KU LEUVEN

FACULTEIT INDUSTRIELE
INGENIEURSWETENSCHAPPEN

TECHNOLOGIECAMPUS GENT

Ontwerp en ontwikkeling van een testraamwerk installer

Pieter-Jan ROBRECHT

Promotor: dr. ir. Annemie Vorstermans

Co-promotor: Carl Eeckhout (Televic)

Masterproef ingediend tot het behalen van de graad van master of Science in de industriële wetenschappen: master of Science in de industriële wetenschappen ICT advanced communication technologies

Academiejaar 2016 - 2017

Dankwoord

Dank aan een ander

Abstract

Televic Rail ontwikkelde een Python testraamwerk voor het testen van verschillende producten. De ontwikkelde software, die op verschillende platformen moet draaien, gebruikt verschillende drivers en bibliotheken. Om producten correct te kunnen testen, wordt het raamwerk vaak geüpdatet, bijvoorbeeld bij het uitbrengen van een nieuwe driver, bibliotheek of om nieuwe producten te ondersteunen. Het installatieproces is tijdrovend, foutgevoelig en wordt best geautomatiseerd. Door dit proces te automatiseren wordt het mogelijk om informatie over het installatie- en updateproces te verzamelen. Het doel van de thesis is het uitvoeren van onderzoek naar een efficiënte oplossing en het ontwikkelen van een prototype. Dit prototype wordt onderverdeeld in drie componenten: een packager, een deployment server en een deployment omgeving. In een eerste fase wordt de packager ontworpen. Deze staat in voor het samenvoegen van de software componenten. Fase twee van de thesis bestaat uit het uitwerken van de deployment server. Met de server worden de verschillende installers verspreid en wordt er informatie verzameld over de deployment environments. Als laatste wordt dan de deployment environment behandeld. In deze geïsoleerde omgeving kan het installatie- en updateproces veilig gebeuren. Na een grondige evaluatie van een eerste basisprototype wordt het ontwerp eventueel aangepast. Het prototype wordt in een laatste fase uitgebreid zodat een rapportering over geïnstalleerde versies, deployment status, ... beschikbaar wordt voor het bedrijf.

Trefwoorden: Automatische – installer – testraamwerk - Python

Inhoudsopgave

Lijst van figuren

Hoofdstuk 1

Inleiding

1.1 Situering

Met meer dan 30 jaar ervaring in het ontwerpen en onderhouden van on-board communicatiesystemen is Televic Rail een toonaangevende producent van Passenger Information Systems, Entertainment Systems and Infotainment Systems. LiveCom is Televic Rail nieuwste generatie van informatie management systemen. Het integreert van alle aspecten van de on- en off-board reizigersinformatie, infotainment en entertainment. Het stelt operatoren in staan om hun volledige verkeersschema's, dienstregelingen, routes, stations en alles met betrekking tot informatie en infotainment omtrent passagiers te beheren, met behulp van off-board software tools.

iCoM, de geïntegreerde oplossing van Televic Rail voor passagiersgegevens en communicatiemanagement, biedt het openbaar vervoer en spoorwegondernemingen een centraal systeem voor het creëren, het beheren, het distribueren en het uitvoeren van real-time on en off-board generieke en commerciële passagiersinformatie op de vloot, in stations en bij haltes.

Naast deze systemen heeft Televic verschillende mechatronica-sensoren en veiligheidcontrolesystemen ontworpen. Alle systemen en apparaten zijn ontworpen in overeenstemming met de betreffende spoorwegsectornormen en voldoen aan de eisen voor passagiersruimte, draaistel en asmontage. On-board controllers verwerken sensordata-informatie en sturen deze naar de betreffende actuators en treinbeheersingssystemen. Fysische parameters die momenteel worden ondersteund zijn onder andere versnelling, druk, rotatie, temperatuur, geluid en de verplaatsing.

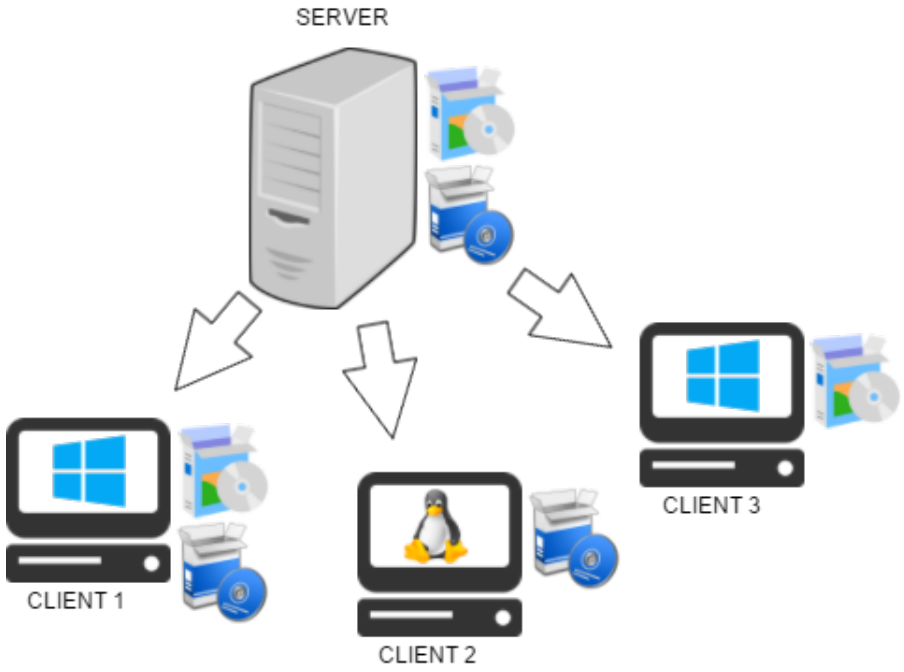
Om te voldoen aan de strenge veiligheidsnormen heeft Televic Rail een Python test framework ontworpen waarmee Televic in staat is om verschillende producten te onderwerpen aan verschillende testscenario's. Het framework werd ontworpen om gebruikt te worden op verschillende testtorens en werd later aangepast om bruikbaar te zijn op gewone computers. Dit framework wordt intensief gebruikt tijdens het productieproces en is cruciaal voor het afleveren van producten die voldoen aan de strenge veiligheidsnormen.

1.2 Probleemstelling

Het Python testraamwerk moet correct functioneren met een grote hoeveelheid aan producten die Televic fabriceert. Hierdoor gebruikt het Python testraamwerk verschillende drivers en bibliotheken. Een direct gevolg hiervan is dat het installatieproces tijdrovend en foutgevoelig is. Verder dient elk nieuw toestel op het raamwerk ondersteunt te worden waardoor er jaarlijks ettelijke re-

leases van het raamwerk verspreid worden. Het installatieproces en updateproces vragen om een vereenvoudiging zodat het testraamwerk gebruiksvriendelijker wordt.

Doordat het installatieproces en updateproces foutgevoelig zijn, moet een rampenplan voorzien worden. Door de aanwezigheid van een rampenplan, worden mogelijke fouten vermeden en indien nodig opgevangen en verholpen. Zo wordt vermeden dat de productielijn moet worden stilgelegd door bijvoorbeeld een fout tijdens het updaten van het testraamwerk. Het is dus belangrijk dat het testraamwerk op een eenvoudige manier geïnstalleerd en geüpdatet kan worden op verschillende testtorens. Om dit te laten slagen moet het testraamwerk eerst bij de verschillende testtorens geraken. Er zullen dus producenten van software zijn die de verschillende testraamwerken produceren en willen verspreiden maar ook gebruikers die deze willen ontvangen. Het proces waar bij software van een producent naar alle toepasbare gebruikers wordt verspreid en vervolgens geïnstalleerd wordt, wordt ook wel het deployment proces genoemd. Een kleine schets van dit idee is zichtbaar in Figuur 1.1.



Figuur 1.1: Voorstelling van software verspreiding

Tijdens het zoeken naar een oplossing voor het installatieproces en updateproces van het Python testraamwerk, moet gezocht worden naar een oplossing die ook antwoorden biedt op toekomstige problemen. Tot op heden werd ervan uitgegaan dat de enige applicatie die verspreid, geïnstalleerd en geüpdatet moet worden het Python testraamwerk is. In de toekomst moet het evenwel mogelijk zijn om verschillende applicaties van Televic te ondersteunen. Het is dan ook belangrijk dat een oplossing gevonden wordt die een groeiend aantal gebruikers ondersteunt. Naargelang het aantal applicaties die ondersteunt worden stijgt, zullen meer systemen gebruik maken van de applicatie. Naarmate het aantal gebruikers stijgt, stijgt ook de vraag naar een algemene administratieve interface. Verschillende versies van de applicaties zullen in omloop zijn waardoor de nood aan een administratieve interface stijgt. Met een administratie interface wordt het mogelijk om bij te houden hoe het uitrollen van een nieuwe versie van het testraamwerk verloopt maar naar de toekomst toe zou het mogelijk moeten zijn om verschillende gebruikers bij te staan. Deze informatie kan gebruikt

worden om het verspreidingsproces bij te sturen zodat een volgende keer het proces vlotter verloopt. Naast het ondersteunen van verschillende applicatie moet tijdens het ontwerpen rekening gehouden worden met verscheidene besturingssystemen. Huidige systemen draaien op Windows maar het gebruik van Linux is in de toekomst niet uit te sluiten.

Het doel van deze thesis is dan ook een oplossing te vinden voor het complexe installatieproces en updateproces. Daarbij moet de software eerst van de producent bij de gebruikers geraken en moeten deze processen rekening houden met een rampenplan om fouten tijdens een installatie of update op te vangen. Verder wordt tijdens de ontwerpfase van de applicatie rekening gehouden met een groeiend aantal applicaties die verspreid en geïnstalleerd moeten worden, een groeiend aantal gebruikers die deze applicaties willen ontvangen en de diversiteit van besturingssystemen van deze gebruikers.

1.3 Overzicht

Hoofdstuk 2

Literatuurstudie

2.1 Inleiding

In dit hoofdstuk, wordt een bespreking gegeven over alle mogelijke technieken, technologieën, architecturen, . . . die een oplossing kunnen bieden op het probleem dat in het vorige hoofdstuk besproken werd. Eerst wordt het deployment proces besproken en wordt nagegaan welke problemen hiermee geassocieerd worden. Vervolgens worden enkele case studies besproken om een beeld te krijgen van alle verschillende tools die aanwezig zijn en die gebruikt kunnen worden om het probleem omtrent het verspreiden van de software van de producent naar de gebruiker op te lossen. Na de cases studies worden ook enkele architecturen besproken die gebruikt kunnen worden om een applicatie te ontwerpen die om kan gaan met een groeiend aantal gebruikers. Hierna wordt nagegaan op welke manier een rampenplan geïmplementeerd kan worden zodanig dat problemen die ontstaan tijdens het deployment proces opgelost kunnen worden. Uit de probleembespreking is ook gebleken dat er nood is aan een applicatie die overweg kan met verschillende programma's die Televic gemaakt heeft of nog zal maken, één van deze programma's is dan het Python testraamwerk. Om te achterhalen wat mogelijk is om dit te realiseren, worden verscheidene technologieën besproken die hiervoor een oplossing kunnen bieden.

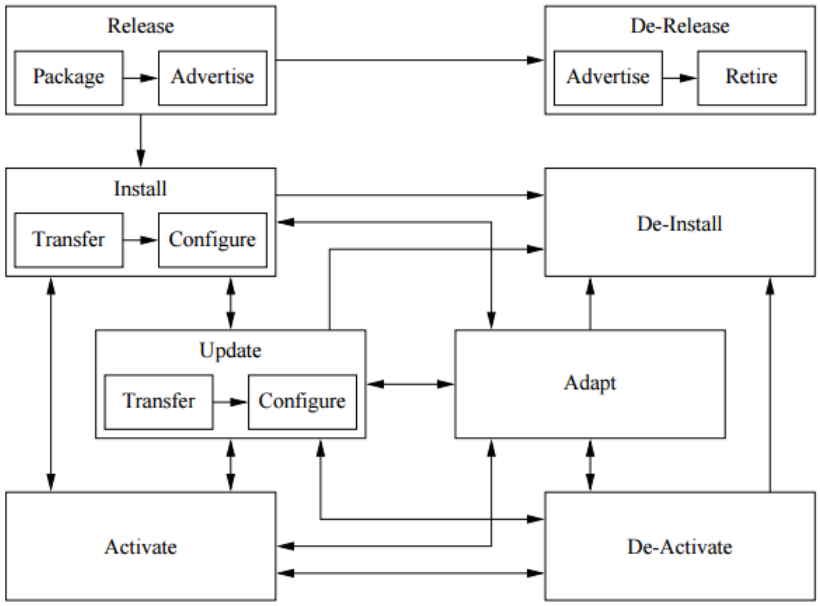
2.2 Software deployment

De levenscyclus van software deployment kan volgens Hall, Heimbigner en Wolf [1] en Dearie [2] beschreven worden in verschillend stappen, namelijk:

- *Release*: de software is volledig samengesteld uit pakketten die voldoende metadata bevatten om de verschillende bronnen te beschrijven waarvan het pakket afhangt.
- *Installatie*: de software moet overgebracht worden naar de client en geconfigureerd worden in voorbereiding op de activatie.
- *Activeren*: tijdens de activatie wordt de software-uitvoering opgestart of worden de nodige triggers geplaatst om de executie op het gepaste tijdstip op te starten.
- *Deactiveren*: dit is het tegengestelde van activeren. Deze stap is nodig zodat een aanpassing of herconfiguratie uitgevoerd kan worden.

- *Updaten*: dit is het proces waarin de software wordt aangepast. Deze stap wordt vaak geactiveerd door het uitbrengen van een nieuwe versie van de software.
- *Deïnstallatie*: tijdens deze stap zal de geïnstalleerd software van het clientsysteem gehaald worden.

De enige fase van de levenscyclus die zich uitsluitend op de server afgespeeld is de release fase. De rest van de fases spelen zich af op de verschillende clientsystemen.



Figuur 2.1: De levenscyclus van software [3]

In theorie zou het deployen van software een eenvoudige klus moeten zijn. Aangezien software bestaat uit een set van bestanden, zou het deployen van software naar een doelcomputer slechts bestaan uit het kopiëren van de nodige bestanden. Maar dit is vaak niet het geval. Volgens Dolstra [4] zijn er in de praktijk verschillende oorzaken die aan de basis liggen van een ingewikkeld deployment proces. Deze oorzaken kunnen in twee grote categorieën ingedeeld worden, namelijk de omgevings- en de onderhoudsproblemen.

Omgevingsproblemen In de eerste categorie ligt de nadruk vooral op correctheid. Voordat de software geïnstalleerd wordt op een doelsysteem, wordt de doelomgeving ondervraagt naar alle eigenschappen: zijn de nodige programma's aanwezig, bestaan alle configuratiebestanden, ... Als deze eisen niet voldaan zijn, dan zal de software niet werken zoals gewenst. Dolstra [4] haalt enkele concrete voorbeelden aan van omgevingsproblemen:

- De deployment van software kan een gedistribueerd probleem opleveren. Software kan afhankelijk zijn van componenten draaiende op externe systemen of van andere processen draaiende op het doelsysteem.
- Software is vaak afhankelijk van verschillende andere software componenten. Deze afhankelijkheden (dependencies) moeten voor de deployment bepaald worden. Dit proces is moeilijk en een fout kan pas laat ontdekt worden.

- De afhankelijkheden moeten compatibel zijn met wat de software verwacht. Bepaalde versies van een afhankelijkheid zijn compatibel met de software terwijl andere versies dit niet zijn. Dit komt doordat sommige depencies build-time variaties vertonen. At build-time worden bepaalde eigenschappen geselecteerd met als gevolg dat de dependency meer of minder functionaliteiten bevat ten opzichte van een vorige build.
- Sommige software-componenten zijn afhankelijk van specifieke hardware. Dit kan enkel verholpen worden door op voorhand te controleren welke hardware aanwezig is.

Uit deze concrete voorbeelden wordt al snel duidelijk dat er twee problemen zijn: de verschillende eisen van de software moeten geïdentificeerd worden en vervolgens moeten deze eisen voldaan worden in het doelsysteem.

Onderhoudsproblemen Naast de verschillende omgevingsproblemen, beschrijft Dolstra [4] ook enkele onderhoudsproblemen. Deze hebben te maken met het feit dat software moet kunnen "evolueren". Om dit te ondersteunen, moeten allerlei actie zoals upgraden en updaten uitgevoerd wordens. Enkele voorbeelden van zulke acties zijn:

- Tijdens het verwijderen van software moeten alle componenten verwijderd worden. Er mogen echter geen componenten verwijderd worden die nog in gebruik zijn door andere software.
- Ook tijdens het updaten van software moet rekening gehouden worden met andere software. Het updaten van een component kan voor problemen en failure zorgen in een andere component. Een DLL-hell wordt best ten alle tijden vermeden.
- Na het upgraden/updaten van een component, is het soms aangewezen om een rollback uit te voeren. Zo'n actie kan overwogen worden als na de upgrade belangrijke functionaliteiten van de software niet meer functioneren.

2.3 Case studies

Uit de vorige sectie blijkt dat het software deployment proces een uitgebreid en ingewikkeld proces is. Er moet rekening gehouden worden met verscheidene stappen die elk een eigen doel en functie hebben. Maar ook met verschillende problemen die kunnen optreden voor, na en tijdens deze stappen. Door de jaren heen zijn er verschillende technologieën ontwikkeld die het probleem van software deployment aanpakken. In wat volgt, worden dan ook enkele van deze technologieën besproken en wordt nagegaan op welke manier zij software deployment aanpakken. Op deze manier wordt een beeld gecreëerd van de mogelijke technologieën die gebruikt kunnen worden om de software van Televic van de producenten naar de gebruikers te krijgen.

2.3.1 Electric cloud

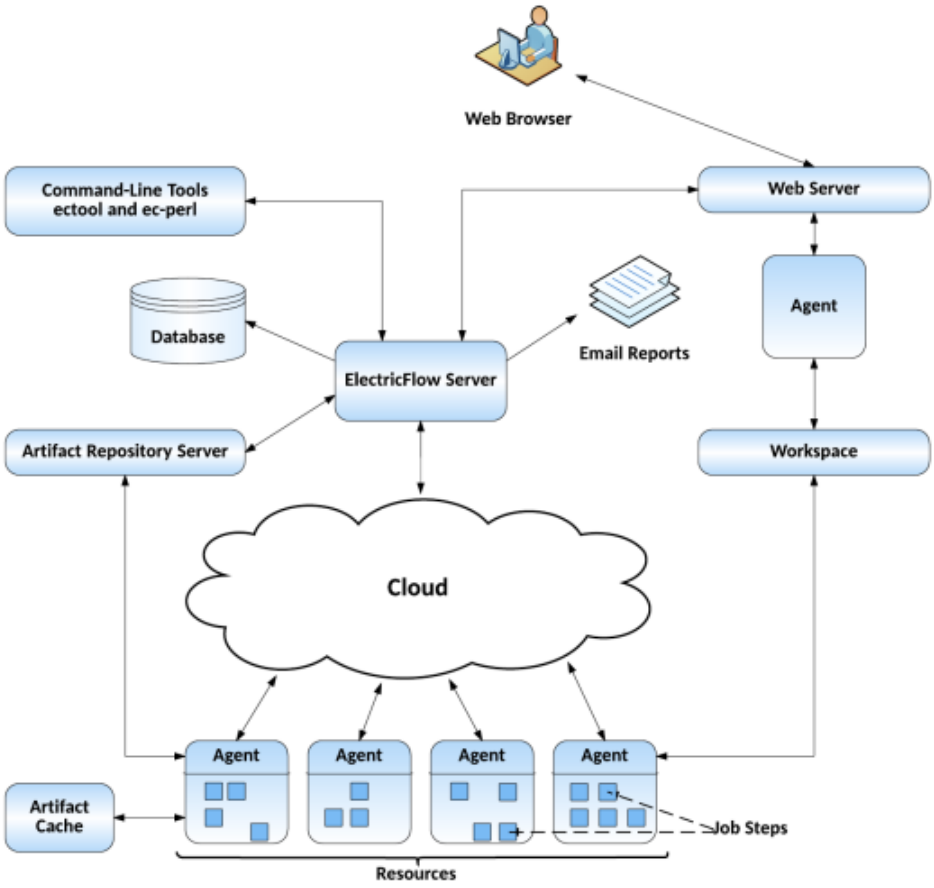
Electric cloud is een bedrijf opgericht in 2002 met een focus op application release automation (ARA). Hun product, ElectricFlow, is ontworpen om gebruikers in staat te stellen om het beschikbaar stellen, het bouwen en het verspreiden van mutlitiered applicaties te vereenvoudigen. Dit allemaal aan de hand van de model-driven architectuur [5]. Cloud [6] beschrijft ElectricFlow als een

enterprise-grade DevOps Release Automation platform. Met behulp van de model-driven aanpak is het mogelijk voor gebruikers om meerde pipelines en releases over verschillende infrastructuren te coördineren op een eenvoudige manier.

De kern van ElectricFlow bestaat uit een web-based systeem dat gebruikt wordt voor de automatie en het onderhouden van het bouw-, test-, deployment en releaseproces. Het automatieplatform bestaat uit een drie-lagen architectuur, een web interface en mogelijkheden om build en release analyses uit te voeren. De drie-lagen architectuur van ElectricFlow bestaat uit:

- **ElectricFlow Server**: een server die instaat voor het managen van resources, commando's en het generen van rapporten.
- **Databank**: een databank die instaat voor het opslaan van commando's, meta-data en logfiles.
- **Agents**: verschillende agenten die instaan voor het uitvoeren van commando's, het monitoren van statussen en het verzamelen van resultaten.

Figuur 2.2 geeft een architecturaal overzicht weer van ElectricFlow. In de figuur zijn de verschillende lagen zichtbaar samen verschillende andere tools die vervat zitten in het automatisatie platform [6].



Figuur 2.2: ElectricFlow Architectuur [6]

Om ElectricFlow te gebruiken voor bouw en test automatie, is het nodig om de volgende objecten te creëren, te configureren en bij te houden.

- **Project**: een project in ElectricFlow dient als container waarin procedures, stappen, workflows, . . . zitten. Op deze manier is het mogelijk om een scheiding te creëren tussen projecten die bijvoorbeeld een ander doel hebben.
- **Resource**: een resource wordt gedefinieerd als een agent machine waar stappen in uitgevoerd kunnen worden.
- **Procedures**: procedure en stappen worden gebruikt om taken in ElectricFlow te definiëren. Een procedure bestaat uit één of meerdere stappen waarbij in ieder stap een commando of script wordt uitgevoerd.
- **Workflow**: een workflow stelt de gebruiker in staat om build-test-deploy levenscycli te definiëren. Zo wordt het managen van verschillende procedures en stappen in een project eenvoudiger.

Met ElectricFlow is het mogelijk om verscheiden applicaties op een autonome wijze te verschepen naar de gebruikers. Hierbij kan door middel van een procedure een gepersonaliseerde afhandeling plaatsvinden. Door gebruik te maken van de verscheidene projecten is het mogelijk om de release van verschillende applicaties op een geordende manier aan te pakken. Verder wordt niks vermeld over hoe omgegaan wordt met installaties en updates die slecht zijn afgelopen.

2.3.2 Redhat package manager

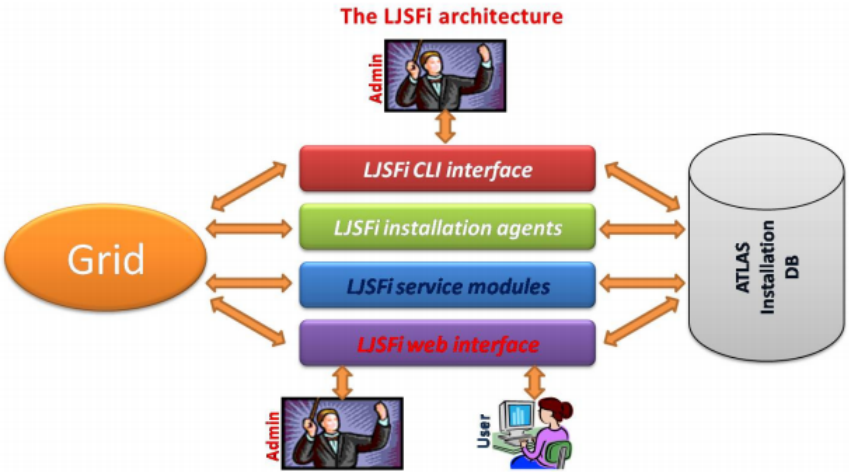
In Linux wordt de Redhat package manager (RPM) het vaakst gebruikt voor de deployment van software. Met hulp van de RPM is het mogelijk om enkele operaties uit te voeren zoals onder andere installatie, updaten, ... De operaties worden ondersteund door een databank die alle informatie en details van de geïnstalleerde pakketten bevat. Een RPM pakket bestaat uit executables gecombineerd met configuratie bestanden en documentatie. Doordat een pakket executables bevat, zal een pakket gekoppeld zijn aan het besturingssysteem van de host [7]. Naast de RPM files bevat een pakket verschillende scripts geschreven in de standaard Unix scripting taal. De verscheidene scripts zijn ingedeeld in sets horende bij een specifieke taak. Bij een error moet een roll back uitgevoerd worden. Dit is de taak van de script schrijver [2].

De Redhat Package Manager is een grofkorrelige, taal onafhankelijke maar besturingssysteem afhankelijke oplossing. Het grootste probleem van RPM schuilt in de afhankelijkheden tussen de pakketten. Niet alle afhankelijkheden zijn explicite gemodelleerd en de afhankelijkheden die wel gemodelleerd zijn, zijn gevormd met de nadruk op de inhoud en niet op de pakketten zelf [2].

2.3.3 ATLAS

ATLAS is één van de vier grote experimenten bij de Large Hadron Collider (LHC) in CERN (de Europese organisatie voor nucleair onderzoek). Het is een algemeen deeltjesfysica experiment onderhouden door een internationale samenwerking met als doel het exploiteren van de mogelijkheden van de Large Hadron Collider. Fysici testen de voorspellingen van het Standaard Model [8] wat kan leiden tot grote ontdekkingen zoals het Higgs boson [9].

Om in de ATLAS samenwerking om te gaan met de grote hoeveelheid bronnen, is er een volledig automatisch installatie systeem ontworpen voor het LCG/EGEE project, LHC Computing Grid/Enabling Grids for E-sciencE [10]. Salvo, Barchiesi, Gnanvo e.a. [11] beschrijft de architectuur van het ontworpen systeem. Het ontwerp van het installatie systeem werd gebaseerd op het Light Job Submission Framework for installation, ook wel LJSFi.



Figuur 2.3: LJSFi Architectuur [11]

De architectuur van het framework is zichtbaar in Figuur 2.3. Het framework vormt een dunne laag over de middleware van Grid. De kern van het systeem bestaat uit de installatie database en de command line interface (CLI). De laatste zorgt voor de interacties met de Grid middleware. Met hulp van de installatie database kan de CLI verschillende taken en job informatie opslaan. Aan de hand van deze informatie kunnen installaties uitgevoerd worden. De installatie databank staat in contact met alle componenten van het framework. Zo kan de status van verscheidene acties en configuraties van verschillende taken opgeslagen worden.

Naast deze twee grote componenten bevat LJSFi service modules en extensies waarmee installatie aanvragen afgehandeld worden. Het systeem bevat drie verschillende componenten die horen bij de LJSFi service modules:

- **RAI module** De Request An Installation module dient als web interface voor het ontvangen van user-driven installatie aanvragen.
- **AIR module** De Automatic Installation Requester schiet in actie als de software release aangeduid staan als productie of veroudert en als de release aangeduid staat met de parameter auto-installatie. De module zal respectievelijk de software installeren of verwijderen op alle sites waar de software tag nog niet aanwezig is. Door de AIR module periodiek te gebruiken, zullen de nodige aanvragen snel afgehandeld worden.
- **InAgent module** Met de InAgent module wordt het mogelijk om volledig geautomatiseerde installatieprocessen te voorzien. Iedere 10 minuten wordt de Installation database gelezen en via de CLI interface worden de nodige installatieprocessen opgestart. Elk installatieproces wordt bijgestaan door een installation agent. De agent zal instaan voor het updaten van de Installatie database met real-time informatie die zichtbaar is online.

Naast de verscheidene automatische services biedt LJSFi enkele gebruiker services aan. Een gebruiker kan zich subscriben voor bepaalde acties op een doel systeem. Als deze actie wordt

uitgevoerd dan krijgt de gebruiker een mail. Hiernaast kan een gebruiker een software release vastpinnen zodat deze niet verwijderd kan worden door het systeem.

Het installatieproces wordt uitgevoerd in drie verschillende stappen. In een eerste stap wordt een site check uitgevoerd door een pilot job naar de site te sturen. Als de check succesvol uitgevoerd wordt, kan het installatieproces beginnen. De acties tijdens het installatieproces worden uitgevoerd door softwaremanagement scripts. Op het einde van het proces, haalt het systeem de job output en exit code op. De laatste wordt opgeslagen in de Installation database.

Obreshkov, Albrand, Collot e.a. [12] bespreekt hoe het ATLAS project te werk gaat bij het inpakken van alle nodige software. Het ATLAS project gebruikt CMT [13] als configuratie manager. Met behulp van een configuratie bestand weten verscheidene tools hoe ze een pakket moeten afhandelen. Rybkin [14] spreekt ook over CMT als informatiebron voor het ophalen van meta-data. Aan de hand van deze data kan een Pacman pakket geproduceerd worden. Met behulp van een "Pacman file" is geweten hoe de ingepakte software behandelt moet worden.

De LJSFi architectuur is een architectuur die om kan gaan met een grote hoeveelheid pakketten. Door gebruik te maken van de modules is het mogelijk om de verschillende pakketten te installeren en te verwijderen op een grote schaal zonder dat menselijke interactie nodig is. Hierbij zorgt CMT voor de nodige meta-data waardoor installatietools niet afhankelijk zijn van de mens. Een nadeel aan deze architectuur is dat het gebruik van Grid nodig is om de LJSFi architectuur in te bouwen. Verder wordt in de lectuur niet ingegaan op welke wijze er wordt omgegaan met een fout tijdens het installatieproces of het verwijderproces. Het enige dat hierover vermeld wordt is dat de exit code van de job wordt opgeslagen in de databank.

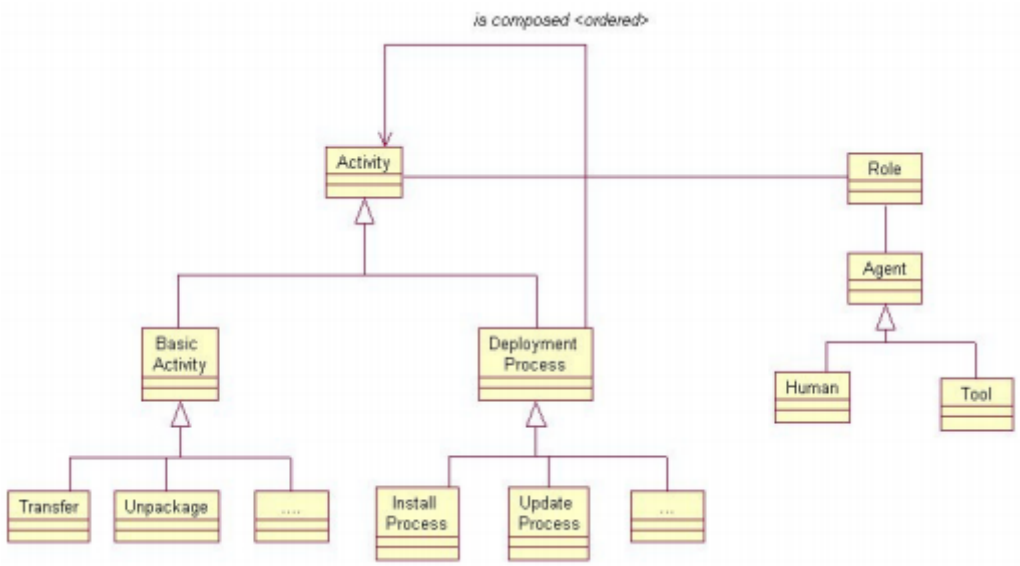
2.3.4 ORYA

Lestideau en Belkhatir [15] legt uit hoe ORYA (Open enviRonment to deploY Applications) verschillende deployment functionaliteiten aanbiedt aan gedistribueerde, autonome entiteiten zoals workstations en servers. Aan de hand van een deployment PSEE [16] wordt het mogelijk om het installatieproces te automatiseren.

In het ontwerp van ORYA worden er drie verschillende entiteiten besproken die nodig zijn om het automatische installatieproces mogelijk te maken.

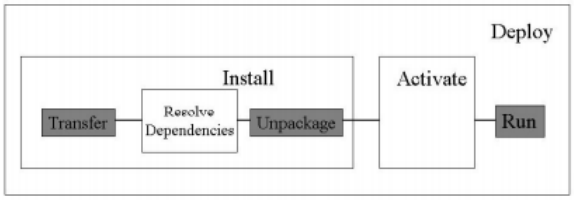
- **Applicatie Server** De applicatie server bevat de informatie nodig voor de installatie. Hieronder bevindt zich onder andere een pakket met de nodige resources en een manifest waarin de afhankelijkheden, beperkingen en kenmerken staan.
- **Target** De target is het doel waarop de deployment uitgevoerd wordt. Iedere target wordt beschreven door de verschillende applicaties die al aanwezig zijn en de fysische beschrijving.
- **Deployment Server** De deployment server vormt de kern van de deployment omgeving en staat in voor het uitvoeren van de deployments. De deployment server zoekt de nodige pakketten, voert een transfer van de pakketten uit en installeert de applicatie. Op het einde moet de deployment server garanderen dat vorige programma's correct blijven functioneren.

Lestideau en Belkhatir [15] beschrijft verder de verschillende modellen die gehanteerd worden om een succesvolle deployment uit te voeren. In Figuur 2.4 op de volgende pagina is het deployment proces model terug te vinden. Een deployment proces zal bestaan uit verschillende basis activiteiten en deployment processen. Iedere activiteit wordt uitgevoerd door een agent.



Figuur 2.4: Deployment proces model [15]

Een toegepast voorbeeld van een deployment aan de hand van dit model is terug te vinden in Figuur 2.5. Een basis activiteit wordt voorgesteld aan de hand van een grijze rechthoek en een deployment proces aan de hand van een witte rechthoek. In het voorbeeld zijn dus vier deployment processen aanwezig en 3 basis activiteiten.



Figuur 2.5: Voorbeeld van een deployment [15]

Aan de hand van deze structuur wordt het mogelijk om het volledige deployment proces voor te stellen.

2.3.5 Ansible

Als laatste case studie wordt Ansible besproken. Ansible is een open source IT configuratie management, deployment en organisatie tool. De architectuur van Ansible bestaat uit een agentless push model. Dit wil zeggen dat er geen additionele software nodig is op de client toestellen. Dit wordt behaald door gebruik te maken van de remote manegement frameworks aanwezig op de toestellen, SSH voor Linux en Unix en WinRM voor Windows. Door geen agents te gebruiken, zal Ansible geen resources gebruiken zolang de Ansible het systeem niet aan het gebruiken is [17].

[17] beschrijft verder dat Ansible gebruik maakt van *Playbooks* voor de organisatie van de IT omgevingen. Playbooks zijn YAML[1] definities van taken die beschrijven hoe een taak moet geautomatiseerd worden. Een Playbook bestaat uit een aantal “plays” die uitgevoerd kunnen worden op een

[1] Een human-readable data serialisatie taal (Een superset van JSON)

set van hosts, ook wel een “inventory” genoemd. Iedere play bestaat uit een set van taken die op een subset van de inventory uitgevoerd kan worden. Een task zelf voert een Ansible module uit[2].

Hiernaast is het mogelijk om de mogelijkheden van Ansible uit te breiden. Modules kunnen zelf geschreven worden in eender welke taal met als enige beperking dat een JSON bestand als input moet gegeven worden en dat een JSON bestand gegenereerd moet worden. De inventory van een Playbook kan dynamisch ontdekt worden at runtime.

In tegenstelling tot ATLAS en Electric Cloud bevat Ansible geen agenten die gebruikt worden om het deployment proces in goede banen te leiden. Het is wel mogelijk om het proces te beïnvloeden door gebruik te maken van de Playbooks. Door gebruik te maken van modules is het in Ansible mogelijk om de functionaliteiten uit te breiden. Wederom is het niet duidelijk wat er gebeurt mochten er fouten optreden tijdens het installatieproces of updateproces.

2.4 Rollback

Tijdens de bespreking van het probleem in Sectie 1.2 op pagina 1 werd al aangehaald dat een soort van rampenplan voorzien moet worden. Zowel omgevingsproblemen als onderhoudsproblemen (zie Sectie 2.2 op pagina 4) kunnen tijdens één van de toestanden in de deployment levenscyclus optreden. Voor Televic is het niet wenselijk dat een probleem tijdens één van deze toestanden ervoor zorgt dat hun software niet meer correct functioneert aangezien dit kan leiden tot het stilleggen van de productie. Daarom wordt in de volgende secties verschillende mechanismen besproken waarmee de verscheidene problemen aangepakt kunnen worden. In wat volgt, wordt vooral rekening gehouden met mechanismen die gebruikt kunnen worden als het probleem zich al voor heeft gedaan. Deze strategieën zullen typisch proberen terug te keren naar een vorige correct werkende toestand van de software. De actie waarbij wordt teruggekeerd naar een vorige toestand wordt ook wel rollback genoemd.

2.4.1 Rollback strategieën

Srinivasan, Kandula, Andrews e.a. [18] spreekt over drie manieren waarop rollback strategieën geïmplementeerd worden in hedendaagse systemen. Checkpointing, main-memory transactions en software rejuvination zijn strategieën die vaak gebruikt zijn.

Checkpointing Checkpointing is een eenvoudige recovery strategie. De toestand van een programma wordt periodiek opgeslagen in een bestand op een extern opslagmedium. Deze kan, na het falen van het programma, gebruikt worden om te herstellen van fouten [19]. Bij een checkpointing systeem wordt de volledige toestand van het programma opgeslagen. Dit zorgt voor een verhoogde overhead waardoor het niet mogelijk is om frequent een checkpoint uit te voeren [18].

Door gebruik te maken van incrementele checkpoints is het evenwel mogelijk om dit probleem aan te pakken. Enkel de veranderingen van de laatste checkpoint worden opgeslagen in een nieuwe checkpoint. Het deel dat niet is aangepast, kan hersteld worden aan de hand van de vorige checkpoint. Dankzij deze strategie is het mogelijk om de hoeveelheid data die moet worden opgeslagen te verkleinen. Hierdoor zullen wel verschillende recovery bestanden nodig zijn. Het is aangewezen om op regelmatige tijdstippen de verschillende recovery bestanden samen te voegen tot één bestand [19], [20].

[2] Een klein stuk code met een specifieke taak

Main-memory transactions Een andere rollback strategie zijn de main-memory transactions. Systemen die deze transacties ondersteunen bezitten vaak de mogelijkheid om terug te keren naar een vorig executie punt. Om deze strategie te kunnen implementeren, moeten applicaties gebruik maken van het transactieprogrammeermodel, waardoor de keuzevrijheden van de programmeur beperkt worden [18].

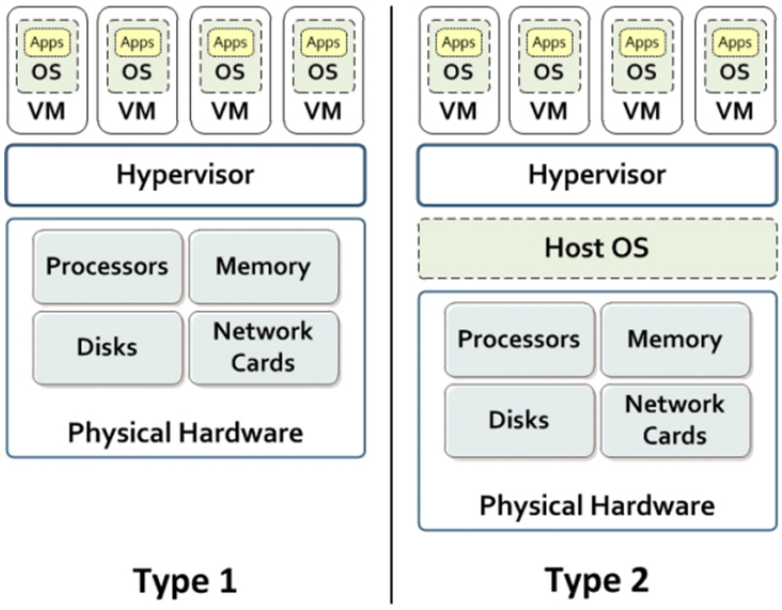
Software rejuvenation Huang, Kintala, Kolettis e.a. [21] definieert software rejuvination als volgt:

> "Software rejuvenation is the concept of gracefully terminating an application and immediately restarting it at a clean internal state."

Tijdens het langdurig uitvoeren van een programma treedt *process aging* op. Door geheugenlekken, niet vrijgegeven bestandlocks, data corruptie, ... zal de performantie van het uitvoerende programma aangetast worden waardoor het programma uiteindelijk faalt. Door het heropstarten van de applicatie worden eventuele fouten uit het systeem gehaald en wordt de software verjongd. De meeste studies over software rejuvenation focussen vooral op het herstarten van de volledige applicatie en werken dus niet op een fijnkorrelige schaal [18].

2.4.2 Virtualisatie

Door middel van virtualisatie wordt de complexiteit die ontstaat door de interactie tussen het programma, de installatieomgeving en de uitvoeringsbeperkingen gelimiteerd. Door het creëren van een perfecte omgeving komen deze problemen niet voor. Er zijn verschillende voordelen gekoppeld aan het gebruiken van een virtuele machine (VM). Bijvoorbeeld, besturingssystemen op verschillende hardware platformen eisen verschillende drivers en deze drivers hebben misschien afhankelijkheden op een bepaalde firmware en BIOS. Een guest OS draaiende op een virtuele machine heeft deze eisen niet [22].



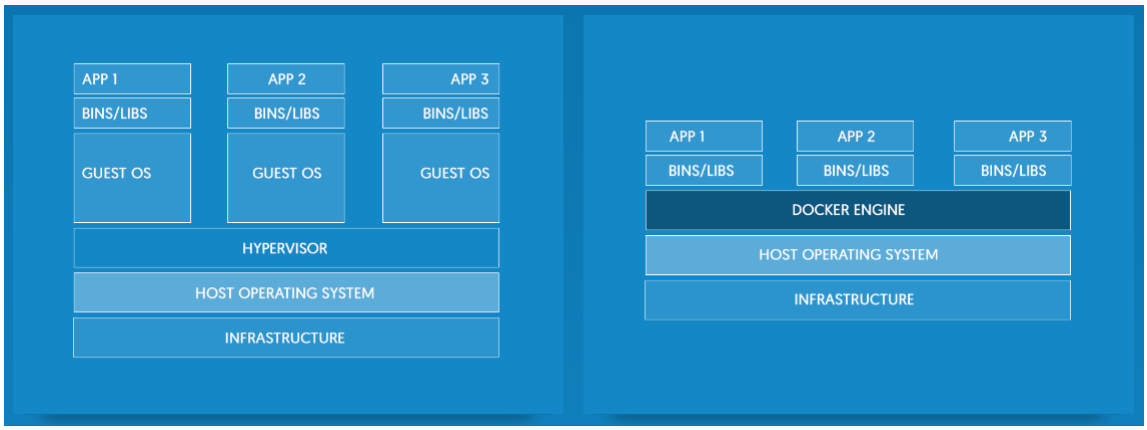
Figuur 2.6: Typer 1 hypervisor in vergelijking met een type 2 hypervisor [23]

Met virtualisatie is het mogelijk om van één computer meerder computers te maken. Dit kan bereikt worden door gebruik te maken van een speciaal programma (een hypervisor). Volgens Fenn, Murphy, Martin e.a. [24] bestaan er twee types hypervisors:

- **Type 1** hypervisors zorgen voor een directe interface naar de host hardware. Er is typisch één virtuele machine met speciale privileges die de andere virtuele machines onderhoudt.
- **Type 2** hypervisors draaien als een normaal programma in het host besturingssysteem. Iedere virtuele machine draait dan als proces in het host besturingssysteem.

Deze twee types zijn zichtbaar in Figuur 2.6 op de vorige pagina.

Een virtuele machine bevat een volledig besturingssysteem. Een gevolg hiervan is dat het uitvoeren van handelingen (zoals opstarten, afsluiten, kopiëren, ...) lang duren. Hiernaast moet tijdens het instellen van een virtuele machine een volledig besturingssysteem geïnstalleerd worden. De tijd die nodig is om een VM in te stellen neemt hierdoor drastisch toe. Aangezien een virtuele machine een volledig besturingssysteem omvat, heeft deze ook de resources nodig om correct te kunnen functioneren. Om verschillende virtuele machines naast elkaar te kunnen laten draaien, moet de host voldoende RAM, CPU en harde schijf ruimte hebben. Door gebruik te maken van een virtuele machine wordt een veilige omgeving gecreëerd maar dit neemt kostbare tijd en resources in beslag.



Figuur 2.7: Architectuur van Virtuele Machine ten opzichte van Docker [25]

2.4.3 Docker

Virtualisatie is niet de enige techniek die gebruikt kan worden om rollbacks te vermijden. Docker containers is een technologie waarmee een stuk software wordt ingepakt in een volledig filesysteem dat alle benodigdheden bevat om correct te functioneren. Hierdoor zal de software overal op eenzelfde manier draaien, ongeacht de omgeving [25]. Docker containers zijn niet hetzelfde als virtuele machines. In de vergelijking [26] worden virtuele machines voorgesteld als huizen terwijl Docker containers worden voorgesteld als appartementen.

De huizen staan volledig op zichzelf en bieden bescherming tegen ongewenste gasten. Ze hebben een eigen infrastructuur met hun eigen water, verwarming, ... Hiernaast zal ieder huis op zijn minste een badkamer, living, slaapkamer en keuken hebben. Het vinden van een klein huis is een ganse klus en vaak zal een huis meer bevatten dan nodig is want dat komt door de manier waarop huizen gebouwd worden.

Appartementen bieden ook bescherming tegen ongewenste gasten, maar zij zijn gebouwd rond een gemeenschappelijke infrastructuur. Het appartementsgebouw biedt gemeenschappelijk water, verwarming, . . . aan, aan elk appartement. Elk appartement verschilt ook nog van grootte. Er bestaan kleine appartementen maar ook grote met meerder slaapkamers. Men huurt enkel hetgeen dat nodig is.

In Figuur 2.7 op de pagina hiervoor zijn de architecturen van virtuele machines en Docker terug te vinden. Het verschil tussen beiden wordt al snel duidelijk. Een virtuele machine zal typisch de applicatie, de nodige binaries en bibliotheken en een volledig besturingssysteem bevatten. Een container bevat de applicatie en de verschillende dependencies maar de kernel wordt gedeeld met alle andere containers en gedragen zich als een geïsoleerd proces in de user space van het host besturingssysteem.

Chamberlain en Schommer [27] bespreekt kort hoe Docker werkt. Docker is een platform dat gebruik maakt van de Linux Containers (LXC de user-space control package voor Linux Containers) om software te encapsuleren. LXC is een virtualisatie techniek waarmee virtuele omgevingen in Linux opgebouwd kunnen worden. De containers zullen processen van elkaar sandboxen zodanig dat een proces een ander niet kan beïnvloeden [28]. Docker zal de LXC software uitbreiden waardoor deployment, distributie en versioning mogelijk wordt. Naast LXC gebruikt Docker AuFS (Advanced Multi-Layered Unification Filesystem) als het filesysteem voor de containers. Doordat het filesysteem gelaagd is, is het mogelijk om verschillende filesystemen over elkaar te leggen.

Merkel [28] vergelijkt de twee virtualisatie technieken en bespreekt de verschillende tussen de twee. Bij virtuele machines moet voor iedere virtuele machine een besturingssysteem geïnstalleerd worden. Al deze besturingssystemen verbruiken RAM, CPU en bandbreedte. Containers zullen piggybacken op het bestaande host besturingssysteem. Hierdoor zal het resource gebruik efficiënter zijn. Een container is goedkoop waardoor het creëren en verwijderen van containers een snelle operatie. Dit komt omdat er enkel een proces moet afgesloten worden in tegenstelling tot het afsluiten van een volledig besturingssysteem. Een voordeel van de VMs ten opzichte van Docker is hun maturiteit. VMs bestaan langer en hebben zichzelf kunnen bewijzen in verschillende situaties.

2.5 Deployment strategieën

Voordat software bij een gebruiker geïnstalleerd of geüpdatet kan worden, moet de software eerst bij de gebruiker geraken. Er kan op twee manieren naar dit proces gekeken worden:

- Wat wordt er precies verzonden naar de gebruiker?
- Wie krijgt de software als eerste?

In de volgende secties worden deze twee verschillende focussen verder besproken.

2.5.1 Data focus

Tian, Zhao, Gao e.a. [29] haalt drie methodes aan om software te deployen:

- disk image-based deployment
- behavior-based deployment
- package-based deployments

Bij disk image-based deployment worden de software en het besturingssysteem op eenzelfde moment naar de target node verzonden. Er zullen verschillende image-servers aanwezig zijn die elk een service aanbieden. Zo zal de image-server van software A een andere image-server zijn dan de image-server die software B gebruikt. Het voordeel van deze strategie is dat, zolang de hardware en software vereisten voldaan zijn, de deployment bestaat uit een simpele read-write operatie. Maar, zoals Tian, Zhao, Gao e.a. [29] al aangeeft, is deze methode niet flexibel. Enkel software die op voorhand werd geconfigureerd op de image-server kan worden gedeployed. Gebruikers met speciale noden kunnen moeilijk worden geholpen. Voor Televic is flexibiliteit een hoofddoelstelling. Iedere node bevat verschillende hardware en is verschillend geconfigureerd. Een disk image-based deployment zal hierdoor niet gebruikt kunnen worden. Het basisidee achter behavior-based deployment is het opnemen van de schijfoperaties tijdens het deployen. Als geweten is welke bestanden aangepast, gecreëerd zijn, ... dan kan het proces nagebootst worden op andere nodes [29]. Zo een proces nabootsen is moeilijk. Kerneloperaties moeten getraceerd worden. Deze methode biedt een verhoogde flexibiliteit aan ten opzichte van de disk image-based deployment maar dit is nog niet voldoende om deployments uit te voeren die uniek zijn per node. De laatste techniek die Tian, Zhao, Gao e.a. [29] aan haalt is package-based deployment. Met behulp van een batch file, waarin alle nodige commando's aanwezig zijn, kan een installatie pakket gedeployed worden naar een target node. Door het gebruik van een batch file wordt de flexibiliteit van de deployment verhoogd.

2.5.2 User focus

De geproduceerde software moet bij verschillende gebruikers terecht komen. Een eenvoudige oplossing zou zijn dat slechts één gebruiker per keer geholpen wordt en dat iedere gebruiker zijn beurt afwacht. Zo'n oplossing is misschien doenbaar mochten slechts een handvol gebruikers de applicatie nodig hebben maar dit is vaak niet het geval. Patterson [30] haalt verschillende argumenten aan voor het distribueren van deze service en haalt enkele punten aan (betrouwbaarheid, bandbreedte en lage wachttijden) waar rekening mee moet gehouden worden alvorens een ontwerpbeslissing genomen wordt. Zoals Patterson [30] aanhaalt, is het belangrijk dat alle gebruikers ten alle tijden de service kunnen gebruiken. Hiernaast moet er rekening gehouden worden met de deployment strategie. Münch, Armbrust, Kowalczyk e.a. [31] spreekt over verschillende strategieën om software uit te brengen:

- *Big-bang*: iedere gebruiker van de applicatie zal op eenzelfde moment overschakelen van de oude naar de nieuwe software. Hierdoor wordt vermeden dat verschillende afdelingen met een andere versie van de software werken. Een nadeel is dat voldoende support aanwezig moet zijn om mogelijke problemen op te lossen.
- *Gefaseerd*: de nieuwe software zal bij een gefaseerde deployment enkel toegepast worden in specifiek geselecteerde projecten. Als deze strategie voor een verlengde periode wordt toegepast, zullen verschillende versies van de software continu aanwezig zijn onder de gebruikers.

Aan beide strategieën zijn zowel voor- en nadelen gekoppeld. Zo zal de big-bang strategie niet voordelig zijn om uit te voeren als de gebruikers verspreid zitten over de wereld. Door het tijdsverschil zal de deployment bij sommige gebruikers plaatsvinden tijdens de werkuren en bij anderen midden in de nacht. Het voordeel van de big-bang strategie is dat alle doelsystemen op een korte

periode omgeschakeld worden naar de nieuwe versie van de applicatie. Het gebruik van de gefaseerde strategie kan het probleem met de tijdszones omzeilen maar deze strategie is ook niet ideaal aangezien het omschakelen van de software naar de nieuwe versie lang kan duren. Een hybride oplossing is hier dus aangewezen.

2.6 Architecturen

Tot op heden werd vooral gekeken naar mogelijke technologieën en tools die gebruikt kunnen worden voor het verspreiden van software. In de komende secties worden enkele architecturen aangehaald die ook een oplossing bieden voor het verspreiden van software. Met een architectuur is het mogelijk om zelf verscheidene ontwerp- en implementatiebeslissingen te nemen. Zo ligt de programmeertaal niet vast en kan een taal gekozen worden die besturingssysteem onafhankelijk is. In de volgende secties wordt vooral gekeken of de architecturen een schaalbare oplossing bieden voor het deployment proces.

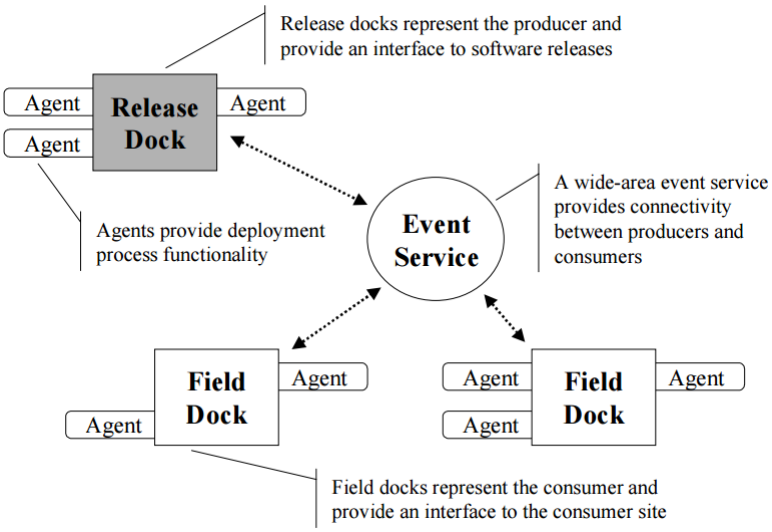
2.6.1 Client/Server architectuur

2.6.2 Software dock architectuur

Hall, Heimbigner en Wolf [1] bespreekt een interessante architectuur die gebruikt kan worden voor het verspreiden van software. Het Software Dock research project creëerde een raamwerk om de samenwerking tussen software producenten en gebruikers te verbeteren. In Figuur 2.8 op de volgende pagina wordt de ontworpen architectuur voorgesteld. Er worden twee verschillende componenten gedefinieerd waarmee de producenten en gebruikers voorgesteld worden. In de architectuur worden de verschillende producenten voorgesteld aan de hand van een release dock en worden de gebruikers voorgesteld als een field dock. Aan deze docks worden verschillende agenten gekoppeld. Elke agent hoort typisch bij één stap uit de software levenscyclus die besproken werd in Sectie 2.2. Naast de verschillende docks wordt ook een wide-area eventsysteem gedefinieerd. Met dit systeem wordt de communicatie tussen de docks aangeboden. Hall, Heimbigner, Van Der Hoek e.a. [32] bespreekt in detail hoe de Software Dock architectuur gebruikt kan worden voor het verspreiden van software.

De release dock is een server die zich bevindt bij de softwareproducent. De release dock biedt een release-repository aan waaruit de gebruikers de applicaties selecteren voor deployment. In de release dock wordt ieder release semantisch beschreven aan de hand van een Deployable Software Description file. Dit bestand bevat onder andere:

- **Assertion**: deze worden gebruikt om beperkingen die aan de gebruikerskant waar moeten te beschrijven. Als niet aan deze beperkingen voldaan wordt, zal het deployment proces falen.
- **Afhankelijkheden**: deze worden gebruikt om beperkingen van de release te beschrijven. Als deze niet voldaan zijn aan de gebruikerskant dan kunnen nog oplossingen gevonden worden. Bijvoorbeeld door additionele software te installeren kan de beperking voldaan worden.
- **Configuratie**: deze zorgt voor een beschrijving van de software die wordt uitgegeven. Hierbij horen bijvoorbeeld varianten en aangepaste versies.



Figuur 2.8: Software Dock Architectuur [1]

Elke release wordt vergezeld door enkele agents die de semantische betekenis lezen en zo de deployment kunnen uitvoeren. Aan de hand van interfaces kunnen de agents aan de services en inhoud van de release dock. Bij het wijzigen van een software release zal de release dock verschillende events afvuren. Agents kunnen zich subscriben bij deze events en weten zo wanneer bepaalde handelingen uitgevoerd moeten worden [1].

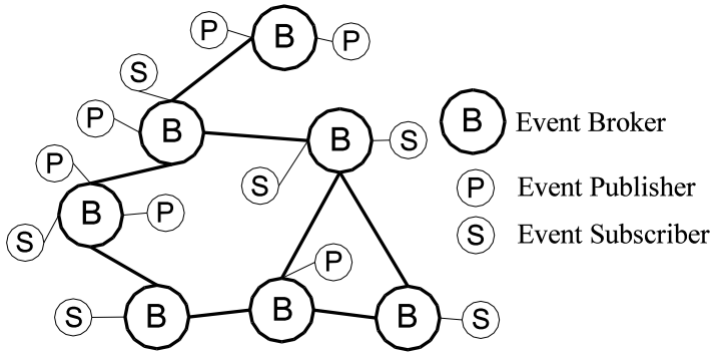
De field dock dient als een interface naar de gebruiker kant toe. Deze interface biedt informatie over de resources en configuratie van het gebruiker systeem. Op basis van deze informatie wordt een context opgebouwd waarin de releases van de resource dock worden gedeployed. De agents die horen bij een release, docken zichzelf in de field dock en kunnen aan de hand van deze interface het gebruikers systeem ondervragen. Aangezien kritische client-side informatie op een gestandaardiseerde wijze aangeboden wordt, met behulp van een geneste collectie van pair-values die een hiërarchie vormen, kan de installatie van de software gepersonaliseerd worden [1].

Event based service

Een belangrijk onderdeel van de software dock architectuur is de event service. Deze handelt de communicatie af tussen de verschillende docks en is een spilfiguur in de architectuur. Aan de hand van Pietzuch en Bacon [33] en Carzaniga, Rosenblum en Wolf [34] is het mogelijk om een bespreking te geven over hoe de event service geïmplementeerd kan worden.

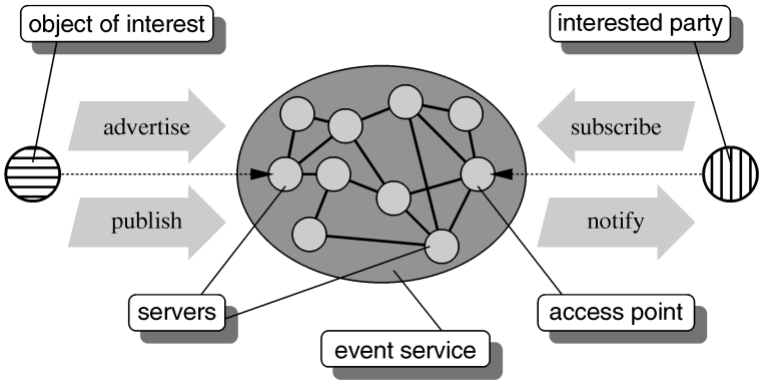
Meeste middleware systemen zijn invocation-based systemen waarbij het request/response paradigma gebruikt wordt om de communicatie tussen client en server te onderhouden. Een client verzoekt een service van de server, waar de server vervolgens op antwoord. Het toepassen van deze strategie is doenbaar in een omgeving met een beperkt aantal clients en servers. Om een grootschalig netwerk op te bouwen en om te kunnen gaan met een dynamische omgeving moet een andere manier van communiceren gebruikt worden. Publish/subscribe systemen bieden een oplossing voor deze problemen. Clients, event subscribers, tonen hun interesse voor een bepaald onderwerp en servers, event publishers, produceren een event die naar alle geïnteresseerden wordt doorgestuurd. Een algemeen voorbeeld is terug te vinden in Figuur 2.9. Door deze manier van communiceren te hanteren, ontstaat een natuurlijke ondersteuning voor many-to-many relaties

tussen de clients en servers. De twee worden ook van elkaar ontkoppeld. Voor de client maakt het niet uit welke publisher de producent is van een event. Hiernaast moet de server niet weten wie allemaal de events ontvangt die hij produceert [33].



Figuur 2.9: Broker netwerk [33]

Een naïeve aanpak, volgens Carzaniga, Rosenblum en Wolf [34], is het gebruiken van één centrale server waar alle subscripties worden bijgehouden, waar alle events toekomen, waar de bestemming van het event beslist wordt en waar het event wordt doorgestuurd naar de gepaste subscribers. Deze strategie is eenvoudig te implementeren maar deze werkt de schaalbaarheid tegen. Dit was ook al duidelijk in Figuur 2.9 waar verschillende “Brokers” aanwezig zijn. Het is belangrijk om stil te staan bij enkele ontwerp beslissingen zodanig dat de service die in Figuur 2.10 zichtbaar is, implementeerbaar is.



Figuur 2.10: Publish/Subscribe service [34]

Naast de architectuur is het belangrijk om te weten wat wordt verzonden en op welke manier. Carzaniga, Rosenblum en Wolf [34] haalt een structuur aan waarin een event beschreven wordt als een set van attributen. Ieder attribuut bestaat uit een type, naam en waarde. De naam van een attribuut is een string en het type komt uit een set van primitieven die terug gevonden worden bij de meeste hedendaagse programmeertalen. Een voorbeeld van zo’n event is terug te vinden in Figuur 2.11 op de pagina hierna

string	*class = finance/exchanges/stock*
time	*date = Mar 4 11:43:37 MST 1998*
string	*exchange = NYSE*
string	*symbol = DIS*
float	*prior = 105.25*
float	*change = -4*
float	*earn = 2.04*

Figuur 2.11: Publish/Subscribe Event [34]

2.7 Technologieën

In de vorige secties werden oplossingen gezocht voor het verspreiden van software naar verschillende gebruikers. Hierbij werd vooral rekening gehouden met op welke wijze de software verzonden word. Er werd nagegaan of een oplossing zorgde voor een schaalbare architectuur die mogelijkheden biedt naar de toekomst toe. Er werd zelfs gezocht naar een rampenplan dat gebruikt kan worden als er fouten optreden tijdens het installatieproces (Sectie 2.4 op pagina 12. In wat volgt wordt niet meer gekeken naar hoe de software bij de gebruiker raakt maar wat er bij de gebruiker raakt.

Implementaties in verschillende programmeertalen, verschillende data representatie formats of incompatibele runtime environments kunnen aan de basis liggen voor een lastige integratie van computerprogramma's. Door gebruik te maken van additionele software, wordt het mogelijk om het gat tussen de verschillen te overbruggen [35]. Het Python testraamwerk is een verzameling van verschillende drivers en bibliotheken elk met een eigen implementatie. Door additionele software toe te voegen is het mogelijk dat verschillende software "pakketten" op een gelijkaardige manier behandelt worden. In wat volgt zullen verschillende technologieën en tools besproken worden die ervoor zorgen dat verschillende software "pakketten" een geheel vormen. Dit geheel kan vervolgens verzonden worden naar de gebruikers waar de software geïnstalleerd moet worden.

Om de verschillende technologieën te vergelijken werd een algemeen testscenario uitgedacht. Het scenario ziet er als volgt uit: er moet een geheel gemaakt worden waarmee twee verschillende pakketten (die drivers en bibliotheken moeten voorstellen) geïnstalleerd moeten worden. Hierna werd er onderzocht hoe één van de twee pakketten geüpdatet kon worden. Door de technologieën te onderwerpen aan een test, wordt het mogelijk om de voor- en nadelen van iedere technologie te achterhalen. Hiernaast wordt het ook mogelijk om de technologieën te vergelijken aangezien zij eenzelfde functionaliteit moeten voorzien.

WiX Toolset

Windows installer XML Toolset is een set van build tools waarmee Windows Installer packages gemaakt worden van XML broncode. De toolset is geschreven in C# en heeft het .Net framework nodig om te kunnen functioneren. Bron code wordt gecompileerd en vervolgens gelinkt om een executable te maken. Met de toolset kunnen .msi installatie pakketten, .msm merge modules en .msp patches gecombineerd worden tot een Windows executabel.[36].

Aan deze technologie zijn verschillende voor- en nadelen verbonden (zie Tabel 2.1). Een fragment van de WiX toolset code is terug te vinden in Listing B.1 op pagina 52. De WiX toolset maakt installer uitsluitend bedoelt voor de Windows installation engine. Hierdoor worden verscheidene functionaliteiten eenvoudig te gebruiken, zoals het maken van uitzonderingen in de Windows Fire-

WiX Toolset	
Pro	Cons
Diepe integratie met Windows	XML structuren zorgt voor veel overhead
Mogelijkheid om externe executables te includeren	Niet cross-platform
NSIS	
Pro	Cons
Scripting taal	Niet cross-platform
Verschillende plug-ins beschikbaar	Geen structuur voor packages
Chocolatey	
Pro	Cons
Volledige deployment infrastructuur al aanwezig	Niet cross-platform
	Command-line tool
Qt Installer Framework	
Pro	Cons
Cross-platform	XML structuren zorgt voor veel schrijfwerk
Mogelijkheid om externe executables te includeren	Enkel Linux installer maken in Linux

Tabel 2.1: Voor- en nadelen van de verschillende technologieën

wall. Door gebruik te maken van de Windows installation engine is het niet mogelijk om de executabel te gebruiken in Linux omgevingen[3]. WiX maakt gebruik van XML broncode om verschillende elementen te definiëren. Liefke en Suciu [38] geeft al aan dat XML niet een van de meest efficiënte dataformaten is, maar het verhoogd de flexibiliteit wel. Het creëren van een XML bestand met de hand is een langdradig en moeilijk werk.

NSIS

Nullsoft Scriptable Install System is een open source systeem waarmee Windows installers gemaakt kunnen worden. Zoals de naam aangeeft, is NSIS script-based. Hierdoor bevatten installers de nodige om verschillende installatie taken uit te voeren. Door de grote gebruikersbasis is een grote hoeveelheid plug-ins en scripts beschikbaar. Alle plug-ins en scripts kunnen op een eenvoudige manier aan een installer toegevoegd worden voor een verhoogde functionaliteit [39].

De voor- en nadelen verbonden aan NSIS zijn terug te vinden in Tabel 2.1. Met de scripting taal van NSIS is het mogelijk om eenvoudige installer te definiëren (een voorbeeld hiervan is terug te vinden in Listing B.2 op pagina 53). De scripting taal is intuïtiever te gebruiken in vergelijking met de XML bestanden van Wix Toolset. Dankzij grote hoeveelheid aan plug-ins die aanwezig zijn, is het eenvoudig om een installer te creëren met verschillende functionaliteiten. De gecreëerde installer is een Windows executabel en de opmerking gegeven bij de WiX Toolset is hier ook van toepassing. Het feit dat NSIS bedoelt is om eenvoudige installers te maken zorgt ervoor dat het niet mogelijk is om aparte pakketten te definiëren. Ieder pakket kan wel een eigen configuratie hebben maar dit wordt allemaal toegevoegd aan één script. Bij een grote hoeveelheid aan pakketten leidt tot wanorde en is er geen globaal overzicht. NSIS beidt ook geen mogelijkheden aan om

[3] Dit kan eventueel omzeilt worden door het gebruik te maken van software zoals Wine [37]. Als geen alternatieven aanwezig zijn, dan is deze strategie eventueel het overwegen waard.

geïnstalleerde software up te daten. Om die eigenschappen toe te voegen moet er beroep gedaan worden op andere software.

Chocolatey

Volgens *Chocolatey About page* [40] is Chocolatey een package mangager voor Windows net zoals apt-get voor Linux is. Het is ontworpen als een gedecentraliseerd framework met als doel het snel installeren van applicaties en tools. Chocolatey werd gebouwd boven op de NuGet infrastructuur gecombineerd voor het verspreiden van de packages en gebruikt PowerShell voor een gepersonaliseerde installatie.

In Tabel 2.1 zijn de verschillende voor- en nadelen gekoppeld aan Chocolatey terug te vinden. Het grootste voordeel dat bekomen wordt bij het gebruiken van een package manager is het al bestaan van een deployment infrastructuur. Na het installeren van Chocolatey op de client kunnen alle nodige packages voor het framework geïnstalleerd worden. Hiernaast kunnen scripts gekoppeld worden aan iedere package zodanig dat een aangepaste installatie mogelijk is. Net zoals apt-get voor Linux, is Chocolatey te gebruiken in de command-line. Dit is vooral een nadeel naar gebruiksvriendelijkheid toe aangezien er vanuit wordt gegaan dat de gebruikers amper tot geen ervaring hebben met de command-line in Windows/Linux. Het grootste nadeel aan deze technologie is, net zoals de vorige opties, het niet cross-platform zijn.

Qt Installer Framework

Het Qt Installer Framework biedt een set van tools aan voor het creëren van installers op verschillende platformen. Aan de hand van een set van pagina's wordt de gebruiker door het installatie-, update- en verwijderproces. Hierbij kunnen scripts gebruikt worden om het proces te vereenvoudigen [41].

Aan deze technologie zijn verschillende plus- en minpunten verbonden. Een vergelijking van de verschillende voor- en nadelen is terug te vinden in Tabel 2.1. Het grootste voordeel van het Qt Installer framework is het cross-platform zijn. Hierdoor is het mogelijk om installers te maken voor zowel Windows als Linux. Een nadeel dat hieraan verbonden is, is dat een Linux installer enkel kan gemaakt worden op in een Linux omgeving. Het is niet mogelijk om een Linux installer te maken op een Windows systeem. Hiernaast is het wel mogelijk om voor ieder pakket een aparte installatieprocedure te implementeren.

Hoofdstuk 3

Analyse en ontwerp

3.1 Inleiding

Uit de probleemstelling werd het snel duidelijk dat het probleem omtrent het installeren en updaten van programma's complexer is dan op het eerste zicht lijkt. In dit hoofdstuk zal het probleem verder geanalyseerd worden. Aan de hand van deze bevindingen gaat een architectuur ontworpen of geselecteerd worden. Deze vormen de basis voor de implementatie van de demo.

3.2 Analyse

Het probleem van Televic was het volgende: Televic fabriceert producten die moeten voldoen aan strenge veiligheidsnormen. Om hun producten hierop te kunnen testen heeft Televic een Python testraamwerk ontworpen waarmee het mogelijk wordt om de producten aan verschillende testscenario's te onderwerpen. Dit testraamwerk maakt gebruik van een grote set aan drivers en bibliotheken om een correcte werking te garanderen. Een direct gevolg hiervan is dat het installatieproces op een nieuwe testtoren veel tijd in beslag neemt en foutgevoelig is. Het is belangrijk om rekening te houden met deze fouten en een rampenplan te voorzien. Hiernaast groeit het aantal gebruikers van het testraamwerk continu samen met het aantal drivers, bibliotheken en programma's die verspreid moeten worden. Verder dient elk nieuw toestel op het raamwerk ondersteund te worden waardoor er jaarlijks ettelijke releases van het raamwerk verspreid moeten worden. Er moet dus een antwoord gegeven worden op de volgende vragen:

- Wat moet verzonden worden naar de gebruikers?
- Hoe raakt de software van de producent naar de gebruiker?

Het doel is om een systeem te ontwikkelen dat Televic kan bijstaan bij het installatieproces en verspreidingsproces maar ook om een systeem te ontwikkelen dat kan blijven gebruikt worden in de toekomst. Schaalbaarheid en flexibiliteit zijn hierbij zeer belangrijk.

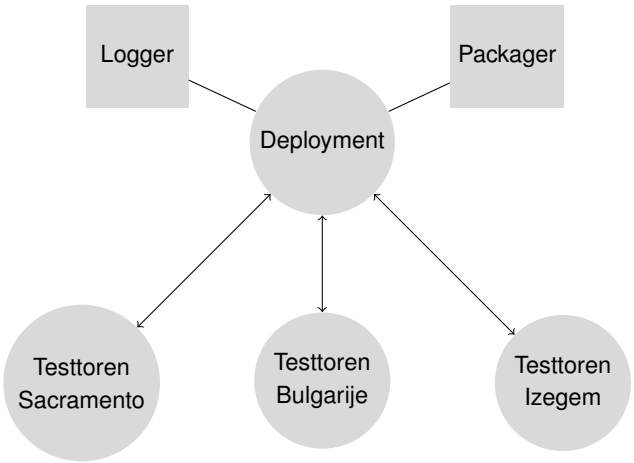
De probleemanalyse onthulde dus dat het probleem onder te verdelen is in verschillende deelproblemen (Wat wordt verzonden en hoe wordt dit aangepakt). Het testraamwerk bestaat uit verschillende componenten, hieronder vallen de drivers en bibliotheken. Elke component heeft een aparte installatiewijze en sommige componenten moeten voor andere geïnstalleerd worden. Zo

zal Python één van de eerste componenten zijn die geïnstalleerd moet worden. Hiernaast moeten verscheidene componenten geconfigureerd tijdens het installatieproces aan de hand van een configuratiebestand. Dit configuratiebestand is afhankelijk van testtoren waarop het testraamwerk op geïnstalleerd word. Door gebruik te maken van additionele software worden verschillen in implementaties, door bijvoorbeeld verschillende programmeertalen, opgevangen. Een deel van de applicatie zal dus bestaan uit deze additionele software die instaat voor het inpakken van de componenten: de *packager*. Hiervoor kan beroep gedaan worden op verscheidene technologieën, structuren en architecturen die besproken werden in Sectie 2.7. Door gebruik te maken van één van deze technologieën is geweten wat er wordt verzonden.

Naast de wat moet er ook geweten zijn hoe de software bij de gebruikers moet geraken. Door dit proces te automatiseren, is het mogelijk om waardevolle informatie te verzamelen. Met deze informatie kunnen rapporten gegeneerd worden over het deployment proces. In Secties 2.5 - 2.3 werden verschillende problemen maar ook oplossingen besproken die aan de basis liggen voor het ontwerp van dit onderdeel van de applicatie. In het vervolg van de thesis zal dit onderdeel (dat zal instaan voor het verspreiden van het testraamwerk maar ook voor de communicatie tussen de producten van het testraamwerk en de gebruikers) vermeld worden als de *deployment server*.

In Sectie 2.2 op pagina 4 werd besproken welke problemen kunnen optreden tijdens het installatieproces. Deze problemen moeten opgevangen worden om een schaalbare oplossing te bedenken voor Televic. Om dit op te vangen, kan er gebruik gemaakt worden van één (of meerdere) strategieën die besproken werd in Sectie 2.4 op pagina 12. Dit onderdeel van de applicatie vooral aanwezig aan de client-side aangezien dat de plaats is waar het testraamwerk aanwezig zal zijn. In de loop van de thesis zal naar dit onderdeel verwezen worden als de *deployment environment*.

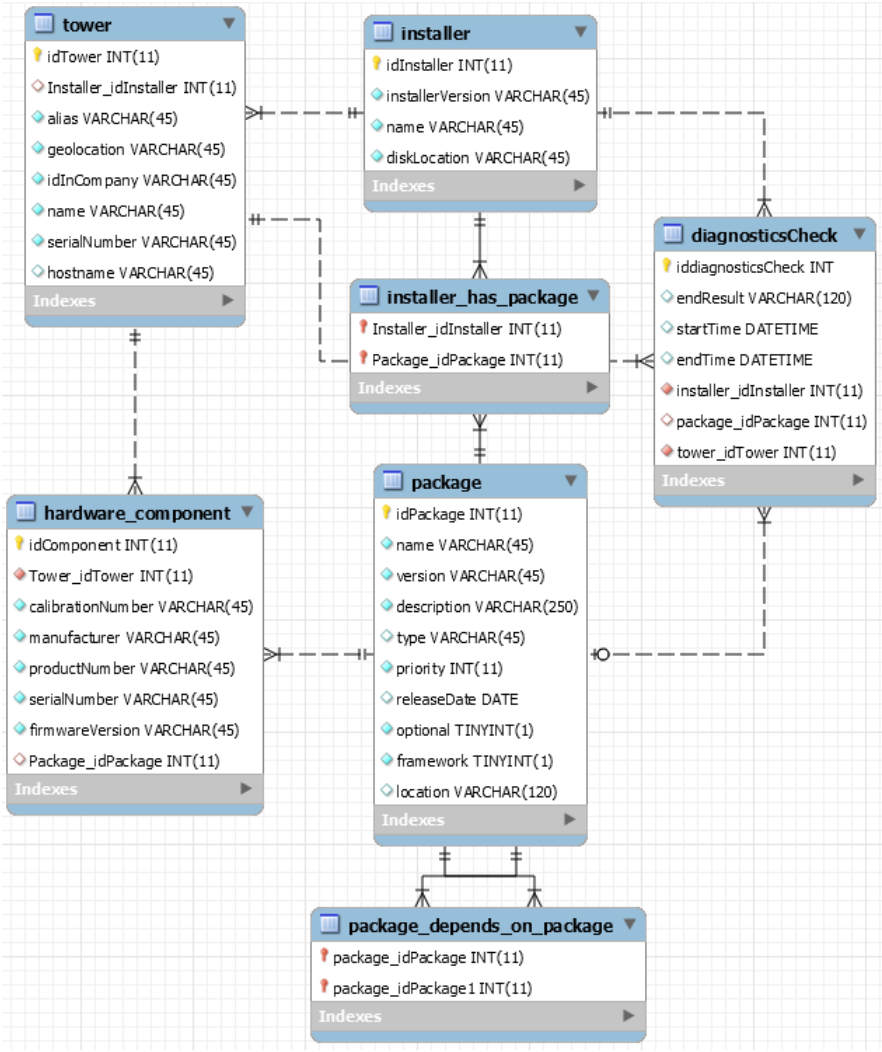
Na de probleemanalyse is het nu duidelijk dat het werk op te delen valt in drie grote componenten. Deze drie onderdelen zullen de basis vormen voor de architectuur en zullen gebruikt worden als leidraad. Het eerste onderdeel zal bestaan uit de packager met als doel het inpakken van de nodige drivers, bibliotheken, ... Naast de packager is er de deployment server instaat voor het verspreiden van de installers die de packager aflevert. Aan de client-side zal de deployment environment aanwezig zijn waardoor installatie-complicaties vermindert worden door de installatie te isoleren. Mocht een rollback nodig zijn, dan kan deze op een eenvoudige manier gebeuren. In Figuur 3.1 wordt de algemene structuur van de applicatie weergegeven. Met behulp van deze basis is het mogelijk om een demo te produceren voor de finale verdediging.



Figuur 3.1: Overzichtsdiagram van de algemene structuur

3.3 Databank ontwerp

In de voorgaande Sectie werd aangehaald met welke problemen Televic kampt. Eén van de problemen is de continue groei van pakketten waar het framework gebruikt van maakt en het aantal gebruikers die het framework gebruiken. Om dit probleem aan te pakken wordt een databank ontworpen voor het opslaan van alle cruciale data over zowel het installatieproces en alle gebruikers. In overleg met Televic werd ervoor gekozen om MySQL te gebruiken als managementsysteem. Het ontwerp van de databank is terug te vinden in Figuur 3.2.



Figuur 3.2: Ontwerp van de databank

De tabellen tower en hardware_component dienen om iedere gebruiker, typisch een testtoren of een laptop, te beschrijven. Iedere toren heeft een ID, naam en serienummer. De combinatie van deze drie waarden is uniek binnen het bedrijf en de combinatie kan gebruikt worden als identificatie binnen het systeem maar deze combinatie draagt geen betekenis voor een gebruiker. Om dit op te vangen wordt aan iedere toren een alias gekoppeld waardoor de identificatie voor mensen vlotter kan verlopen. Elke toren bestaat typisch uit verschillende hardware componenten, zoals voedingen of netwerkkaarten, die nodig zijn om testen uit te voeren. Iedere component is gemaakt door een

bepaalde fabrikant en krijgt van de fabrikant een serienummer. Vanuit het bedrijf wordt losstaand hiervan een nummer toegekend aan iedere component die gebruikt wordt om de calibratie instellingen te achterhalen. Iedere hardware component gebruikt firmware om correct te functioneren. Naast alle bovengenoemde informatie wordt ook de versie van de firmware opgeslagen. Door het opslaan van al deze informatie wordt het mogelijk om:

1. Torens van elkaar te onderscheiden
2. Achterhalen welke hardware componenten aanwezig zijn op welke toren
3. Welke firmware versie draait op welke hardware component

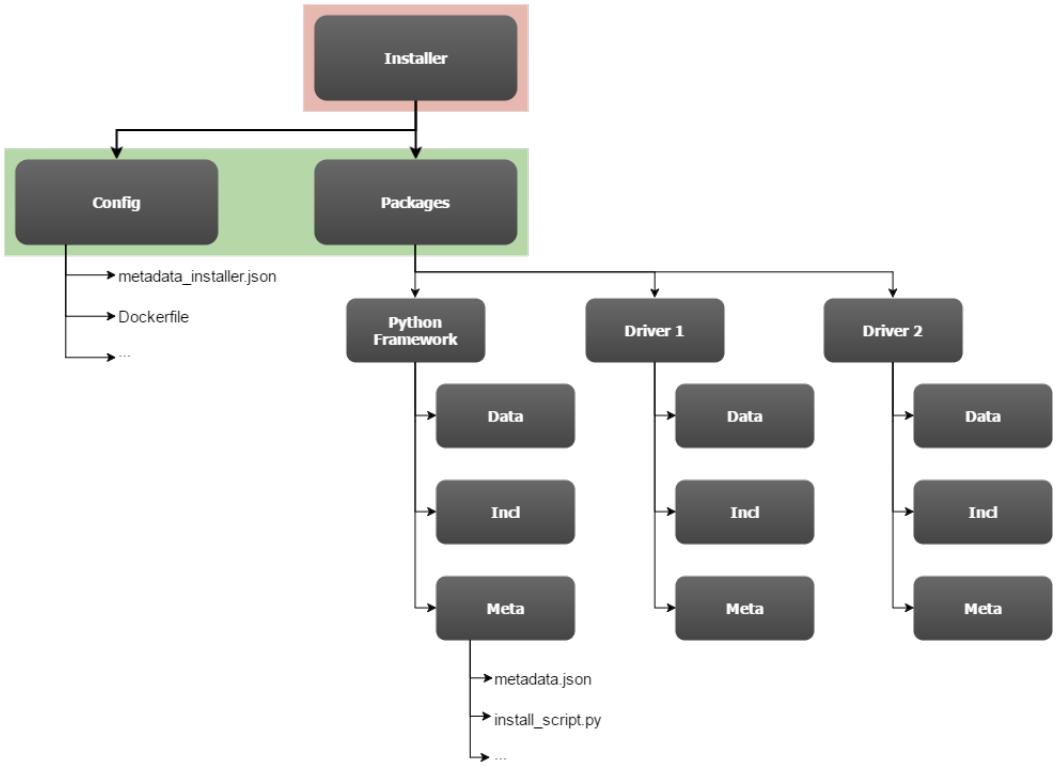
Naast informatie over de gebruikers wordt ook informatie over de verschillende installers en pakketten bijgehouden. Een installer bestaat uit een combinatie van verschillende software pakketten. Bij deze pakketten moet één pakket aanwezig zijn dat het Python testraamwerk bevat. Hiernaast zijn verschillende andere pakketten aanwezig voor drivers. In Figuur 3.3 op de pagina hierna is een voorbeeld zichtbaar van een installer die bestaat uit drie verschillende pakketten. Eén pakket wordt gebruikt door het testraamwerk en de twee anderen voor drivers die nodig zijn om het testraamwerk correct te laten functioneren. Iedere toren wordt gekoppeld aan één installer en zo aan één testraamwerk. Hiernaast is het mogelijk om pakketten te koppelen aan hardware componenten. Zo kan een driver voor een voeding gekoppeld worden aan de entry van de voeding die aanwezig is in de hardware_component tabel. Op deze wijze worden de hardware-software afhankelijkheden bijgehouden. Van ieder pakket wordt bijgehouden welk type pakket het is (een executabel, zip bestand, ...), de prioriteit voor de installatievolgorde, een korte beschrijving en de release datum. Naast al deze informatie wordt er ook bijgehouden welke pakketten afhankelijk zijn van elkaar. Een voorbeeld is hiervan is een testraamwerk pakket en een pakket waarmee Python geïnstalleerd wordt. Het testraamwerk is afhankelijk van Python om correct te functioneren. Zo worden de verschillende software-software afhankelijkheden bijgehouden. Voordat een installer gemaakt wordt, die een testraamwerk pakket bevat, kan gecontroleerd worden dat ook het Python installatie pakket aanwezig is.

Verder zijn er enkele tabellen aanwezig voor het ondersteunen van testen. Tijdens en na het installatieproces moet het mogelijk zijn om testen uit te voeren. Dankzij deze testen is het duidelijk of een bepaald pakket correct functioneert en op het einde van het installatieproces kan gecontroleerd worden of het volledige testraamwerk in zijn geheel functioneert. Doordat er een link wordt bijgehouden tussen een hardware component en een pakket, is het mogelijk om hieruit waardevolle informatie te halen. Zo kan bijvoorbeeld een verband gelegd worden tussen een bepaalde versie van een driver en de firmware die aanwezig is in een hardware component. Deze informatie kan gebruikt worden om problemen in testtorens te vermijden.

3.4 Architectuur

3.4.1 Packager

De architectuur van de packager wordt gebaseerd op de architectuur en structuur van het Qt installer framework. Er wordt een installer geproduceerd die bestaat uit verschillende pakketten die elk instaan voor het installeren van een software component. Voor iedere installer wordt een aparte folder structuur aangemaakt die zichtbaar is in Figuur 3.3 op de volgende pagina. In de config folder van de installer worden alle globale scripts en beschrijvingsbestanden bijgehouden. Verder bevat



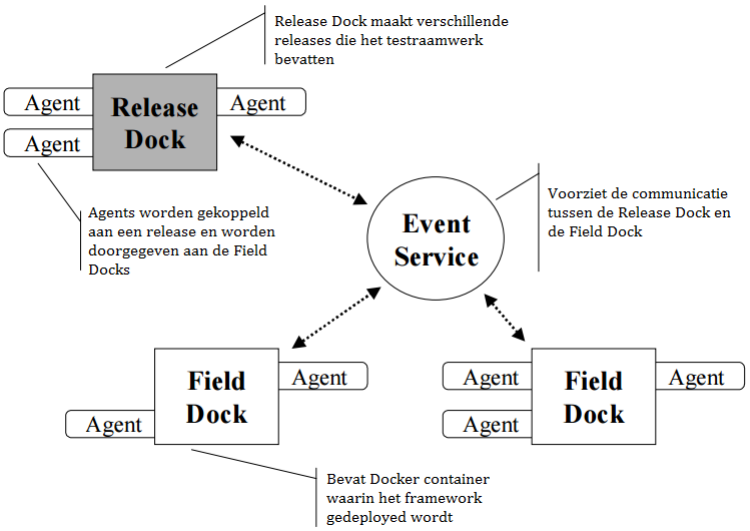
Figuur 3.3: Structuur van een installer bestaande uit drie pakketten

de installer subtrees voor ieder pakket. Een pakket bestaat vervolgens uit een data, include en meta folder. De data folder wordt gebruikt om de effectieve driver/bibliotheek in op te slaan. Hiernaast is een include folder aanwezig waarin verschillende afzonderlijke scripts toegevoegd kunnen worden. Op deze manier kunnen willekeurige scripts (bijvoorbeeld een script die de firewall instellingen aanpast) rap toegevoegd worden aan een pakket. Als laatste bevat de meta folder alle meta-data horende het pakket. Dit omvat onder andere een script die gebruikt wordt om te testen of het pakket wel correct functioneert maar ook een beschrijving van het pakket zelf.

Door zelf een packager te produceren, is het mogelijk om iedere stap in het deployment proces te personaliseren. Op deze manier kan na het installeren van een pakket een zo optimaal mogelijke afhandeling plaats vinden. Zo kunnen testen op ieder moment in het installatieproces toegevoegd worden, een handeling die met het Qt installer framework ook mogelijk is maar dit is moeilijker te realiseren. Doordat een gepersonaliseerde packager wordt ontworpen, worden problemen met Docker vermeden. Verder wordt besproken hoe Docker gebruikt wordt in de deployment environment om verscheidene problemen die gerelateerd zijn aan het installatieproces op te vangen. De Docker omgeving, zoals reeds vermeld in Sectie 2.4.3 op pagina 14, maakt gebruikt van LXC. Het besturingssysteem van de containers is hierdoor Linux. Windows gebruiken als besturingssysteem voor de containers is mogelijk in Docker is mogelijk maar deze optie staat nog altijd in beta schoenen. Mocht het Qt installer framework gebruikt worden als packager, dan moet het besturingssysteem van de server ook Linux zijn. Door ervoor te kiezen om zelf een packager te produceren, kan voor een cross-platform oplossing gezocht worden en kan de integratie van Docker vlotter verlopen. Op deze manier wordt er een abstractie gemaakt van het besturingssysteem van zowel de server als client.

3.4.2 Deployment server

Het centrale systeem in de architectuur is de deployment server. Zoals reeds uitgelegd zal dit onderdeel instaan voor het verspreiden van de verschillende installers en functioneren als een verzamelcenter voor alle informatie. De architectuur van de deployment server wordt gebaseerd op de software dock architectuur die besproken werd in Sectie 2.6.2 op pagina 17 en is terug te vinden in Figuur 3.4. De software dock architectuur bestaat uit 4 grote componenten, namelijk het release dock, field dock, event service en de agenten.



Figuur 3.4: Software Dock Architectuur [1]

Release Dock Het release dock bevindt zich aan de serverzijde en bevat de code voor zowel de packager als de monitor van de clients. Met hulp van de packager worden de verschillende installers geproduceerd. Naast het produceren van installers zal het release dock instaan voor het monitoren van de field docks. Hier wordt vooral gebruik gemaakt van de verschillende resultaten van de test die uitgevoerd worden tijdens het installatieproces. Op de manier wordt achterhaalt welke installer geïnstalleerd werd op een field dock en welke fouten eventueel optraden tijdens het installatieproces.

Event Service Nevens de verschillende docks beschrijft de software dock architectuur een Event Service. Deze staat in voor het afhandelen van de communicatie tussen de verschillende docks. Om dit te implementeren, wordt er beroep gedaan op Sectie 2.6.2 op pagina 18. Er worden evenwel enkele kleine aanpassingen gemaakt. De broker worden verschillende lijsten bijgehouden voor ieder type van bericht dat mogelijk is. Bij het toekomen van een subscribe/unsubscribe bericht, zal de broker de nodige lijsten aanpassen zodat de zender van het bericht wordt toegevoegd of verwijderd. Berichten in het netwerk zullen bestaan uit een set van attributen en waarden in een JSON formaat. Het uiterlijk van zo'n bericht is gelijkaardig als het uiterlijk van een bericht in Figuur 2.11 op pagina 20. In Listing 3.1 op de volgende pagina wordt een voorbeeld van een bericht weergegeven. Ieder bericht heeft dezelfde set aan attributen die elk een eigen functie hebben:

- *Timestamp*: ieder bericht zal op het moment van de creatie een timestamp krijgen. De timestamp wordt uitgedrukt in seconden sinds epoch. Op deze wijze wordt eenzelfde referentie

punt gebruikt en is het mogelijk om de creatie van berichten in de tijd te ordenen.

- *Type*: ieder bericht hoort toe aan een bepaald type. De verschillende types van berichten zijn: subscribe, unsubscribe, new, change, rapport, update, release. Afhankelijk van het type bericht dat toekomt, bij zowel de broker als een dock, zal een andere actie ondernomen worden. De eerste twee types zijn uitsluitend bedoelt voor de broker. De volgende drie voor een release dock en de laatste twee voor de field docks.
- *ID*: naast een timestamp wordt aan ieder bericht een id meegegeven. Dit attribuut wordt meegegeven zodanig dat de field docks kunnen achterhalen of zij achterlopen op berichten.
- *Data*: Dit is het meest flexibele attribuut. Hier wordt de data meegegeven die hoort bij het bericht. Als het type van een bericht “subscribe” is, dan zit in het data veld een lijst met alle types waarvoor de gebruiker zich voor wilt inschrijven.
- *Sender*: Het laatste attribuut dat wordt meegegeven is de zender van het bericht. De broker moet bij een subscribe bericht kunnen achterhalen wie de zender is. Op die manier weet de broker wie moet worden toegevoegd aan de nodige lijsten.

Listing 3.1: Format voor een bericht

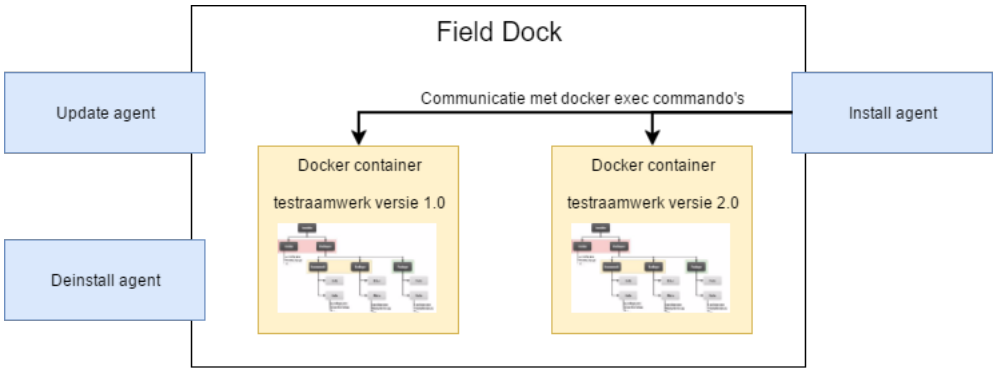
```
{
 'timestamp': 1491982212.555,
 'type': 'subscribe',
 'id': 0,
 'data': {
   'type': [
     'new',
     'change',
     'rapport'
   ]
 },
 'sender': 'localhost'
}
```

Agenten Naast de docks beschrijft de software dock architectuur ook agenten. Deze staan in voor het uitvoeren van allerlei deployment gerelateerde handelingen. Iedere agent is gekoppeld aan één stap uit de software levenscyclus die besproken werd in Sectie 2.2 op pagina 4. Hiernaast zal aan iedere installer afkomstig van het release dock een subset van alle agenten toegevoegd en verscheept worden naar het field dock. Op deze manier is het mogelijk om iedere stap in de levenscyclus van de installer te personaliseren. Ieder agent zal een bepaalde set van handelingen uitvoeren die overeenkomt met een deployment proces die besproken werd in de ORYA case studie in Sectie 2.3.4 op pagina 10. Net zoals bij ORYA wordt ieder deployment proces beschreven aan de hand van andere deployment processen en basis activiteiten. Een voorbeeld: Tijdens de creatie van een installer wordt een agent voorzien die instaat voor het installatieproces. De agent wordt samen met de installer verscheept naar het field dock waarna de agent op het gepaste moment in actie schiet. De agent begint met het hernoemen van de oude Docker container met daarin de vorige versie van een installer. Hierna wordt een nieuwe container aangemaakt waarin de nieuwe installer wordt losgelaten. Vervolgens zal de agent de installatie aanvangen en zullen de scripts

horende bij de pakketten uitgevoerd worden in de container. Zoals reeds werd aangegeven, zijn de handelingen van een agent gebaseerd op het model van ORYA. Zo wordt het creëren van een nieuwe container in de installatie agent wordt gezien als zo'n basis activiteit. In Bijlage A op pagina 45 zijn verschillende flowcharts terug te vinden die horen bij enkele types van agenten. Door agenten te gebruiken, een strategie die ook gezien werd in de Atlas case studie in Sectie 2.3.3 op pagina 8, wordt het mogelijk om alle stappen in de software levenscyclus uniek te behandelen. Hiernaast kan bij iedere installer een andere set van agenten geassocieerd worden waardoor iedere installer verder gepersonaliseerd kan worden.

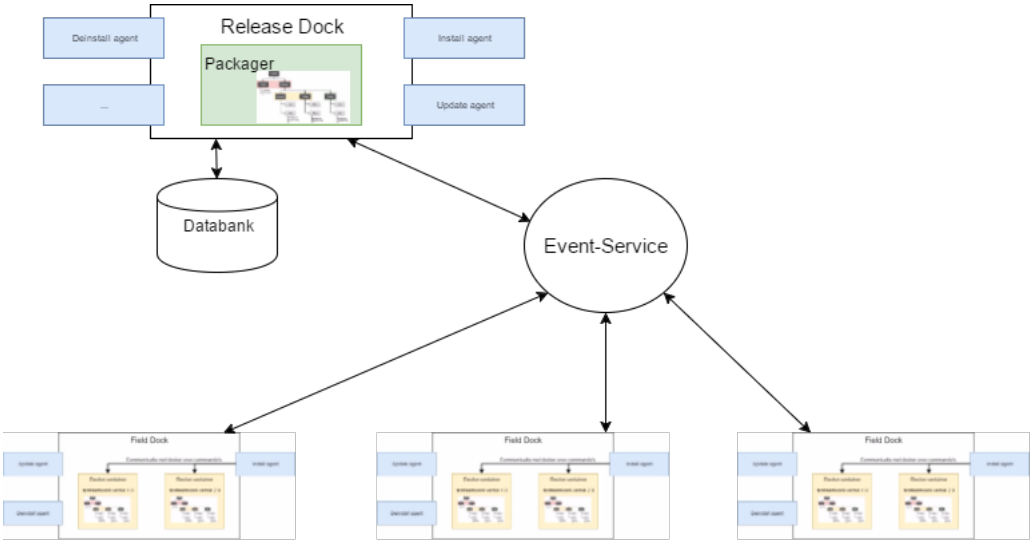
3.4.3 Deployment environment

De deployment environment komt overeen met de field dock in de software dock architectuur. In de omgeving gaat de installer, afkomstig van de packager, uitgevoerd worden zodanig dat het test framework geïnstalleerd wordt. Aan dit proces zijn de verschillende problemen verbonden die besproken werden in Sectie 2.2 op pagina 4. Om de verschillende deployment problemen te vermijden en om ervoor te zorgen dat geen uitgebreide rollback strategieën nodig zijn, wordt een geïsoleerde omgeving voorzien waarin de software geïnstalleerd wordt. Dit wordt gerealiseerd aan de hand van virtualisatie technieken, meer bepaald aan de hand van Docker. Docker wordt verkozen boven een gewone virtuele machine omdat het uitvoeren van handelingen (zoals opstarten, stoppen, . . .) op een container minder resources en tijd vraagt in vergelijking met een virtuele machine. Doordat een virtualisatie techniek wordt gebruikt, wordt het zeer eenvoudig om problemen tijdens de verschillende processen op te vangen. In Figuur A.6 op pagina 50 en Figuur A.7 op pagina 51 is het duidelijk dat, door het gebruik van Docker, het rollback proces zeer eenvoudig is. Figuur 3.5 geeft weer op welke manier de agenten met de containers gaan communiceren en geeft ook weer hoe Docker wordt geïntegreerd in het geheel.



Figuur 3.5: Structuur van een field dock

De volledige architectuur wordt weergegeven in Figuur 3.6 op de pagina hierna. In de figuur worden de verschillende docks afgebeeld samen met zowel Docker als de packager. Met deze architectuur is het mogelijk om aan de slag te gaan en een goede oplossing te vinden voor het probleem van Televic.



Figuur 3.6: Architectuur van het prototype

Hoofdstuk 4

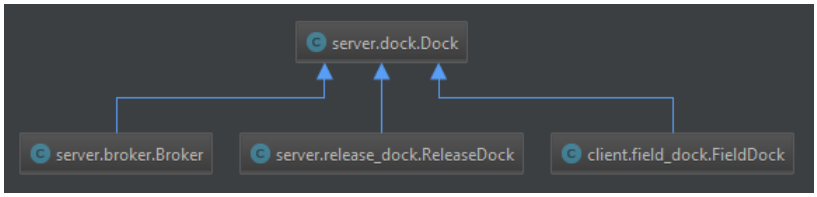
Implementatie

Na de literatuurstudie en de ontwerpfase, is het tijd om het ontwerp dat besproken werd in Hoofdstuk 3 op pagina 23 te implementeren. Als programmeertaal werd Python gekozen zodanig dat de applicatie uniform is met het testraamwerk. In de komende secties worden de verschillende implementatie beslissingen toegelicht en wordt de geschreven code toegelicht. De applicatie werd geschreven met twee doelen in het achterhoofd: de demo die gegeven gaat worden tijdens de verdediging en een basis vormen voor Televic om van te vertrekken. Het doel van de presentatie is het tonen van het installatieproces beginnende bij de creatie van een installer tot het installeren zelf in een container in een field dock. Om dit te kunnen realiseren, was het niet nodig/mogelijk om alle nodige functionaliteiten te implementeren en toe te voegen aan de applicatie. Tijdens het implementeren, werden de server-side en client-side van elkaar gescheiden en er werden twee verschillende packages aangemaakt. In wat volgt, zullen deze dan ook afzonderlijk besproken worden.

4.1 Server-side

Initialisatie De server package bevat alle logica die hoort bij het release dock als bij de broker, maar bevat ook enkele modules die gebruikt worden om de realiteit af te beelden. De flow van de applicatie begint met het opstarten van de deployment_server module. Deze module zal een release dock, een broker en een packager opstarten. Vervolgens wordt het release dock gesubscribed voor de berichten van het type new, change en rapport. Nadat de nodige services zijn opgestart, is het release dock klaar voor het afhandelen van alle nodige taken zoals het afhandelen van binnenkomende berichten.

Zowel het release dock als de broker erven eigenschappen van de klasse Dock. Dit is zichtbaar in Figuur 4.1 op de pagina hierna. Aangezien alle docks en de broker dezelfde functionaliteiten moeten hebben (openen van een socket, luisteren voor data op de socket, versturen van berichten en afhandelen van berichten), is het eenvoudig om deze functionaliteiten in de superklasse te steken. Voor de verschillende threads die gebruikt worden voor het luisteren naar nieuwe informatie maar ook voor het afhandelen Om de verscheidene threads te implementeren, werd altijd gebruik gemaakt van methodes die blocking zijn. Op deze manier wordt ervoor gezorgd dat threads geen onnodige resources gebruiken. Een soortgelijke strategie wordt toegepast voor de agenten aangezien elk type van agent een actie moet kunnen uitvoeren. De nodige methodes worden vervolgens ingevuld in de subklasse.



Figuur 4.1: Klassendiagram van Dock

Grafische User Interfaces Het is ook mogelijk om de Grafische User Interface (GUI) op te starten door middel van de “Enter” toets. Vanuit de GUI is het mogelijk om de verschillende clients te controleren en is het mogelijk om nieuwe installers te creëren. De GUI werd gemaakt met de Python bibliotheek wxPython[1] en met de hulp van wxFormBuilder[2] werd basis code gegenereerd. De nodige methodes werden vervolgens overschreven in de module overview_impl. Via deze methode wordt een gelijkaardig resultaat behaald voor het maken van de GUI voor de installer creatie.

Berichten versturen Bij het versturen van een bericht moet eerst een object aangemaakt worden van de Message klasse. Vervolgens wordt de inhoud toegevoegd aan het bericht en kan het verzonden worden naar de broker. In de broker wordt gecontroleerd welk type bericht het is om het dan vervolgens door te sturen naar alle doelen die in de gepaste list zitten. Voordat het bericht wordt doorgestuurd wordt het eerst ingepakt in een ander bericht met als type notificatie. Zo weet de ontvanger dat het bericht afkomstig is van de broker en is weet de ontvanger dat het data veld het doorgestuurde bericht bevat. Deze kan vervolgens uitgepakt worden. Afhankelijk van het type van het doorgestuurde bericht zal de correcte methode opgeroepen worden.

Installer creatie In de overzicht GUI is het mogelijk om een nieuwe installer te maken. Met behulp van de aparte GUI, die geïmplementeerd wordt door de module release_creator_impl, is het mogelijk om een installer te definiëren. Tijdens het samenstellen van de installer kunnen verscheidene nieuwe pakketten aangemaakt worden door de bovenste velden meermaals in te vullen. Hierbij is het belangrijk om te weten dat een folder moet geselecteerd worden als bestandslocatie. Alle bestanden aanwezig in de geselecteerde folder worden tijdens de creatie verplaatst naar de correcte locatie. Bij het invullen van de gegevens van de installer moet ook een folder geselecteerd worden waarin de installer moet terecht komen.

Bij het indienen van de gegevens van de installer zelf worden de nodige handelingen uitgevoerd om een correcte installer te produceren. Eerst worden de pakketten en installer toegevoegd aan de databank. Vervolgens wordt de nodige folder structuur opgebouwd zoals aangegeven werd in Figuur 3.3 op pagina 27. In de config folder wordt een leeg Dockerfile en een metadata bestand met daarin de volledige beschrijving van de installer. Tijdens het creëren van de installer moet één van de pakketten aangeduid worden als een framework pakket. Bij het toevoegen van de verschillende bestanden zal dit pakket een start_script meekrijgen. Dit is het script dat wordt uitgevoerd iedere keer als het framework wordt opgestart. Hierna worden alle pakketten overlopen. Als het pakket nieuw is, worden de nodige folders en bestanden aangemaakt anders worden de oude bestanden uit de meta folder gekopieerd. Nadat de folder structuur aangemaakt is, is het mogelijk om de verschillende scripts aan te passen. Net als bij het QT Installer Framework is het de bedoeling om te scripts te personaliseren naargelang het pakket. Op deze wijze heeft ieder pakket een uniek

[1] https://wxpython.org/
[2] https://github.com/wxFormBuilder/wxFormBuilder

installatie en test proces. De installer is hierna klaar om gereleased te worden. Van zodra in de overview GUI het release signaal gegeven wordt, schiet het tweede deel van het release proces in werking. Een install agent wordt aangemaakt en wordt toegevoegd aan de lijst van agenten horende bij de installer. De folder structuur uit het eerste deel wordt gezipt zodanig dat minder data verzonden moet worden. Als laatste deel van wordt een “release” bericht aangemaakt en verzonden naar de broker. Deze schiet vervolgens in gang en stuurt het bericht door naar de nodige field docks.

Client monitoring In de applicatie is het mogelijk om in een beperkte manier de clients te controleren. De overview GUI bevat een tabel met de verschillende clients die aanwezig zijn in de databank. Met hulp van de mysql.connector module van Python lukt het om de databank te ondervragen. Alle gegevens over de torens wordt opgevraagd en vervolgens worden er Tower objecten van gemaakt. Hierna wordt gecontroleerd of op de toren een installer aanwezig is. Mocht dit zo zijn, worden de gegevens over deze installer uit de databank gehaald. Op deze manier kan nagekeken worden welke installer aanwezig is en kan dit worden weergegeven in de GUI.

4.2 Client-side

Initialisatie Aan de client-side wordt een gelijkaardige werkwijze gebruikt als bij de server-side. De module deployment_client zal een field dock object aanmaken en de nodige services aanpassen. Tijdens het eerste gebruik van de code wordt een controle uitgevoerd waarmee gecontroleerd wordt of de description_file aanwezig is. Dit bestand bevat de volledige beschrijving van de testtoren met alle verschillende hardware componenten. Vervolgens gaat het field dock zich subscriben voor de nodige berichten bij de broker. Als beide stappen zijn afgelopen, wordt de Grafische User Interface opgestart en is het systeem klaar om gebruikt te worden.

Eerste gebruik Bij het eerste gebruik van de applicatie is de description_file nog niet aanwezig. Er is dan nog geen beschrijving van de gebruiker aanwezig aan zowel de client- als server-side. Om dit aan te pakken, wordt een GUI opgestart waarmee het mogelijk is om het volledige systeem te beschrijven. Na het invullen van de nodige informatie en het indienen van alle gegevens, wordt de description_file aangemaakt. Een “new” bericht wordt gefabriceerd met in het data veld de volledige beschrijving van het systeem. Dit bericht wordt verzonden naar de broker en de broker stuurt het bericht door naar iedereen die gesubscribed is voor de “new” berichten. Het bericht komt toe bij het release dock. Het bericht wordt uitgepakt, omgevormd naar een Tower object en vervolgens toegevoegd aan de databank.

Installatieproces Van zodra een “release” bericht binnen komt bij het field dock schiet deze in gang om het installatieproces op poten te zetten. Met de informatie uit het bericht is het field dock in staat om een Installer object te maken. Het field dock achterhaalt wie de zender van het bericht en contacteert dan direct de zender[3]. Nadat een connectie tussen de twee werd opgestart, wordt de gezipte installer samen met de nodige agenten verzonden naar het field. Dill[4] wordt gebruikt om de agent objecten te serialiseren voor ze verzonden worden naar het field dock. De lijst van agenten wordt overlopen en het install agent object zal zijn functie uitvoeren.

[3] Dit is het enige moment dat directe communicatie tussen de field en release docks mogelijk is

[4] `https://pypi.python.org/pypi/dill`

De eerste actie die de install agent gaat ondernemen is het uitpakken van het zip bestand. Met de Dockerfile uit de installer en de docker-py[5] module wordt een Docker image aangemaakt die de basis zal vormen voor de container. Voor de presentatie werd een image aangemaakt voor een Linux container waarin Python en wxPython op voorhand waren geïnstalleerd. Vervolgens wordt een container gecreëerd met als naam “fieldcontainer”. Dit is de hoofdcontainer die altijd de laatst werkende versie van het framework zal bevatten. Tijdens de creatie werden de nodige parameters doorgegeven om Grafische User Interfaces van de container op te kunnen starten. Om dit mogelijk te maken, is het nodig om aan X11 forwarding te doen. Belangrijk hierbij is dat op de host computer een X11 server moet draaien voordat dit kan plaats vinden. Voor de demonstratie werd Cygwin[6] gebruikt om de X11 server op te starten en de correcte omgeving te generen[7]. Hierna wordt het metadata bestand gelezen en gebruikt om de installatievolgorde van de pakketten te achterhalen. Elk pakket wordt in de container gekopieerd en het installatie script wordt uitgevoerd. Vervolgens gaat, met hulp van het test script, nagegaan worden of het installatieproces correct verlopen is. De output van het script wordt gecontroleerd. Als de uitgangsstatus begint met 0 wordt de test gezien als geslaagd. Van zodra dit niet het geval is, wordt de container gemerkt als slecht en zal deze op het einde van het installatieproces in quarantaine geplaatst worden. De container wordt niet verwijdert. Er werd voor deze strategie gekozen aangezien het mogelijk dat het test script slecht geschreven is. Het framework is dan correct geïnstalleerd maar wordt door de test toch gezien als foutief. Door de container in quarantaine te plaatsen blijft het mogelijk om achteraf test uit te voeren. Vervolgens wordt een rapport aangemaakt met daarin het resultaat van de test, de begin- en eindtijd. Het rapport wordt toegevoegd aan een bericht en wordt verzonden naar de broker die het doorstuurt.

Framework opstarten Na het opstarten van alle services verschijnt de Grafische User Interface van de client. Van hieruit is het mogelijk om het pakket dat gemarkeerd staat als framework te openen. De “fieldcontainer” wordt opgestart en het start script wordt uitgevoerd waardoor het framework zal opstarten.

[5] `https://docker-py.readthedocs.io/en/stable/`

[6] `https://www.cygwin.com/`

[7] De tutorial `https://manomarks.net/2015/12/03/docker-gui-windows.html` werd hiervoor gevolgd

Hoofdstuk 5

Evaluatie

Na de ontwerp- en implementatiefase was het tijd om de gecreëerde applicatie te beoordelen en testen. In het komende hoofdstuk is het dan ook de bedoeling om de applicatie te onderwerpen aan verschillende tests die enkele aspecten in de applicatie gaan testen. Hierbij is het de bedoeling om te achterhalen wat wel en niet mogelijk is met de huidige versie van de applicatie. Na het testen van de applicatie, wordt nagedacht voor mogelijke uitbreidingen voor de applicatie.

5.1 Testen

5.1.1 Cross-platform host

Doel Het doel van deze test is het uittesten hoeverre de geschreven applicatie uitvoerbaar is op verschillende besturingssystemen. De applicatie zelf is volledig geschreven in Python met modules die cross-platform zijn.

Scenario Om uit te testen of de applicatie cross-platform is, wordt een virtuele machine aangemaakt met de nodige programma's. Als guest-besturingssysteem voor deze virtuele machine, wordt Linux Mint gekozen. Docker, Mysql workbench en wxPython worden geïnstalleerd en met hulp van pip worden de docker-py en dill modules aan python toegevoegd waardoor het systeem klaar is om getest te worden. De databank werd aangemaakt en de code werd overgebracht naar de virtuele machine.

Tijdens de test is het de bedoeling om alle functionaliteiten van de applicatie uit te testen. De verschillende docks moeten correct subscriben bij de broker en de nieuwe field dock moet zonder fouten worden toegevoegd aan de databank. Vervolgens wordt getest of een installer kan aangemaakt worden, verscheept worden naar de field dock en geïnstalleerd kan worden. Er wordt een installer aangemaakt die bestaat uit 3 verschillende pakketten: één framework pakket, één optioneel pakket en één niet-optioneel pakket. De twee niet-framework pakketten bevatten een eenvoudig script die output genereert. Het framework pakket bevat een Grafische User Interface gemaakt met wxPython.

De test is geslaagd als alle verschillende handelingen in de goede volgorde overlopen, ondertussen geen fouten optreden en op het einde het framework opgestart kan worden.

Uitvoering

5.1.2 Cross-platform containers

Doel Naast het controleren of de applicatie zelf cross-platform is, is het belangrijk om te achterhalen of de applicatie om kan met Windows containers. Het doel van deze test is dan ook om een installer te creëren specifiek bedoelt voor deze Windows containers.

Scenario Om gebruik te kunnen maken van Windows containers, moeten enkele vereisten voltooid worden. Met hulp van Friis [42] is het mogelijk om een field dock op te zetten die om kan gaan met de Windows containers. De testomgeving bestaat, naast de enkele field dock, verder nog uit één release dock. Deze wordt gebruikt om een installer te creëren die bestaat uit alle nodige pakketten om het Python testraamwerk werkende te krijgen. Het framework pakket zal dan ook bestaan uit het daadwerkelijke testraamwerk. Op deze wijze is het mogelijk voor Televic om te achterhalen welke stappen allemaal gezet zijn en welke nog moeten gezet worden om de applicatie zo snel mogelijk in gebruik te nemen.

Uitvoering

5.1.3 Meerdere clients

Doel Voor deze test is het de bedoeling om te achterhalen of de applicatie meerdere clients kan ondersteunen. Het doel is dan ook een omgeving op te zetten waarin verschillende clients aanwezig zijn.

Scenario De omgeving die gebruikt gaat worden bestaat uit één laptop die zowel gebruikt wordt als release dock en als field dock en twee andere computers die gebruikt worden als field docks. Op de ieder computer is Docker, Python, wxPython, docker-py en dill aanwezig zodat de geschreven code bruikbaar is.

Tijdens de test wordt op één release dock een installer aangemaakt en klaargemaakt voor release. De installer bestaat uit 2 verschillende pakketten: één framework pakket en één optioneel pakket. Beide pakketten zullen bestaan uit enkele scripts die output zullen generen wanneer deze worden uitgevoerd. Naast de release dock zullen 3 field docks toegevoegd worden aan het netwerk.

Om een geslaagde test te verkrijgen, moet de installer bij alle field docks toekomen en correct geïnstalleerd worden. Dit moet op het einde van het proces zichtbaar zijn in zowel het overzicht op de release dock als in de databank.

Uitvoering Na het instellen van de nodige parameters, was het tijd om de nodige systemen op te starten. Het duurde dan ook niet lang om drie torens te beschrijven die elk bestaan uit één hardware component. Deze opstellingen werden vervolgens naar het release dock gestuurd en opgeslagen in de databank.

De volgende stap bestaat uit het beschrijven van een simpele installer bestaande uit twee pakketten en deze klaar te maken voor release.

5.1.4 Meerdere servers

Doel Een volgende test bestaat uit het testen van de functionaliteiten als meerdere servers aanwezig zijn. Het doel van de test is te achterhalen wat er gebeurd als meerdere servers aanwezig zijn.

Scenario Na het testen of het mogelijk is om meerdere clients te ondersteunen, wordt nu getest of meerdere servers ondersteunt wordt. Dit gebeurt wederom aan de hand van een virtuele machine met Windows 10. Iedere virtuele machine bevat de server code samen met een databank om alle gegevens in op te slaan.

Gedurende de test worden 3 release docks aangemaakt en één field dock. In het eerste deel van de test zal iedere release dock apart een installer creëren en deployen. De installers zullen bestaan uit één pakket, namelijk een framework pakket. Het pakket bevat een simpel script dat uitgevoerd wordt. Voor ieder release dock zal dit script lichtjes verschillen. Het tweede deel van de test bestaat uit het releasen van 3 verschillende installers op eenzelfde moment.

Om de test te doen slagen is het nodig dat: iedere installer moet goed toekomen bij het field dock, de installers moeten in de correcte volgorde geïnstalleerd worden, alle databanken moeten geüpdatet worden en in de verschillende aanpassingen moeten in volgorde toegevoegd worden aan de databank.

Uitvoering Met de huidige versie van de applicatie is het mogelijk om drie verschillende servers op te starten en te verbinden met de broker. Hierbij moet wel rekening gehouden worden met het feit dat het deployment_server script naast een release dock ook een broker opstart. Om dit niet te gebruiken moet een boolean op false gezet worden zodanig dat deze code niet wordt uitgevoerd.

Een eerste fase uit de test bestaat uit het correct configureren van een client in het systeem. Deze wordt correct uitgevoerd en na de beschrijving van de toren worden alle nodige databases aangepast. Vervolgens wordt op één van de release docks een installer aangemaakt die bestaat uit een simpel script die output genereert.

5.1.5 Slecht werkend pakket

Doel De geschreven applicatie bevat enkele methodes om te controleren of het installatieproces foutloos is verlopen. Mocht dit niet het geval zijn, dan wordt de foutieve container in quarantaine geplaatst. Het doel van deze test is het uittesten of dit wel degelijk gebeurt.

Scenario Om deze eigenschap van de applicatie uit te testen wordt een simpele omgeving gerealiseerd waarin getest kan worden. Eén release dock en field dock worden opgestart in het begin. Vervolgens wordt een installer gemaakt die bestaat uit twee pakketten. Het eerste pakket wordt gebruikt om een bestand aan te maken dat nodig is voor het framework pakket. Het eerstgenoemde pakket zal weliswaar geen bestand aanmaken.

Om de test te laten slagen, moet de test van het niet-framework pakket registeren dat het bestand niet is aangemaakt. Op het einde van het installatieproces moet de container in quarantaine geplaatst worden. Zo is het mogelijk om na het installatieproces te controleren wat het probleem is in de container.

Uitvoering

5.1.6 Netwerk monitoring

Doel Als laatste test wordt nagegaan hoe het netwerkverkeer eruit ziet tijdens het releasen van een nieuwe installer.

Scenario Deze test wordt gebruikt om te achterhalen hoeveel en wat voor netwerkverkeer gecreëerd wordt door de applicatie. Het monitoren van het netwerk wordt uitgevoerd tijdens de meerdere client test. De test produceert voldoende netwerk verkeer om een idee te krijgen van de hoeveelheid verkeer die zal geproduceerd worden. De volledige communicatie tussen de verschillende docks wordt opgenomen met Wireshark voor later een analyse op uit te voeren.

Uitvoering

5.2 Uitbreidingen

Hoofdstuk 6

Conclusie

Referenties

[1] R. S. Hall, D. Heimbigner en A. L. Wolf, „A cooperative approach to support software deployment using the software dock”, in *Software Engineering, 1999. Proceedings of the 1999 International Conference on*, IEEE, 1999, pp. 174–183.

[2] A. Dearie, „Software Deployment, Past, Present and Future”, in *Future of Software Engineering, 2007. FOSE '07*, mei 2007, pp. 269–284. DOI: 10.1109/FOSE.2007.20.

[3] A. Carzaniga, A. Fuggetta, R. S. Hall, D. Heimbigner, A. Van Der Hoek en A. L. Wolf, „A characterization framework for software deployment technologies”, DTIC Document, tech. rap., 1998.

[4] E. Dolstra, *The purely functional software deployment model*. Utrecht University, 2006.

[5] C. Fletcher, D. P. Williams en L. F. Wurster, *Magic Quadrant for Application Release Automation*, https://www.gartner.com/doc/reprints?id=1-3DSWYP2&ct=160801&st=sb, [Online; geraadpleegd 22-04-2017], 2016.

[6] E. Cloud, *ElectricFlow 7.2 API Guide*, http://docs.electric-cloud.com/eflow_doc/7_2/API/PDF/APIflow_7_2.pdf, [Online; geraadpleegd 22-04-2017], 2016.

[7] E. C. Bailey, *Maximum rpm*, 1997.

[8] CERN, *The Standard Model*, http://home.cern/about/physics/standard-model, [Online; geraadpleegd 22-04-2017], 2017.

[9] ——, *About the ATLAS Experiment*, https://atlas.cern/discover/about, [Online; geraadpleegd 22-04-2017], 2017.

[10] I. Bird, K. Bos, N. Brook, D. Duellmann, C. Eck, I. Fisk, D. Foster, B. Gibbard, M. Girone, C. Grandi e.a., „LHC computing Grid”, *Technical design report*, p. 8, 2005.

[11] A. Salvo, A. Barchiesi, K. Gnanvo, C. Gwilliam, J. Kennedy, G. Krobath, A. Olszewski en G. Rybkine, „The ATLAS software installation system for LCG/EGEE”, in *Journal of Physics: Conference Series*, IOP Publishing, deel 119, 2008, p. 052 013.

[12] E. Obreshkov, S. Albrand, J. Collot, J. Fulachier, F. Lambert, C. Adam-Bourdarios, C. Arnault, V. Garonne, D. Rousseau, A. Schaffer, H. von der Schmitt, A. D. Salvo, V. Kabachenko, Z. Ren, D. Qing, E. Nzuobontane, P. Sherwood, B. Simmons, S. George, G. Rybkine, S. Lloyd, A. Undrus, S. Youssef, D. Quarrie, T. Hansl-Kozanecka, F. Luehring, E. Moyse en S. Goldfarb, „Organization and management of {ATLAS} offline software releases”, *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, deel 584, nr. 1, pp. 244–251, 2008, ISSN: 0168-9002. DOI: http://dx.doi.org/10.1016/j.nima.2007.10.002. adres: http://www.sciencedirect.com/science/article/pii/S0168900207020967.

[13] *CMT configuration management tool*, http://www.cmtsite.org/, [Online; geraadpleegd 20-04-2017], 2016.

[14] G. Rybkin, „ATLAS software packaging", *Journal of Physics: Conference Series*, deel 396, nr. 5, p. 4, 2012.

[15] V. Lestideau en N. Belkhatir, „Providing highly automated and generic means for software deployment process", in *European Workshop on Software Process Technology*, Springer, 2003, pp. 128–142.

[16] N. Belkhatir, J. Estublier en W. Melo, „THE ADELE-TEMPO experience: an environment to support process modeling and enaction.", *Software Process Modelling and Technology Research Studies Press*, pp. 1–37, 2007.

[17] *Whitepaper: Ansible in depth*, https://cdn2.hubspot.net/hub/330046/file-480366556-pdf/pdf_content/Ansible_in_Depth.pdf?t=1487567092458, [Online; geraadpleegd 10-04-2017], 2016.

[18] S. M. Srinivasan, S. Kandula, C. R. Andrews, Y. Zhou e.a., „Flashback: A Lightweight Extension for Rollback and Deterministic Replay for Software Debugging.", in *USENIX Annual Technical Conference, General Track*, Boston, MA, USA, 2004, pp. 29–44.

[19] J. S. Plank, M. Beck, G. Kingsley en K. Li, *Libckpt: Transparent checkpointing under unix*. Computer Science Department, 1994.

[20] E. N. Elnozahy, L. Alvisi, Y.-M. Wang en D. B. Johnson, „A survey of rollback-recovery protocols in message-passing systems", *ACM Computing Surveys (CSUR)*, deel 34, nr. 3, pp. 375–408, 2002.

[21] Y. Huang, C. Kintala, N. Kolettis en N. D. Fulton, „Software rejuvenation: Analysis, module and applications", in *Fault-Tolerant Computing, 1995. FTCS-25. Digest of Papers., Twenty-Fifth International Symposium on*, IEEE, 1995, pp. 381–390.

[22] S. Shumate, *Implications of Virtualization for Image Deployment*, 2004. adres: http://www.dell.com/downloads/global/power/ps4q04-20040152-Shumate.pdf.

[23] J. De Jesús, *How, where, and why IBM PureApplication fits in your cloud*, https://www.ibm.com/developerworks/websphere/techjournal/1506_dejesus/1506_dejesus-trs.html, [Online; geraadpleegd 22-04-2017], 2015.

[24] M. Fenn, M. A. Murphy, J. Martin en S. Goasguen, „An evaluation of KVM for use in cloud computing", in *Proc. 2nd International Conference on the Virtual Computing Initiative, RTP, NC, USA*, 2008.

[25] *Docker Main Page*, https://www.docker.com/, [Online; geraadpleegd 16-08-2016], 2016.

[26] Docker, „Docker for the Virtualization Admin", 2016.

[27] R. Chamberlain en J. Schommer, „Using Docker to support reproducible research", *DOI: http://dx. doi. org/10.6084/m9. figshare*, deel 1101910, 2014.

[28] D. Merkel, „Docker: lightweight linux containers for consistent development and deployment", *Linux Journal*, deel 2014, nr. 239, p. 2, 2014.

[29] H. Tian, X. Zhao, Z. Gao, T. Lv en X. Dong, „A Novel Software Deployment Method Based on Installation Packages", in *2010 Fifth Annual ChinaGrid Conference*, jul 2010, pp. 228–233. DOI: 10.1109/ChinaGrid.2010.32.

[30] D. A. Patterson, „The data center is the computer”, *Communications of the ACM*, deel 51, nr. 1, pp. 105–105, 2008.

[32] R. S. Hall, D. Heimbigner, A. Van Der Hoek en A. L. Wolf, „An architecture for post-development configuration management in a wide-area network”, in *Distributed Computing Systems, 1997., Proceedings of the 17th International Conference on*, IEEE, 1997, pp. 269–278.

[33] P. R. Pietzuch en J. M. Bacon, „Hermes: A distributed event-based middleware architecture”, in *Distributed Computing Systems Workshops, 2002. Proceedings. 22nd International Conference on*, IEEE, 2002, pp. 611–618.

[34] A. Carzaniga, D. S. Rosenblum en A. L. Wolf, „Design and evaluation of a wide-area event notification service”, *ACM Transactions on Computer Systems (TOCS)*, deel 19, nr. 3, pp. 332–383, 2001.

[35] J. R. Callahan, „Software packaging”, tech. rap., 1998.

[36] *WiX Toolset*, `http://wixtoolset.org/`, [Online; geraadpleegd 3-08-2016], 2016.

[37] B. Amstadt en M. K. Johnson, „Wine”, *Linux Journal*, deel 1994, nr. 4es, p. 3, 1994.

[38] H. Liefke en D. Suciu, „XMill: An Efficient Compressor for XML Data”, in *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, reeks SIGMOD '00, Dallas, Texas, USA: ACM, 2000, pp. 153–164, ISBN: 1-58113-217-4. DOI: `10.1145/342009.335405`. adres: `http://doi.acm.org/10.1145/342009.335405`.

[39] *NSIS Main page*, `http://nsis.sourceforge.net/Main_Page`, [Online; geraadpleegd 3-08-2016], 2016.

[40] *Chocolatey About page*, `https://chocolatey.org/about`, [Online; geraadpleegd 3-08-2016], 2016.

[41] *QT Installer Framework Documentation*, `http://doc.qt.io/qtinstallerframework/`, [Online; geraadpleegd 4-08-2016], 2016.

[42] M. Friis, *Build and run your first docker windows server container*, `https://blog.docker.com/2016/09/build-your-first-docker-windows-server-container/`, [Online; geraadpleegd 10-04-2017], 2016.

[45] *WiX Toolset Tutorial*, `https://www.firegiant.com/wix/tutorial/`, [Online; geraadpleegd 3-08-2016], 2016.

[46] *Chocolatey Main page*, `https://chocolatey.org/`, [Online; geraadpleegd 3-08-2016], 2016.

[48] M. J. Rutherford, K. Anderson, A. Carzaniga, D. Heimbigner en A. L. Wolf, „Reconfiguration in the Enterprise JavaBean component model”, in *International Working Conference on Component Deployment*, Springer, 2002, pp. 67–81.

Bibliografie

[31] J. Münch, O. Armbrust, M. Kowalczyk en M. Sotó, *Software process definition and management*. Springer Science & Business Media, 2012.

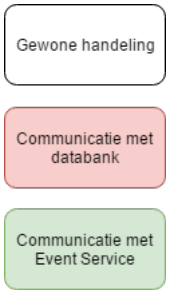
[43] R. S. Pressman, *Software engineering: a practitioner's approach*. Palgrave Macmillan, 2005.

[44] D. Pitts, B. Ball e.a., *Red Hat Linux*. Sams, 1998.

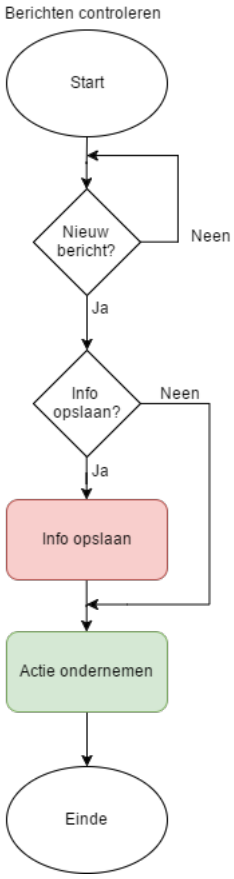
[47] G. S. Machado, F. F. Daitx, W. L. da Costa Cordeiro, C. B. Both, L. P. Gaspary, L. Z. Granville, C. Bartolini, A. Sahai, D. Trastour en K. Saikoski, „Enabling rollback support in IT change management systems", in *Network Operations and Management Symposium, 2008. NOMS 2008. IEEE*, IEEE, 2008, pp. 347–354.

Bijlage A

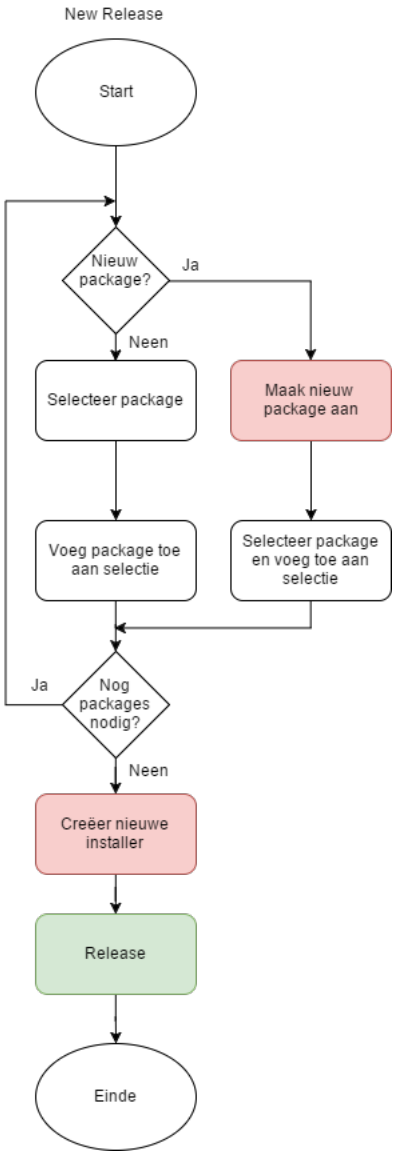
Flowcharts



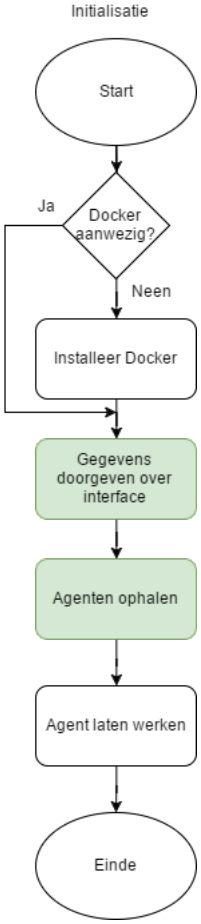
Figuur A.1: Flowchart kleurenlegende



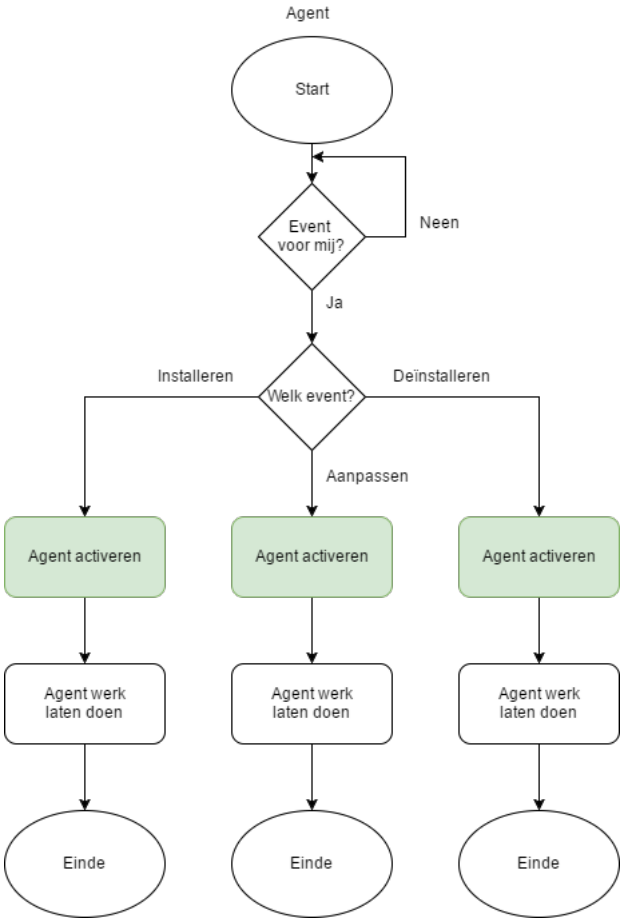
Figuur A.2: Flowchart voor het ophalen/controleren van berichten



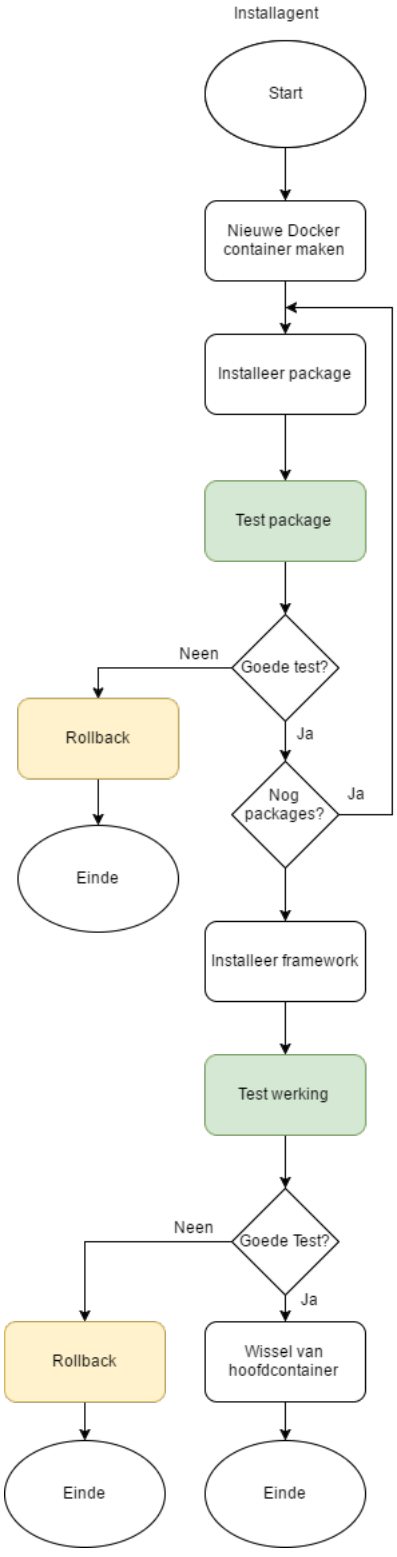
Figuur A.3: Flowchart van het creëren van een nieuwe release



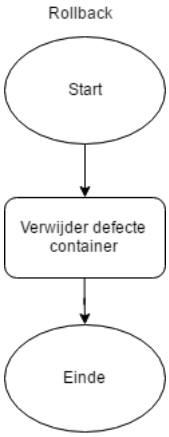
Figuur A.4: Flowchart van de initialisatie van een field dock



Figuur A.5: Flowchart van een agent



Figuur A.6: Acties van de installagent



Figuur A.7: Rollback

Bijlage B

Technologieën

Listing B.1: WiX Toolset installer

```
<?xml version="1.0"?>
<Wix xmlns="http://schemas.microsoft.com/wix/2006/wi">
   <Product Id="*" UpgradeCode="12345678-1234-1234-1234-111111111111"
            Name="Python Framework Installer" Version="0.0.1"
               ↪ Manufacturer="PJ Industries" Language="1033">
      <Package InstallerVersion="200" Compressed="yes" Comments="Windows
          ↪  Installer Package"/>
      <Media Id="1" Cabinet="product.cab" EmbedCab="yes"/>

      <Directory Id="TARGETDIR" Name="SourceDir">
          <Directory Id="PersonalFolder">
          <Directory Id="INSTALLDIR" Name="Example">
           <Component Id="ApplicationFiles" Guid
               ↪ ="12345678-1234-1234-1234-222222222222">
              <RegistryKey Root="HKCU"
                                        Key="Software\My Application\
                                            ↪ Uninstall">
              <RegistryValue Value="testFile."
                                          Type="string"
                                          KeyPath="yes" />
              </RegistryKey>
                <File Id="ApplicationFile1" Source="python-2.7.3.msi
                    ↪ "/>
                <File Id="ApplicationFile2" Source="pyusb-1.0.0a2.zip
                    ↪ "/>
                <RemoveFolder Id='INSTALLDIR' On='uninstall' />
           </Component>
          </Directory>
          </Directory>
       </Directory>

      <Feature Id="DefaultFeature" Level="1">
```

```
            <ComponentRef Id="ApplicationFiles"/>
        </Feature>

        <InstallExecuteSequence>
            <Custom Action='FooAction' After='InstallFiles'/>
        </InstallExecuteSequence>
    </Product>
</Wix>
```

Listing B.2: NSIS installer

```
# define installer name
OutFile "installer0.2.0.exe"

# set desktop as install directory
InstallDir $PROFILE\NsisExample

# default section start
Section

# define output path
SetOutPath $INSTDIR

# specify file to go in output path
File python-2.7.3.msi
File pyusb-1.0.0a2.zip

# define uninstaller name
WriteUninstaller $INSTDIR\uninstaller.exe


#--------
# default section end
SectionEnd

Section

ExecWait '"msiexec" /i "$INSTDIR\python-2.7.3.msi" /quiet'

SectionEnd

Section

ZipDLL::extractall "$INSTDIR\pyusb-1.0.0a2.zip" "$INSTDIR"

ExecWait 'cmd /K "cd "$INSTDIR\pyusb-1.0.0a2" && "C:\Python27\python.exe
    ↪ " "setup.py" install && exit"'
```

```
SectionEnd

# create a section to define what the uninstaller does.
# the section will always be named "Uninstall"
Section "Uninstall"

ExecWait '"msiexec" /i "$INSTDIR\python-2.7.3.msi"'

# Always delete uninstaller first
Delete $INSTDIR\uninstaller.exe

# now delete installed file
Delete $INSTDIR\python-2.7.3.msi
Delete $INSTDIR\pyusb-1.0.0a2.zip

RMDir /r $INSTDIR\pyusb-1.0.0a2

SectionEnd
```

Bijlage C

Beschrijving van deze masterproef in de vorm van een wetenschappelijk artikel

Bijlage D

Poster

KU LEUVEN

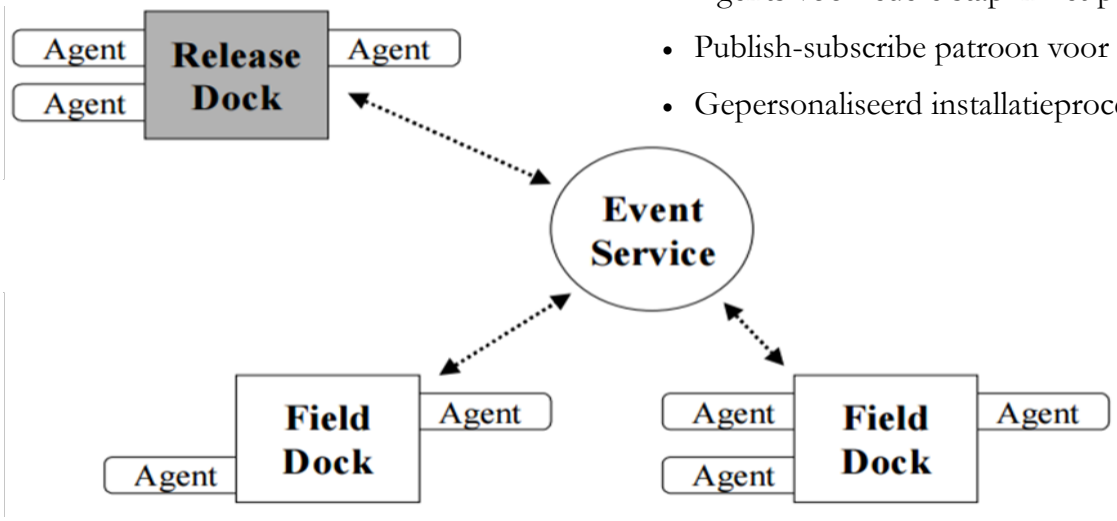
Ontwerp en ontwikkeling van een testraamwerk installer

Doel

- Installatieproces vereenvoudigen
- Schaalbare oplossing zoeken
- Client monitoring
- Prototype ontwerpen

Architectuur

- Software dock architectuur
- Agents voor iedere stap in het proces
- Publish-subscribe patroon voor communicatie
- Gepersonaliseerd installatieproces

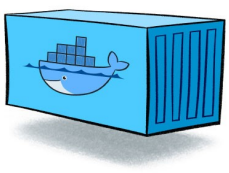


Release Dock

Server-side

- Software packaging
- Installer creatie
- Remote field dock monitoring

Field Dock



Client-side

- Docker containers
- Veilige deployment omgeving

Promotor: Dr. ir. Annemie Vorstermans
Co-Promotor: Carl Eeckhout (Televic)
Opleiding: MELICTI

Naam: Pieter-Jan Robrecht
Jaar: 2016-2017
Opmaak: 11/04/'17

python

docker

FACULTEIT INDUSTRIELE INGENIEURSWETENSCHAPPEN
TECHNOLOGIECAMPUS GENT
Gebroeders De Smetstraat 1
9000 GENT, België
tel. + 32 92 65 86 10
fax + 32 92 25 62 69
iiw.gent@kuleuven.be
www.iiw.kuleuven.be