Aan: Nate Finch <nate.finch@canonical.com>
Cc: juju <juju@lists.ubuntu.com>

Thanks for your answer! I didn't know Windows and Centos support was coming so soon, great to know!

The lacking documentation is the biggest issue to me. The charm-helpers documentation is outdated in a lot of places and that makes it seem as it isn't being actively maintained anymore. Ofcourse, this is a side-effect of a rapidly expanding product...
The charm-helpers documentation also lacks some good examples and "guidelines". Things like "What's the best way to create templates, What's the easiest way to get relation data, ..". The documentation shows you how to do it in bash, but is really lacking for python. I had a really hard time trying to decipher how the services framework works exactly. Then again, this is probably also partly due to the fact that I'm still learning my way around python.

As for the bugs. I submitted/"affects me" a few:
Critical:
https://bugs.launchpad.net/juju-deployer/+bug/1434458

Medium:
https://bugs.launchpad.net/charm-tools/+bug/1433035
https://bugs.launchpad.net/juju-core/+bug/1415176
https://bugs.launchpad.net/juju-core/+bug/1429790
https://bugs.launchpad.net/juju-core/+bug/1316174

Feature request:
https://bugs.launchpad.net/juju-core/+bug/1432331

The saltstack charm-helpers integration also has few problems. I just gave up on it and wrote the install hooks in python.

[Tekst uit oorspronkelijke bericht is verborgen]

---

**Michael Nelson** <michael.nelson@canonical.com> 25 maart 2015 22:39
Aan: Merlijn Sebrechts <merlijn.sebrechts@gmail.com>
Cc: Nate Finch <nate.finch@canonical.com>, juju <juju@lists.ubuntu.com>

> On Thu, Mar 26, 2015 at 8:24 AM, Merlijn Sebrechts
> <merlijn.sebrechts@gmail.com> wrote:
> > Thanks for your answer! I didn't know Windows and Centos support was coming
> > so soon, great to know!
> >
> > The lacking documentation is the biggest issue to me. The charm-helpers
> > documentation is outdated in a lot of places and that makes it seem as it
> > isn't being actively maintained anymore. Ofcourse, this is a side-effect of
> > a rapidly expanding product...
> > The charm-helpers documentation also lacks some good examples and
> > "guidelines". Things like "What's the best way to create templates, What's
> > the easiest way to get relation data, ..". The documentation shows you how
> > to do it in bash, but is really lacking for python. I had a really hard time

> trying to decipher how the services framework works exactly.

The services framework is quite new... I've not yet had the chance to use it yet either.

> The saltstack charm-helpers integration also has few problems. I just gave
> up on it and wrote the install hooks in python.

... and yes, I'll try to get some time to reevaluate the saltstack charm-helpers intergration and see whether it should be fixed or removed. We (the team I work in) used it initially for some charms, but then migrated quite soon to use the ansible support which we're using now, but the next time we write a new charm, we'll evaluate the services framework instead.

So the issue with the salt support may be that there are no charms (that I'm aware of) using it and so it is not being integration-tested regularly (though I'm keen to check).

I'd recommend exactly what you're doing - starting your charm in plain python. As you develop the charm, you may start to understand the need for, and use the services framework (or other charm-helper-provided support).

Cheers,
-Michael
[Tekst uit oorspronkelijke bericht is verborgen]

---

**Merlijn Sebrechts** <merlijn.sebrechts@gmail.com> 25 maart 2015 23:00
Aan: Michael Nelson <michael.nelson@canonical.com>

Are there any reference charms that show best practices of how charms should be written in python and ansible? I'm very interested in how relations are handled there.
[Tekst uit oorspronkelijke bericht is verborgen]

---

**Michael Nelson** <michael.nelson@canonical.com> 25 maart 2015 23:08
Aan: Merlijn Sebrechts <merlijn.sebrechts@gmail.com>

The one public one I'm aware of is the elasticsearch charm which we wrote in our team - but it has some (unrelated) complexities:

https://demo.jujucharms.com/trusty/elasticsearch-8/

But I have written a few blog posts about my experience using ansible for authoring charms (reusable ansible roles etc.) here:

https://micknelson.wordpress.com/

You just tag a task with the hook name (or multiple relation names) to ensure it'll be included on that hook.

Let me know if you go down that route.
-Michael

On Thu, Mar 26, 2015 at 9:00 AM, Merlijn Sebrechts
[Tekst uit oorspronkelijke bericht is verborgen]

---

**Mark Ramm-Christensen (Canonical.com)** <mark.ramm-christensen@canonical.com> 26 maart 2015 15:35
Aan: Merlijn Sebrechts <merlijn.sebrechts@gmail.com>
Cc: "juju@lists.ubuntu.com" <juju@lists.ubuntu.com>

As promised partners making use of juju as part of the OpenStack Interoperability Lab (OIL) include:

- IBM
- Microsoft
- Intel
- AMD
- HP
- Juniper
- Lenovo
- Melanox
- Metaswitch
- OCP (Open Compute Project)
- SanDisk
- VMWare

There are actually quite a few more, but that should give you some confidence that we are building traction with many infrastructure providers of all kinds.

[Tekst uit oorspronkelijke bericht is verborgen]

---

**Cory Benfield** <Cory.Benfield@metaswitch.com> 26 maart 2015 16:40
Aan: "Mark Ramm-Christensen (Canonical.com)" <mark.ramm-christensen@canonical.com>, Merlijn Sebrechts <merlijn.sebrechts@gmail.com>
Cc: "juju@lists.ubuntu.com" <juju@lists.ubuntu.com>

On Thu, Mar 26, 2015 at 14:35:23, Mark Ramm-Christensen (Canonical.com) wrote:
> As promised partners making use of juju as part of the OpenStack
> Interoperability Lab (OIL) include:
>
> * IBM
> * Microsoft
> * Intel
> * AMD
> * HP
> * Juniper
> * Lenovo
> * Melanox
> * Metaswitch
> * OCP (Open Compute Project)
> * SanDisk
> * VMWare

Metaswitch's engagement with Juju goes beyond the OIL: all of our open source products have Juju charms available, many of which are already in the charm store with more in review. That said, we have had a lot of success working with Juju for OpenStack deployments: on a personal level I've found it the best tool in my toolbox for deploying large OpenStack clusters.

We're really happy with where Juju is and where it's going, and we're excited to be a part of that. I think there are a lot of reasons to be optimistic about Juju's future.

Cory

---

**Nate Finch** <nate.finch@canonical.com> 26 maart 2015 17:04
Aan: Merlijn Sebrechts <merlijn.sebrechts@gmail.com>
Cc: juju <juju@lists.ubuntu.com>

I wanted to specifically thank you for pointing out the bugs that affect you. It's a huge help in prioritizing what we work on.

[Tekst uit oorspronkelijke bericht is verborgen]

---

**Alexis Bruemmer** <alexis.bruemmer@canonical.com> 26 maart 2015 20:14
Aan: Nate Finch <nate.finch@canonical.com>
Cc: Merlijn Sebrechts <merlijn.sebrechts@gmail.com>, juju <juju@lists.ubuntu.com>

I second Nate's statement!
[Tekst uit oorspronkelijke bericht is verborgen]
--
Alexis Bruemmer
Juju Core Manager, Canonical Ltd.
(503) 686-5018
alexis.bruemmer@canonical.com

---

**Mark Shuttleworth** <mark@ubuntu.com> 27 maart 2015 15:58
Aan: Merlijn Sebrechts <merlijn.sebrechts@gmail.com>, juju@lists.ubuntu.com

On 25/03/15 20:01, Merlijn Sebrechts wrote:
> I'm interested in what the future of Juju is. From the small experience
> I've had with it, it seems like a product with a lot of potential. It fills
> a gap in our project that no other technology can fill. Its biggest
> strength is how relations between services are managed. This is something
> that, to my knowledge, does not exist in any tool today. It enables a very
> modular approach and solves a lot of problems we would have with other
> tools.

Yes, agreed!

> However, I've also seen some things that worry me. Even after three years,
> there are still a lot of bugs in the project. The documentation is lacking,
> especially in the parts of Juju that are the most competitive. The
> community is also very small. The fact that it can still only manage Ubuntu
> servers worries me too. I could go more into detail here, but I don't think
> it is relevant to this question.

I understand what you mean. In part I think this comes from the team being focused on getting to our key Juju 2.0 milestone, and perhaps not taking as much time to celebrate the smaller milestones on the way, especially when they are things that we know need a little more work. For example, the Windows and CentOS support that has landed is a huge step, but we know it is still a bit rough and needs more polish. Perhaps we should be louder about some of those steps to draw in people who could help provide feedback and patches!

> I'm considering starting a big long-term project on top of Juju. The
> project would be very dependent on Juju, so I don't want to do this if
> there is a chance that Juju will be abandoned in 5 years...

No chance of that.

In my travels now I am glad to say I see a shift in the way people engage with us about Juju. It used to be "why would you want to compete with Puppet?" but now folks understand that we do not want to compete with Puppet, we want to come up a level and enable people to collaborate and reuse their puppet / chef / bash / python across different clouds and environments.

That focus on collaboration and reuse is what makes Juju special. In the long run we will integrate Juju into places like HEAT so you get the benefit of charms, together with the underlying cloud information about performance, so that things like autoscaling are magical, but in the short term it's easy to think the projects compete. But folks are starting to understand that now, so I don't get asked "why do you not just do HEAT?".

I also think that it will emerge that there are many levels of orchestration; Juju will be the BEST for the lowest level of infrastructure orchestration, but people will use Juju to deploy things like PAAS which themselves offer a kind of orchestration. So end-users

might use a PAAS, which is like a model or API or orchestration system, and underneath that, administrators will use Juju for the base level. We won't try to push Juju into every layer; just like we have kept MAAS and Juju separate with different interests and different responsibilities, so now the Chef guys are happy to use MAAS, which is good for us both.

> What can you tell me about the future of Juju? Things I'm interested in:
>
> - Big companies building services on top of Juju

Pretty much the biggest telco's in the world, and many of the companies that supply them, are writing charms.

Pretty much the biggest investment banks in the world, are writing charms.

Pretty much the biggest media companies in the world, are writing charms.

Pretty much the biggest big data and machine learning companies, are writing charms.

Now, of course, those companies also do a lot of OTHER things :) so I can't say they will all move solely to Juju because they naturally will not. But smart folks are seeing what you see - the model is amazing. And if we are able to get the next round of model pieces in place - status, network, disk - then we'll be able to do some really incredible rapid deployment and service evolution things.

My biggest concern at the moment is that I think it's too hard to add interfaces to existing charms. I think about adding say a Nagios interface to an existing charm - that should be really easy, but I think we haven't optimised the whole charm development process very well, so I will host the lead charmers at my house next week for a mini sprint so they can show me what I'm missing and we can discuss how to make it much better.

Also, I think we need to really socialise the status work, because it's too easy for charms to fail mysteriously, and that makes Juju look bad. So automated testing of charms is going to get even more important.

> - Statements of long-term commitment from Canonical

You heard it from me. I'm personally interested in Juju and it has a future, for sure, under all the scenarios I can influence. I think the cloud world needs something like Juju that is cross-platform and cross-cloud, and I find the project personally challenging and interesting. Looking at the thread, you also heard from folks who are enjoying working on it :)

> - Usage statistics

Best indicator is probably to look at the usage stats on individual charms you care about.

> - Statements of commitment to support other distro's

Windows and CentOS is a pretty diverse list. RHEL and SUSE will be easy once CentOS is A-grade, and I expect we'll get contributions of that or perhaps a Canonical customer will ask us to do that work. We will certainly accept patches for those or other platforms, being a cross-platform service oriented architecture modelling system is the point of the project!

> - .. or else, signs that Juju doesn't have a bright future.

It's been a long road so far, as you noted. My take is that:

* modelling *real* systems is very hard compared to, say, docker containers, but much more valuable at the infrastructure level
* doing it across AWS, OpenStack, Azure, GCE and bare metal means it takes time for things to get complete
* writing charms has been too hard - poorly documented, poorly considered

That said, I don't see any other project stepping up at the baseline infrastructure level (the Docker command systems are interesting for internal business apps).

For example, the HP Triple-O project for OpenStack deployment just imploded, because it's hard to get this stuff right. By contrast, installing OpenStack with Juju and charms got even better this last three months, and when we have status in the model, it will be pretty much incredible. Any time you set out to model real systems, it's hard! And in that world, I think Juju has the best chance of success. That's why the telco's seem to be growing comfortable with it for their OpenStack and other network function management.

Mark

---

**Eric Snow** <eric.snow@canonical.com> 27 maart 2015 20:50
Aan: Mark Shuttleworth <mark@ubuntu.com>
Cc: Merlijn Sebrechts <merlijn.sebrechts@gmail.com>, juju <juju@lists.ubuntu.com>

> On Fri, Mar 27, 2015 at 8:58 AM, Mark Shuttleworth <mark@ubuntu.com> wrote:
> In my travels now I am glad to say I see a shift in the way people
> engage with us about Juju. It used to be "why would you want to compete
> with Puppet?" but now folks understand that we do not want to compete
> with Puppet, we want to come up a level and enable people to collaborate
> and reuse their puppet / chef / bash / python across different clouds
> and environments.
>
> That focus on collaboration and reuse is what makes Juju special. In the
> long run we will integrate Juju into places like HEAT so you get the
> benefit of charms, together with the underlying cloud information about
> performance, so that things like autoscaling are magical, but in the
> short term it's easy to think the projects compete. But folks are
> starting to understand that now, so I don't get asked "why do you not
> just do HEAT?".
>
> I also think that it will emerge that there are many levels of
> orchestration; Juju will be the BEST for the lowest level of
> infrastructure orchestration, but people will use Juju to deploy things
> like PAAS which themselves offer a kind of orchestration. So end-users
> might use a PAAS, which is like a model or API or orchestration system,
> and underneath that, administrators will use Juju for the base level. We
> won't try to push Juju into every layer; just like we have kept MAAS and
> Juju separate with different interests and different responsibilities,
> so now the Chef guys are happy to use MAAS, which is good for us both.

Yay to all of this! I'm presenting a poster on juju at PyCon in a couple weeks and this is exactly the focus of it.

> Now, of course, those companies also do a lot of OTHER things :) so I
> can't say they will all move solely to Juju because they naturally will
> not. But smart folks are seeing what you see - the model is amazing. And
> if we are able to get the next round of model pieces in place - status,
> network, disk - then we'll be able to do some really incredible rapid
> deployment and service evolution things.

:)

> My biggest concern at the moment is that I think it's too hard to add
> interfaces to existing charms. I think about adding say a Nagios
> interface to an existing charm - that should be really easy, but I think
> we haven't optimised the whole charm development process very well, so I
> will host the lead charmers at my house next week for a mini sprint so
> they can show me what I'm missing and we can discuss how to make it much
> better.

I'm so glad you are doing this! In some ways making charm authoring easier/better is the lowest-hanging fruit for accelerating juju adoption. So this sort of effort has a big impact.

> Also, I think we need to really socialise the status work, because it's
> too easy for charms to fail mysteriously, and that makes Juju look bad.
> So automated testing of charms is going to get even more important.

+1

-eric

---

**Merlijn Sebrechts** <merlijn.sebrechts@gmail.com> 28 maart 2015 21:35
Aan: Mark Shuttleworth <mark@ubuntu.com>
Cc: juju <juju@lists.ubuntu.com>

Thanks to everyone for taking the time to answer this so thoroughly! The project is clearly going in a direction that will be very beneficial to us. The monitoring, multi-user and cross-platform features will come in very handy!

I do agree with Mark that "writing charms has been poorly documented". Writing charms in Bash is pretty well documented and has some good examples, but this is much less the case for python. Writing Charms in Bash is a great way to get started with Juju, but scripts get very complicated very fast. The python-charmhelpers modules seem to have some pretty strong features, but they aren't well documented. It seems like you brag about your "beginner features" but you keep quiet about what makes Juju charms really competitive...

This might contribute to the (incorrect) idea people have about what Juju is. A lot of people I speak to think Juju is "Chef/puppet for people who want a nice gui". Although this idea is slowly changing.. Talks like the one at Fosdem this year help spread the word about what Juju exactly is..
[Tekst uit oorspronkelijke bericht is verborgen]

---

**Nick Veitch** <nick.veitch@canonical.com> 28 maart 2015 23:32
Aan: Merlijn Sebrechts <merlijn.sebrechts@gmail.com>
Cc: juju <juju@lists.ubuntu.com>

Hi,
Thanks for the useful feedback on docs! I agree that good examples for Python and the charm helpers are things that are sorely missing, and we will address that soon.
I don't want to burden you further, but would you mind if I emailed you some questions about your experience with the documentation? I don't like to pass up the opportunity to talk to real users!
In the meantime if you have any more suggestions for docs, please feel free to email me or you can report specific issues on the docs repository (https://github.com/juju/docs/issues).

--
Nick Veitch, Documentation Team
nick.veitch@canonical.com

---

**Merlijn Sebrechts** <merlijn.sebrechts@gmail.com> 29 maart 2015 09:11

Aan: Nick Veitch <nick.veitch@canonical.com>
Cc: juju <juju@lists.ubuntu.com>

No problem, ask away!
[Tekst uit oorspronkelijke bericht is verborgen]

---

**Charles Butler** <charles.butler@canonical.com> 31 maart 2015 16:10
Aan: Merlijn Sebrechts <merlijn.sebrechts@gmail.com>
Cc: Mark Shuttleworth <mark@ubuntu.com>, juju <juju@lists.ubuntu.com>

Hey Merlijn,

I'm a bit late to this thread, but wanted to thank you for calling out the FOSDEM talk this year :) That was me on stage giving the overview.

Really glad you took the time to reach out and get some answers. If there's anything else we can do for you feel free to drop by #juju on irc.freenode.net - i'm lazypower.

All the best,

Charles Butler <charles.butler@canonical.com> - Juju Charmer
Come see the future of datacenter orchestration: http://jujucharms.com

[Tekst uit oorspronkelijke bericht is verborgen]

> [Tekst uit oorspronkelijke bericht is verborgen]
> --
> Juju mailing list
> Juju@lists.ubuntu.com
> Modify settings or unsubscribe at: https://lists.ubuntu.com/mailman/listinfo/juju

# Bijlage B

# Workflow

Juju heeft veel verschillende tools die het ontwikkelen van Charms eenvoudig maken. In dit hoofdstuk wordt de workflow besproken die werd gebruikt bij het ontwikkelen van Charms voor deze masterproef.

## B.1 Omgeving

Voor deze masterproef werd gewerkt op een Ubuntu 14.04 laptop. Sublime Text werd gebruikt als IDE. De PyFlakes SublimeLinter plugin spoort fouten in je code op. Volgend artikel raadt ook Anaconda aan. Deze plugin is echter heel onstabiel. [1]

**Listing B.1:** PyFlakes installeren

```
sudo pip install pyflakes

install SublimeLinter
install SublimeLinter-pyflakes
```

Bij het uitvoeren van `tengu install ...` worden alle vereiste softwarepakketten voor tengu geïnstalleerd.

## B.2 Ontwikkeling

Na het uitvoeren van `tengu install ...` zijn alle juju tools geïnstalleerd. Je deployt een Charm met het commando `juju deploy <name>`. Indien je enkel de naam meegeeft, bijvoor-

[1] https://realpython.com/blog/python/setting-up-sublime-text-3-for-full-stack-python-development/

beeld `hadoop`, dan wordt de Charm uit de Charm Store gedeployd. Wil je een lokale Charm deployen dan moet je dit expliciet aangeven: `juju deploy local:hadoop`.

Als een hook bij het uitvoeren een foutmelding geeft, dan komt die node in een error staat terecht. Je kan de hook debuggen door middel van het commando `juju debug-hooks hadoop/0`. Dit commando opent een tmux sessie op de aangegeven node. Door op je lokale machine `juju resolved --retry hadoop/0` uit te voeren, wordt de hook opnieuw uitgevoerd.

In de tmux sessie zal nu de hookenvironment opgestart worden. Nu kan je manueel de hook opnieuw uitvoeren door het bijhorende script uit te voeren. Je kan nu de fout zoeken, het script aanpassen en het script opnieuw testen. Let op! Wijzigingen die je uitvoert aan het script in deze tmux sessie worden niet doorgevoerd naar de repository op jouw lokale machine. Na het oplossen van de fout moet je deze wijzigingen ook lokaal doorvoeren.