KU LEUVEN

FACULTEIT INDUSTRIELE
INGENIEURSWETENSCHAPPEN

TECHNOLOGIECAMPUS GENT

Ontwerp en ontwikkeling van een testraamwerk installer

Pieter-Jan ROBRECHT

Promotor: Annemie Vorstermans

Co-promotor: Carl Eeckhout

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 Communicatie Technologies

Academiejaar 2016 - 2017

Dankwoord

Dank aan mezelf
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

Situering

Televic Rail heeft een Python test framework ontworpen waarmee zij in staat zijn 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 een spilfiguur voor het afleveren van producten die voldoen aan strenge veiligheidsnormen.

1.1 Het probleem

De applicatie gebruikt verschillende Python bibliotheken en hardware drivers om correct te functioneren. Het installatieproces is tijdrovend en foutgevoelig. Bij het uitbrengen bij een nieuwe versie van de applicatie, bijvoorbeeld bij het uitbrengen van een nieuwe driver, bibliotheek of om nieuwe producten te ondersteunen, moet de applicatie geüpdatet worden. Dit proces lijdt aan dezelfde gebreken als het installatieproces. Het doel van deze thesis is zoeken van een langdurige oplossing voor dit probleem. Na een onderzoek zal een prototype geproduceerd worden die als oplossing aangeboden wordt aan het bedrijf.

Een probleemanalyse onthulde al snel dat dit probleem onder te verdelen is in verschillende deelproblemen. Het framework bestaat uit verschillende componenten, hieronder vallen de drivers en bibliotheken. Elke component heeft een unieke installatiewijze en de componenten moeten in een specifieke volgorde geïnstalleerd worden. Hiernaast moeten verscheidene componenten geconfigureerd worden aan de hand van een configuratiebestand. Dit configuratiebestand is doelsysteem specifiek. Tijdens het onderzoek zal er onderzocht welke technologieën gebruikt zal worden om al deze componenten te combineren tot één uitvoerbaar bestand. Bij het selectieproces moet er rekening gehouden worden met de toekomst. Er wordt best een technologie gekozen die cross-platform is, zodanig dat het bedrijf niet beperkt wordt in de toekomst.

De probleemanalyse onthulde ook dat de verschillende executables verspreidt moeten worden. Door dit proces te automatiseren, is het mogelijk om waardevolle informatie te verzamelen. Hiermee zouden rapporten gegeneerd kunnen worden voor het bedrijf. Tijdens het ontwerpen zal schaalbaarheid een belangrijke factor spelen. Naar de toekomst toe zal het aantal systemen waarop de applicatie geïnstalleerd wordt toenemen.

Zoals reeds hierboven vermeld, is het installatieproces foutgevoelig. Na de foutanalyse bleek dat hiervoor een schaalbare oplossing voor gevonden moet worden. Bij het finaliseren van een component installatie, moeten de functionaliteiten van de component gecontroleerd worden. Mocht

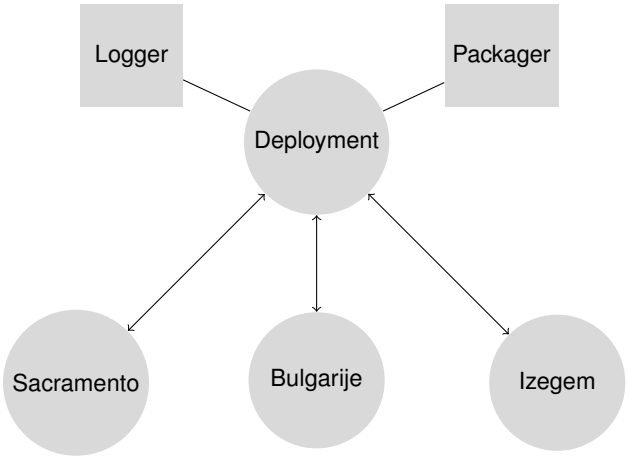
deze incorrect functioneren dan moet een gepaste actie, eventueel een rollback, gebeuren. Ook na de volledige installatie moet gecontroleerd worden of het geheel correct functioneert. Hierdoor wordt er voorkomen dat er nodeloos veel tijd verloren gaat in het herinstalleren van de applicatie. Robuustheid zal een belangrijke component zijn tijdens de ontwerpfase. Bij het ontwerpen moet gebruiksvriendelijkheid in het achterhoofd gehouden worden.

1.2 Werkwijze

In Sectie 1.1 werd opgelijst wat de problemen zijn. In wat volgt zullen verschillende oplossingswijzes opgesomd worden met vervolgens de werkwijze.

Na de probleemanalyse werd het al snel 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 een packager met als doel het inpakken van de nodige drivers, bibliotheken, ... Naast de packager is er een deployment server nodig die instaat voor het verspreiden van de installers die de packager aflevert. Dankzij het automatiseren van de deployment, kan het installatieproces per client getrackt worden. Om deze stap te vereenvoudigen, moet aan de client-side een deployment environment voorzien worden. In die omgeving wordt de installatie geïsoleerd. Mocht een rollback nodig zijn, dan kan deze op een eenvoudige manier gebeuren.

Gedurende deze thesis wordt een prototype ontworpen. In Figuur 1.1 wordt de algemene structuur van de applicatie weergegeven. Met behulp van deze basis is het mogelijk om een demo te produceren die de verschillende problemen verhelpt.



Figuur 1.1: Overzichtsdiagram van de algemene structuur

Hoofdstuk 2

Bespreking

In dit hoofdstuk zal de structuur die in Hoofdstuk 1 op pagina 1 werd opgebouwd, verder uitgebouwd worden. In de loop van dit hoofdstuk zullen de verschillende technologieën besproken worden die in aanmerking komen voor de verschillende componenten. Naar het einde van het hoofdstuk toe, zal een keuze gemaakt worden en zal voor iedere component de meest geschikte technologie gekozen worden.

2.1 Packager

De eerste component die besproken wordt is de packager. Zoals reeds besproken, moet tijdens het ontwerpen van deze component rekening gehouden worden met enkele eigenschappen. Alvorens bepaald wordt welke technologie geschikt is, moet onderzocht worden wat ingepakt moet worden en op welke manier.

Tian, Zhao, Gao, Lv en Dong (2010) haalt drie methodes aan om software te deployen: disk image-based deployment, behavior-based deployment en 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. 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 e.a. (2010) al aangeeft, deze methode is niet flexibel. Voor de applicatie is flexibiliteit een must. Iedere node bevat verschillende hardware en is verschillende geconfigureerd. Een disk image-based deployment zal hierdoor niet gebruikt worden. Het basis idee achter behavior-based deployment is het opnemen van de schijf operaties tijdens het deployen. Als geweten is welke bestanden aangepast zijn, gecreëerd zijn, ... dan kan het proces nagebootst worden op andere nodes (Tian e.a., 2010). Zo een proces nabootsen is moeilijk. Kernel operaties 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 e.a. (2010) 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. Dit is een strategie die bruikbaar is voor het probleem op te lossen. Voor ieder pakket kan een specifieke batch file gemaakt worden. Deze strategie biedt de flexibiliteit die nodig is voor het oplossen van het probleem.

Voordelen	Nadelen
Diepe integratie met Windows	XML structuren zorgt voor veel schrijfwerk
Mogelijkheid om externe executables te includeren	Niet cross-platform

Tabel 2.1: Voor- en nadelen WiX Toolset

Om een correct werkende applicatie te hebben, heeft het Python raamwerk verschillende drivers en bibliotheken nodig. Obreshkov e.a. (2008) bespreekt hoe het ATLAS project te werk gaat bij het inpakken van alle nodige software. Het ATLAS project gebruikt CMT als configuratie manager. Met behulp van een configuratie bestand weten verscheidene tools hoe ze een pakket moeten afhandelen (Obreshkov e.a., 2008). Een gelijkaardige structuur is wenselijk voor het probleem. Net zoals Obreshkov e.a. (2008) zou er voor ieder bibliotheek en driver een pakket voorzien worden. Hierdoor wordt de herbruikbaarheid bevordert en kan voor ieder pakket apart behandelt worden. Rybkin (2012) 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.

2.1.1 Technologieën

In voorbereiding op het academiejaar, was het mogelijk om een stage te lopen bij het bedrijf. Het doel van deze stage was het voorbereiden van de thesis. Tijdens deze voorbereiding, werd er vooral gezocht naar een goede technologie die gebruikt kon worden voor de packager. In wat volgt, gaat er een opsomming volgen van de onderzochte technologieën samen met hun voor- en nadelen.

WiX Toolset

Windows installer XML Toolset is een set van build tools waarmee Windows Installer packages gemaakt kunnen worden. Bron code wordt gecompileerd en vervolgens gelinkt om een executable te maken. De toolset zal .msi installatie pakketten, .msm merge modules en .msp patches combineren („WiX Toolset", 2016).

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. In de tabel wordt vermeld dat WiX een diepe integratie heeft met Windows. Dit komt doordat WiX installers produceert voor de Windows Installer, het Windows installation engine. Zulke installers worden gemaakt aan de hand van XML bestanden die de verschillende elementen definiëren („WiX Toolset Tutorial", 2016). Liefke en Suciu (2000) haalt verschillende eigenschappen aan van XML waar rekening gehouden moet worden. XML is niet het meest efficiënte dataformaat maar biedt een verhoogde flexibiliteit aan.

NSIS

Nullsoft Scriptable Install System is een open source systeem waarmee Windows installers gemaakt kunnen worden. Zoals de naam aangeeft, is NSIS script-based waardoor installers logica

Voordelen	Nadelen
Scripting taal	Niet cross-platform
Verschillende plug-ins beschikbaar	Geen structuur voor packages

Tabel 2.2: Voor- en nadelen NSIS

Voordelen	Nadelen
Volledige deployment infrastructuur al aanwezig	Niet cross-platform
	Command-line tool

Tabel 2.3: Voor- en nadelen Chocolatey

bevatten waarmee verschillende installatie taken behandelt kunnen worden. Hiernaast bestaan er al verschillende plug-ins en scripts voor verschillende taken af te handelen („NSIS Main page", 2016).

Aan deze technologie zijn verschillende plus- en minpunten verbonden. Een vergelijking van de verschillende voor- en nadelen is terug te vinden in Tabel 2.2. Aangezien NSIS een scripting taal gebruikt om een installer te definiëren, is het zeer eenvoudig om een eenvoudige installer te creëren. Gecombineerd met de grote hoeveelheid aan plug-ins die aanwezig zijn door de community, wordt het eenvoudig om een installer te produceren die het probleem kan oplossen. De tool ondersteunt geen structuur waarmee verschillende pakketten gedefinieerd kunnen worden. Zoals WiX Toolset is deze technologie uitsluitend voor Windows.

Chocolatey

Volgens „Chocolatey About page" (2016) 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. Het werd gebouwd boven op het NuGet infrastructuur en gebruikt PowerShell als zijn focus voor het verspreiden van de packages.

Aan deze technologie zijn verschillende plus- en minpunten verbonden. Een vergelijking van de verschillende voor- en nadelen is terug te vinden in Tabel 2.3. 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.

Voordelen	Nadelen
Cross-platform	XML structuren zorgt voor veel schrijfwerk
Mogelijkheid om externe executables te includeren	Enkel Linux installer maken in Linux

Tabel 2.4: Voor- en nadelen Qt Installer Framework

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 („QT Installer Framework Documentation", 2016).

Aan deze technologie zijn verschillende plus- en minpunten verbonden. Een vergelijking van de verschillende voor- en nadelen is terug te vinden in Tabel 2.4. 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.

Schrijven over het gebruiken van Windows .exe in Linux? Bijvoorbeeld hoe Wine misschien een oplossing is?

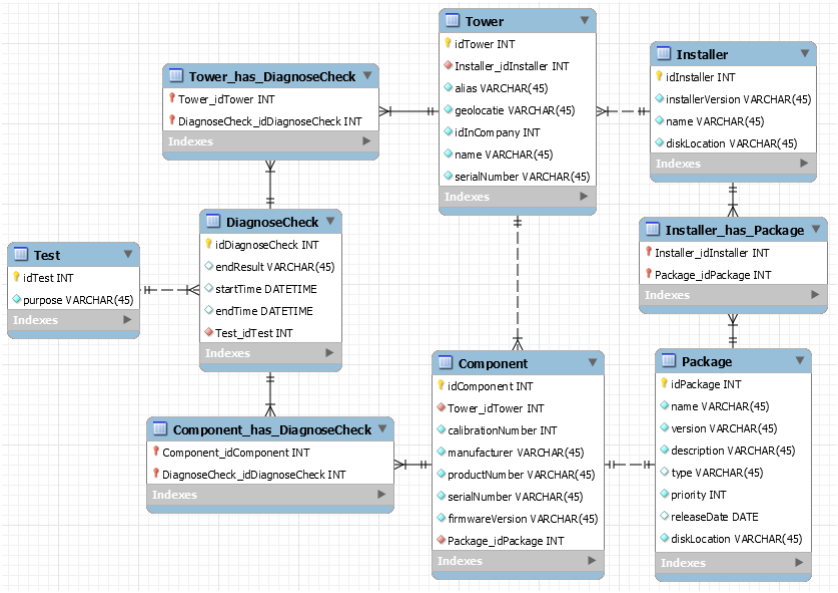
Schrijven over de test setup voor de verschillende technologieën die gebruikt is tijdens de stage?

Schrijven over waarom Qt effectief is gekozen

2.1.2 Databank

Het aantal pakketten die gebruikt gaan worden, gaan alleen toenemen. Naast deze groei zullen ook het aantal gebruikers toenemen. Daarom werd er geopteerd om een databank te ontwerpen voor het opslaan van alle cruciale data. In overleg met het bedrijf werd ervoor gekozen om MySQL te gebruiken als managementsysteem. Het ontwerp van de databank is terug te vinden in Figuur 2.1 op de volgende pagina.

Tijdens het ontwerp van de databank, in overleg met het bedrijf, zijn er verschillende keuzes gemaakt die moeten toegelicht worden. Dit ontwerp weerspiegelt de structuur die aangehaald werd in Figuur 1.1 op pagina 2 met als verschil dat in het ontwerp de client centraal staat en niet de server. Verder zullen de verschillende tabellen overlopen worden en enkele termen vastgelegd worden zodanig dat deze een eenduidige betekenis gedurende de rest van de thesis. Centraal staan alle tabellen horende bij de een testtoren. Iedere toren heeft een ID, naam en serienummer. De combinatie van deze drie waarden zijn uniek binnen het bedrijf. Aangezien deze combinatie niet betekenisvol is, wordt er een alias gekoppeld aan iedere toren. Elke toren bevat 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 daar een serienummer. Vanuit het bedrijf wordt er nummer toegekend aan iedere component die gebruikt wordt om de calibratie te achterhalen. Naast het calibratienummer heeft een component een firmware versie.



Figuur 2.1: Ontwerp van de databank

Rechts van de client bevinden zich alle tabellen die horen bij de packager. De packager levert verschillende installers af die bestaan uit packages. Iedere package heeft een naam en versienummer die uniek is. Bij iedere package wordt bijgehouden welk type package het is, de prioriteit voor de installatievolgorde, een korte beschrijving en de release datum. Een package wordt gekoppeld aan een installer maar kan gebruikt worden in verschillende installers. Uiteraard draait op iedere toren een bepaalde installer en wordt één installer gebruikt op verschillende torens.

Aan de linkerzijde bevinden zich de tabellen die horen bij het logger gedeelte. Tijdens het installeren van het framework kunnen verschillende tests uitgevoerd worden om te testen of een component correct is geïnstalleerd. Op basis van de gegevens van een component kan, bij het falen van de test, gecontroleerd worden wat bijvoorbeeld de firmware versie van de component is. Ook na de volledige installatie van het framework kan een test uitgevoerd worden zodat gecontroleerd wordt of de connecties tussen de componenten goed werkt.

2.2 Deployment server

Na het packagen van de software, moet deze bij de doelsystemen geraken. Dearie (2007) spreekt kort over de release fase in de deployment levenscyclus. Deze stap is een interface tussen de ontwikkelaar en de rest van de actoren. Om een installatie uit te voeren, moet de software bij de gebruiker geraken en correct geconfigureerd zijn.

2.2.1 Deployment strategieën

Tijdens het ontwerpen van een deployment server moet er rekening gehouden worden met verscheidene beperkingen. Patterson (2008) haalt verschillende argumenten aan voor het distribueren van deze service. Enkele punten waar rekening mee moet gehouden worden alvorens een ontwerp beslissing genomen wordt, zijn: betrouwbaarheid, bandbreedte en lage wachttijden. Zoals Patterson (2008) aanhaalt, is het belangrijk dat alle gebruikers ten alle tijden de service kunnen ge-

bruiken. Aangezien het bedrijf verspreid zit over zowel Europa als de Verenigde Staten, speelt de afstand een rol in het ontwerp. Hiernaast moet er rekening gehouden worden met de deployment strategie. Münch, Armbrust, Kowalczyk en Sotó (2012) spreekt over verschillende strategieën om software te deployen:

- *Big-bang*: ieder systeem in het bedrijf 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 voortgezet wordt dan zullen verschillende versies van de software continue aanwezig zijn in het bedrijf.

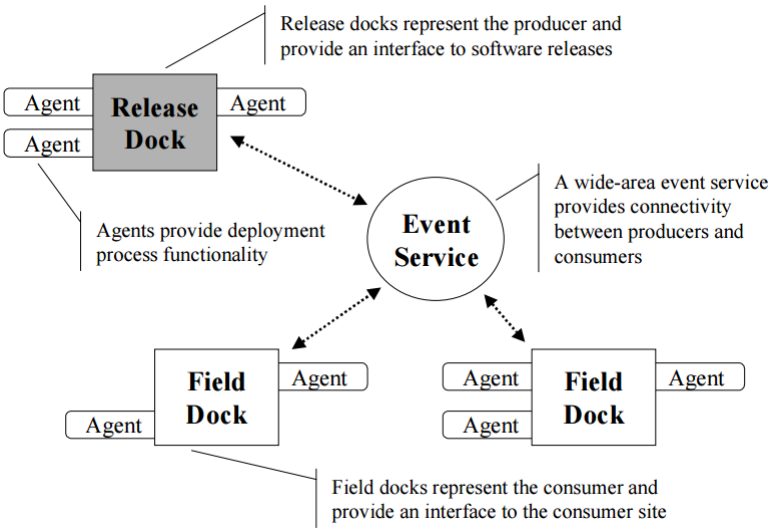
Beide strategieën zijn niet geschikt als oplossing. Door het verschil in tijdzones is het niet mogelijk om een big-bang strategie uit te voeren. Tijdens het gebruik van een testtoren is het niet mogelijk om een update uit te voeren. Het gebruik van een gefaseerde strategie kan dit probleem omzeilen. Hiernaast is de gefaseerde strategie niet ideaal aangezien het doel is dat alle doelsystemen overschakelen naar de nieuwe versie van het framework. Een hybride oplossing is hier dus aangewezen.

2.2.2 Software Dock Architectuur

Hall, Heimbigner en Wolf (1999) bespreekt een interessante architectuur. Het Software Dock research project creëerde een framework om de samenwerking tussen software producenten en verbruikers te verbeteren. In Figuur 2.2 op de volgende pagina wordt de ontworpen architectuur voorgesteld. Er worden twee verschillende componenten gedefinieerd waarmee de producenten en verbruikers voorgesteld worden. In de architectuur worden de verschillende producenten voorgesteld aan de hand van een release dock en worden de verbruikers voorgesteld als een field dock. Aan deze docks worden verschillende agents gekoppeld. Elke agent hoort typisch bij één stap uit de software levenscyclus die besproken wordt in Sectie 2.3.1 op pagina 13. Naast de verschillende docks wordt ook een wide-area event systeem gedefinieerd. Met dit systeem wordt de communicatie tussen de docks aangeboden. Hall, Heimbigner, Van Der Hoek en Wolf (1997) legt alle stappen van de Software Dock architectuur uit.

De release dock is een server die zich bevindt bij de software producent. Dit dock biedt een release repository aan waar de verbruikers de nodige software selecteren voor deployment. In de release dock wordt ieder release semantisch beschreven. 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 (Hall e.a., 1999).

De field dock dient als een interface naar de verbruiker kant toe. Deze interface biedt informatie over de resources en configuratie van het verbruiker 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 verbruikers systeem ondervragen. Aangezien kritische client-side informatie op een gestandaardiseerde wijze aangeboden wordt, met behulp van een geneste collectie van pair-values



Figuur 2.2: Software Dock Architectuur (Hall, Heimbigner & Wolf, 1999)

die een hiërarchie vormen, kan de installatie van de software gepersonaliseerd worden (Hall e.a., 1999).

2.2.3 Case studies

Door de jaren heen zijn er verschillende technologieën ontwikkeld die het probleem van software deployment aanpakken. In wat volgt, worden enkele van deze technologieën besproken gebaseerd op enkele case studies die Dearie (2007) aanreikt.

Java Beans

Enterprise JavaBeans (EJB) zijn een standaard voor het bouwen van server-side componenten. De EJB's zijn speciaal ontworpen voor het vereenvoudigen van de deployment. Dearie (2007) beschrijft JavaBeans als eenheden van business logic in een component die uitgevoerd wordt in een container. De verschillende containers zorgen voor een abstractie van de hosting omgeving en bieden verscheidene services aan. Een JavaBean moet ingepakt worden volgens de specificaties die Sun Microsystems oplegt. Met deze standaard is het mogelijk om verschillende management en deployment tools te schrijven die de EJB's kunnen beïnvloeden. Enterprise JavaBeans worden ingepakt in de standaard Java JAR file, samen met een XML deployement descriptor. De descriptor beschrijft de verschillende eigenschappen van de bijhorende EJB.

De Enterprise JavaBeans zijn volgens Dearie (2007) fijnkorrelige en taalafhankelijke oplossing voor het deployment probleem. Door het isoleren van de Beans door middel van een gestandaardiseerde container interface zullen Enterprise JavaBeans zo een oplossing vinden voor het deployment probleem. Hierdoor moeten de EJB's zodanig ontworpen worden dat ze voldoen aan de eisen van de interface. Enterprise JavaBeans hebben geen idee van het op afstand installeren van componenten. Een groot probleem bij EJB's is dat referenties naar afhankelijke beans gebeurd via niet unieke namen. Twee verschillende beans met eenzelfde naam moeten hierdoor manueel herladen worden zodanig dat de bindings up-to-date zijn (Rutherford, Anderson, Carzaniga, Heimbigner & Wolf, 2002).

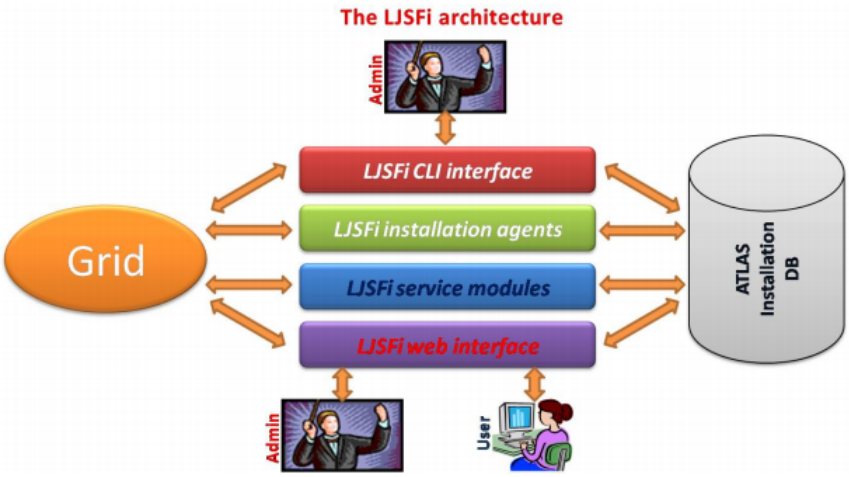
Redhat package manager

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 (Bailey, 1997). 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 (Dearie, 2007).

De Redhat Package Manager is in tegenstelling met de EJB's 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 (Dearie, 2007).

ATLAS

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 (Bird e.a., 2005). Salvo e.a. (2008) 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 (Salvo e.a., 2008)

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 modules en extensies waarmee installatie aanvragen afgehandeld worden. Het systeem bevat drie verschillende componenten:

- **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 software release aangeduid staan als productie of auto-installatie. De module verwijdert of installeert de software op alle sites waar de software nog niet gepubliceerd 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.

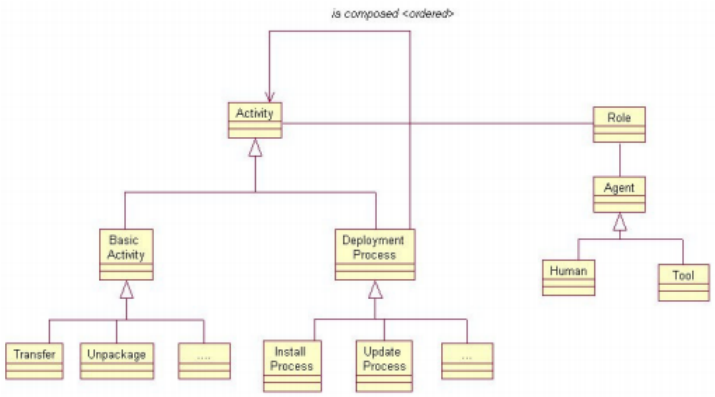
ORYA

Lestideau en Belkhatir (2003) legt uit hoe met 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 (Belkhatir, Estublier & Melo, 2007) 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.

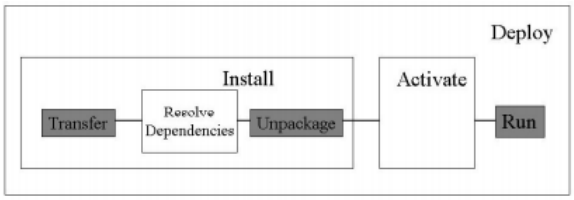
- **Applicatie Server** De applicatie server bevat nodige informatie nodig voor de installatie. Hieronder bevindt zich onder andere een pakket met de nodige resources en een manifest waarin de afhankelijkheden, beperkingen en features staan.
- **Target** De target is het doel waarop de deployment uitgevoerd 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 (2003) beschrijft verder de verschillende modellen die gehanteerd worden om een succesvolle deployment uit te voeren. In Figuur 2.4 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 (Lestideau & Belkhatir, 2003)

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 (Lestideau & Belkhatir, 2003)

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

Nix

Dolstra (2006) bespreekt hoe Nix systemen omgaan met het deployment probleem. Nix houdt verschillende componenten bij in de component store waarbij iedere component een set van bestanden is. De componenten worden van elkaar gescheiden door een unieke naam. Dit wordt bekomen door een cryptografische hash op te nemen in de naam. Nix biedt geen software deployment aan. Het biedt verscheidene mechanismen aan waarmee verschillende deployment beleid beschikbaar worden. Met de volgende deployment models is het mogelijk om in Nix de Nix expressies (de bouwstenen van de Nix componenten) te verspreiden:

- **Handmatige download** Een gebruiker kan zelf pakketten downloaden in de vorm van tar archieven, deze zelf uitpakken en vervolgens installeren. Deze strategie is arbeidsintensief en maakt het moeilijk om alles up-to-date te houden.

- **Updaten aan de hand van een versie management systeem** Een andere strategie is het gebruik van een versie management systeem. Hiermee is het up-to-date houden van de pakketten zeer eenvoudig.
- **Kanalen** Een verdere uitbreiding zijn de kanalen. Een kanaal is een URL naar een tar archief die de nodige Nix expressies bevat. Met deze strategie is het even eenvoudig om pakketten te installeren en up-to-date te houden.
- **One-click installatie** Als er enkel één pakket geïnstalleerd moet worden, dan is de one-click installatie de eenvoudigste optie. Via de website van de verdeler kan een link gebruikt worden om het nodige pakket te installeren.

2.3 Deployment environment

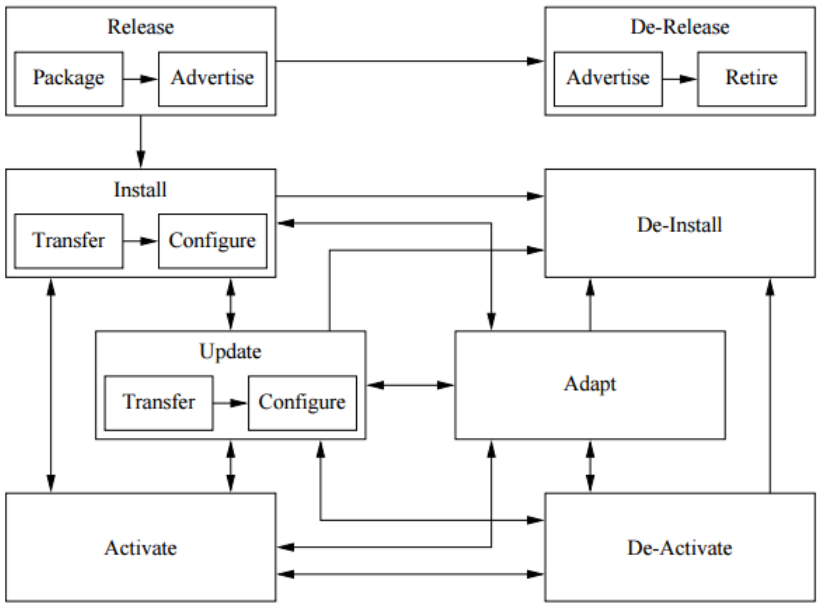
2.3.1 Software levenscyclus

De levenscyclus van software deployment kan volgens Dearie (2007), Hall e.a. (1999) 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 tegengesteld aan activeren. Deze stap is voor nodig voordat een aanpassing of herconfiguratie kan uitgevoerd 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 client systeem 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 client systemen.

In theorie zou het deployen van software een eenvoudig probleem moeten zijn. Aangezien software bestaat uit een set van bestanden, dan zou het deployen van software naar een doelcomputer slechts bestaat uit het kopiëren van de nodige bestanden. Maar dit is vaak niet het geval. Volgens Dolstra (2006) zijn er in de praktijk verschillende oorzaken die ervoor zorgen dat het deployment probleem ingewikkelder wordt. Deze oorzaken kunnen in twee grote categorieën ingedeeld worden, namelijk de omgevings- en de onderhoudsproblemen.



Figuur 2.6: Levenscyclus van software (Carzaniga e.a., 1998)

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 configuratie bestanden, ... Als deze eisen niet voldaan zijn, dan zal de software niet werken zoals gewenst. Dolstra (2006) haalt enkele concrete voorbeelden aan van omgevingsproblemen:

- De deployment van software kan een gedistribueerd probleem zijn. Software kan afhankelijk zijn van componenten draaiende op verwijderde systemen of van andere processen draaiende op het doelsysteem.
- Software is vaak afhankelijk van verschillende andere software componenten. Deze afhankelijkheden, of ook wel dependencies genoemd, moeten voor de deployment bepaald worden. Dit proces is moeilijk en een fout kan pas laat ontdekt worden.
- De dependencies moeten compatibel zijn met wat er verwacht wordt van de software. Niet elke versie zal werken. Sommige depencies vertonen build-time variaties. De component kan dan gebouwd zijn met enkele optionele eigenschappen of eigenschappen die gekozen worden at build-time.
- 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 het snel duidelijk dat er twee problemen zijn: de verschillende eisen van de software moeten geïdentificeerd worden en vervolgens moeten deze gerealiseerd worden in het doelsysteem.

Onderhoudsproblemen Naast de verschillende omgevingsproblemen, beschrijft Dolstra (2006) ook enkele onderhoudsproblemen. Software moet kunnen "evolueren". Om dit te ondersteunen,

moeten allerlei actie zoals upgraden en updaten uitvoerbaar zijn. Enkele voorbeelden van zulke acties zijn:

- Tijdens het verwijderen van software, moeten alle componenten verwijderd worden. Ondertussen mogen 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 roll back uit te voeren. Zo'n actie kan overwogen worden als de upgrade belangrijke functionaliteiten van de software kapot maakt.

2.3.2 Rollback

Het deployen van software op een doelsysteem brengt verscheidene problemen met zich mee. Bij het voorkomen van een probleem moet een mechanisme aanwezig zijn zodanig dat de problemen opgevangen worden.

Rollback strategieën

Stuk schrijven over rollback strategieën? Kan dan eenvoud vm aanhalen? Srinivasan, Kandula, Andrews, Zhou e.a. (2004) bespreekt 3 mogelijke manieren van rollback Elnozahy, Alvisi, Wang en Johnson (2002) $->$ is vooral voor gedistribueerde systemen (is dit wel van toepassing hier?) Machado e.a. (2008) $->$ rollback maar dan atomisch

Virtualisatie

Dearie (2007) brengt virtualisatie ter sprake. 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. 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 (Shumate, 2004).

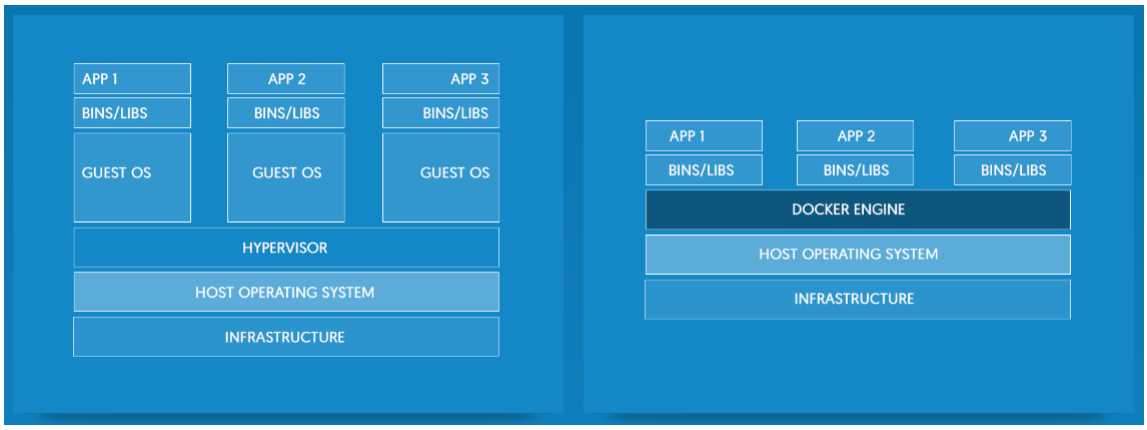
De structuur die Shumate (2004) voorstelt voor het deployen van VM images lijkt handig

Maar virtualisatie is niet de enige techniek die gebruikt zou kunnen worden om roll backs 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 („Docker Main Page", 2016). Docker containers is niet hetzelfde als virtuele machines. Docker (2016) bespreekt de twee aan de hand van een vergelijking. In de vergelijking 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.



Figuur 2.7: Architectuur van Virtuele Machine ten opzichte van Docker („Docker Main Page”, 2016)

In Figuur 2.7 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 (2014) 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 (Merkel, 2014). 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 (2014) 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.

Hoofdstuk 3

Prototype

In Hoofdstuk 2 op pagina 3 werden verschillende oplossing aangehaald die een oplossing bieden op de problemen die in Hoofdstuk 1 op pagina 1 voorgelegd werden. In het volgende hoofdstuk worden de verschillende keuze toegelicht en vervolgens de structuur en architectuur van het prototype voorgelegd.

3.1 Uitwerking

3.1.1 Architectuur

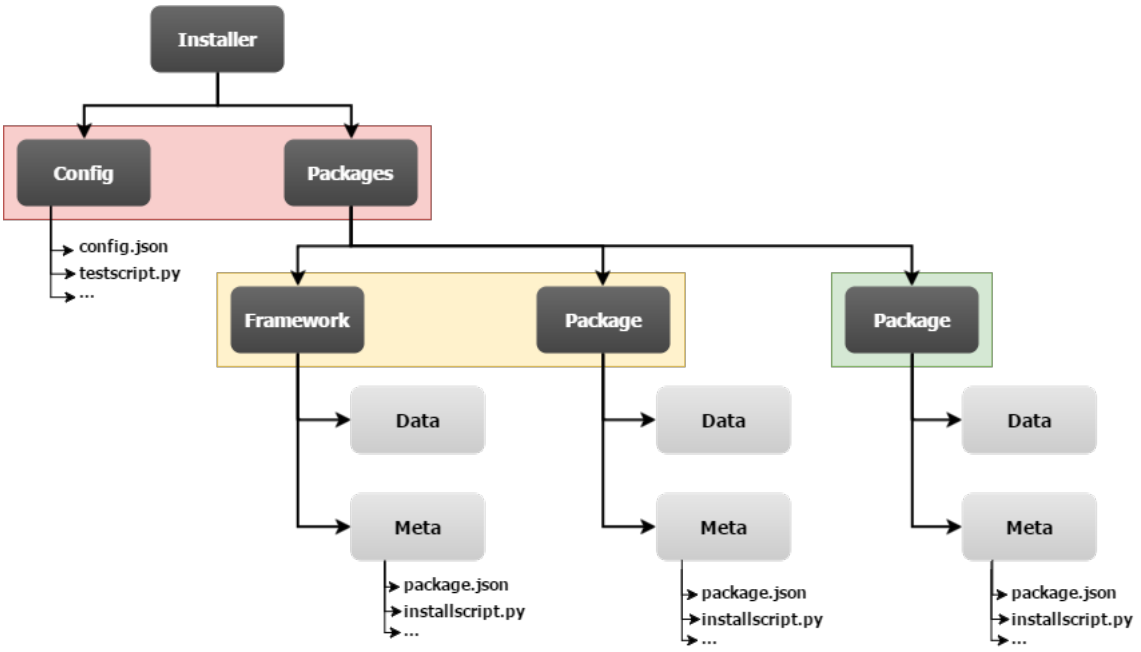
De gekozen architectuur berust op verschillende concepten die gezien zijn in Hoofdstuk 2 op pagina 3. Wederom wordt het project opgedeeld in de drie grote componenten: de packager, deployment server en de deployment environment. Elke component wordt apart belicht, gevolgd door een globale bespreking.

Packager

De architectuur van de packager wordt gebaseerd op de architectuur en structuur van het Qt installer framework. Hierbij wordt bedoelt dat één overkoepelende installer wordt geproduceerd die bestaat uit verschillende kleine componenten. Het Qt installer framework wordt hiervoor niet zelf gebruikt. Deze keuze werd gemaakt op basis van verschillende argumenten. Door het personaliseren van de packager, is het mogelijk om iedere stap in het deployment proces te personaliseren. Op deze manier kan na het installeren van een pakket een verbeterde afhandeling plaats vinden. Een test kan uitgevoerd worden op het einde van de installatie, een handeling die met het Qt installer framework mogelijk is maar moeilijk te realiseren is. Doordat een gepersonaliseerde packager wordt ontworpen, worden problemen met Docker vermeden. In de deployment environment wordt Docker gebruikt om verscheidene problemen met het deployment proces op te vangen (dit wordt verder besproken). De Docker omgeving, zoals reeds vermeld in Sectie 2.3.2 op pagina 15, gebruikt van LXC. Het besturingssysteem van de containers aan de client side is hierdoor Linux. Windows gebruiken als besturingssysteem is mogelijk maar deze optie staat nog altijd in beta schoenen. De installer geproduceerd door het Qt installer framework moet dus compatibel zijn met Linux. Dit is mogelijk met het framework maar de productie van de installer moet ook plaatsvinden in een Linux besturingssysteem. Om dit te realiseren zou Docker gebruikt kunnen worden zodanig

dat er een abstractie gedaan wordt van het host besturingssysteem. Zo'n opstelling creëren, waarbij een container gebruikt wordt met het Qt installer framework in verwerkt, brengt meer werkt en is omslachtiger ten opzichte van het zelf fabriceren van een packager met een gelijkaardige structuur en gelijkaardige functionaliteiten.

De packager gaat een gelijkaardige structuur hebben als het Qt installer framework. Om de packager te maken wordt Python gebruikt als programmeertaal. Een installer bestaat uit verschillende pakketten en functioneert als overkoepelend geheel. Per test framework wordt één installer geassocieerd. Voor alle drivers/bibliotheken die nodig zijn, worden verschillende pakketten voorzien waarbij één driver hoort bij één pakket. Ieder pakket bestaat uit twee delen: een data en metadata gedeelte. Het data gedeelte bevat de effectieve driver/bibliotheek en het metadata gedeelte bevat de nodige beschrijving van het pakket in de vorm van een "package.json". Hiernaast zijn verschillende scripts aanwezig die gebruikt worden voor een gepersonaliseerde installatie. De structuur van een mogelijke installer is terug te vinden in Figuur 3.1. In het voorbeeld wordt een installer opgebouwd bestaande uit twee delen. Het eerste deel bestaat uitsluitend uit configuratie bestanden en het tweede deel bestaat uit de pakketten. De twee onderdelen zijn terug te vinden in de rode rechthoek. Verder worden de packages ook opgedeeld in twee categorieën. De eerste categorie is cruciaal voor de correcte werking van het test framework. Dit onderdeel bestaat uit een versie van het test framework gecombineerd met de pakketten die nodig zijn voor een fatsoenlijke werking (deze pakketten worden aangegeven door de gele rechthoek). Hiernaast is er één pakket aanwezig die niet cruciaal is voor het testraamwerk maar wel nodig is om een correcte werking te verzekeren op de site zelf (aangegeven door de groene rechthoek). Deze bevindt zich in de tweede categorie.



Figuur 3.1: Structuur van een installer bestaande uit drie pakketten

Om een dergelijke structuur op te bouwen, werkt de packager nauw samen met de databank waarin alle informatie vervat zit. Het ontwerp van de databank werd al uitvoerig besproken in Sectie 2.1.2 op pagina 6 en is terug te vinden in Figuur 2.1 op pagina 7.

Deployment server

Deployment environment

De deployment omgeving bestaat uit een

3.1.2 Flowcharts

3.2 Testen

Hoofdstuk 4

Conclusie

Referenties

Bailey, E. C. (1997). Maximum rpm.

Belkhatir, N., Estublier, J. & Melo, W. (2007). THE ADELE-TEMPO experience: an environment to support process modeling and enaction. *Software Process Modelling and Technology Research Studies Press*, 1–37.

Bird, I., Bos, K., Brook, N., Duellmann, D., Eck, C., Fisk, I., ... Grandi, C. e.a. (2005). LHC computing Grid. *Technical design report*, 8.

Carzaniga, A., Fuggetta, A., Hall, R. S., Heimbigner, D., Van Der Hoek, A. & Wolf, A. L. (1998). *A characterization framework for software deployment technologies*. DTIC Document.

Chamberlain, R. & Schommer, J. (2014). Using Docker to support reproducible research. *DOI: http://dx. doi. org/10.6084/m9. figshare*, *1101910*.

Chocolatey About page. (2016). https://chocolatey.org/about. [Online; geraadpleegd 3-08-2016].

Chocolatey Main page. (2016). https://chocolatey.org/. [Online; geraadpleegd 3-08-2016].

Dearie, A. (2007). Software Deployment, Past, Present and Future. In *Future of Software Engineering, 2007. FOSE '07* (pp. 269–284). doi:10.1109/FOSE.2007.20

Docker. (2016). *Docker for the Virtualization Admin*.

Docker Main Page. (2016). https://www.docker.com/. [Online; geraadpleegd 16-08-2016].

Dolstra, E. (2006). *The purely functional software deployment model*. Utrecht University.

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

Hall, R. S., Heimbigner, D. & Wolf, A. L. (1999). A cooperative approach to support software deployment using the software dock. In *Software Engineering, 1999. Proceedings of the 1999 International Conference on* (pp. 174–183). IEEE.

Lestideau, V. & Belkhatir, N. (2003). Providing highly automated and generic means for software deployment process. In *European Workshop on Software Process Technology* (pp. 128–142). Springer.

Liefke, H. & Suciu, D. (2000). XMill: An Efficient Compressor for XML Data. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data* (pp. 153–164). SIGMOD '00. Dallas, Texas, USA: ACM. doi:10.1145/342009.335405

Merkel, D. (2014). Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*, *2014*(239), 2.

NSIS Main page. (2016). http://nsis.sourceforge.net/Main_Page. [Online; geraadpleegd 3-08-2016].

Obreshkov, E., Albrand, S., Collot, J., Fulachier, J., Lambert, F., Adam-Bourdarios, C., ... Goldfarb, S. (2008). Organization and management of {ATLAS} offline software releases. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, De-*

tectors and Associated Equipment, *584*(1), 244–251. doi:http://dx.doi.org/10.1016/j.nima.2007.10.002

Patterson, D. A. (2008). The data center is the computer. *Communications of the ACM*, *51*(1), 105–105.

QT Installer Framework Documentation. (2016). http://doc.qt.io/qtinstallerframework/. [Online; geraadpleegd 4-08-2016].

Rutherford, M. J., Anderson, K., Carzaniga, A., Heimbigner, D. & Wolf, A. L. (2002). Reconfiguration in the Enterprise JavaBean component model. In *International Working Conference on Component Deployment* (pp. 67–81). Springer.

Rybkin, G. (2012). ATLAS software packaging. *Journal of Physics: Conference Series*, *396*(5), 4.

Salvo, A., Barchiesi, A., Gnanvo, K., Gwilliam, C., Kennedy, J., Krobath, G., ... Rybkine, G. (2008). The ATLAS software installation system for LCG/EGEE. In *Journal of Physics: Conference Series* (Deel 119, *5*, p. 052013). IOP Publishing.

Shumate, S. (2004). Implications of Virtualization for Image Deployment. Dell Power Solutions. Verkregen van http://www.dell.com/downloads/global/power/ps4q04-20040152-Shumate.pdf

Tian, H., Zhao, X., Gao, Z., Lv, T. & Dong, X. (2010). A Novel Software Deployment Method Based on Installation Packages. In *2010 Fifth Annual ChinaGrid Conference* (pp. 228–233). doi:10.1109/ChinaGrid.2010.32

WiX Toolset. (2016). http://wixtoolset.org/. [Online; geraadpleegd 3-08-2016].

WiX Toolset Tutorial. (2016). https://www.firegiant.com/wix/tutorial/. [Online; geraadpleegd 3-08-2016].

Bibliografie

Elnozahy, E. N., Alvisi, L., Wang, Y.-M. & Johnson, D. B. (2002). A survey of rollback-recovery protocols in message-passing systems. *ACM Computing Surveys (CSUR)*, *34*(3), 375–408.

Machado, G. S., Daitx, F. F., da Costa Cordeiro, W. L., Both, C. B., Gaspary, L. P., Granville, L. Z., ... Saikoski, K. (2008). Enabling rollback support in IT change management systems. In *Network Operations and Management Symposium, 2008. NOMS 2008. IEEE* (pp. 347–354). IEEE.

Münch, J., Armbrust, O., Kowalczyk, M. & Sotó, M. (2012). *Software process definition and management*. Springer Science & Business Media.

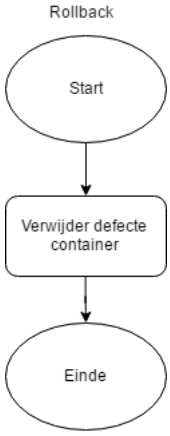
Pitts, D., Ball, B. e.a. (1998). *Red Hat Linux*. Sams.

Pressman, R. S. (2005). *Software engineering: a practitioner's approach*. Palgrave Macmillan.

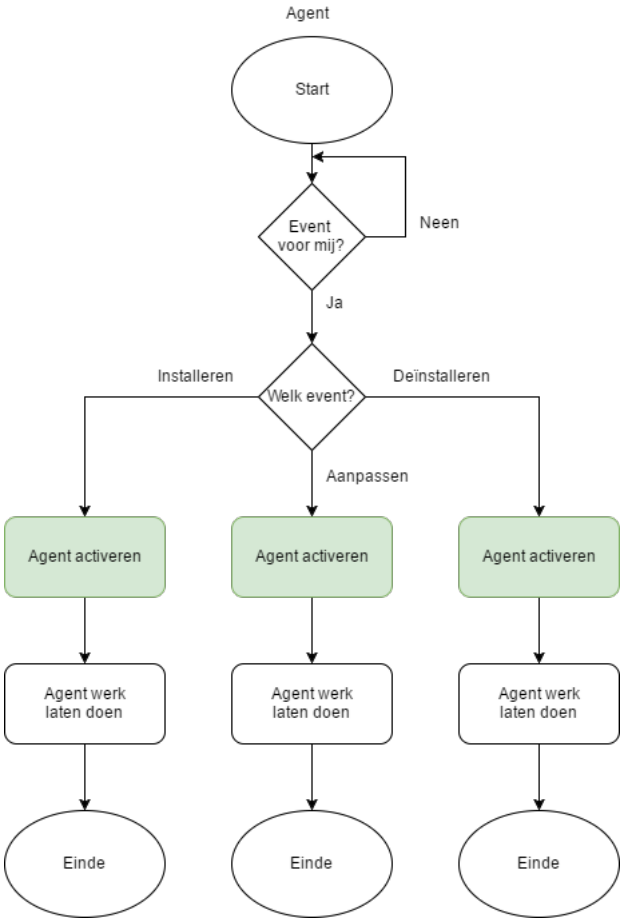
Srinivasan, S. M., Kandula, S., Andrews, C. R., Zhou, Y. e.a. (2004). Flashback: A Lightweight Extension for Rollback and Deterministic Replay for Software Debugging. In *USENIX Annual Technical Conference, General Track* (pp. 29–44). Boston, MA, USA.

Bijlage A

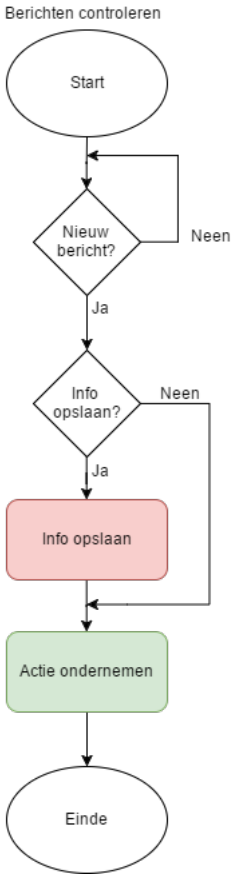
Flowcharts



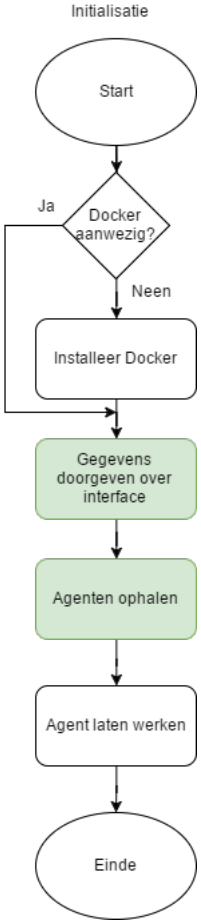
Figuur A.1: Rollback



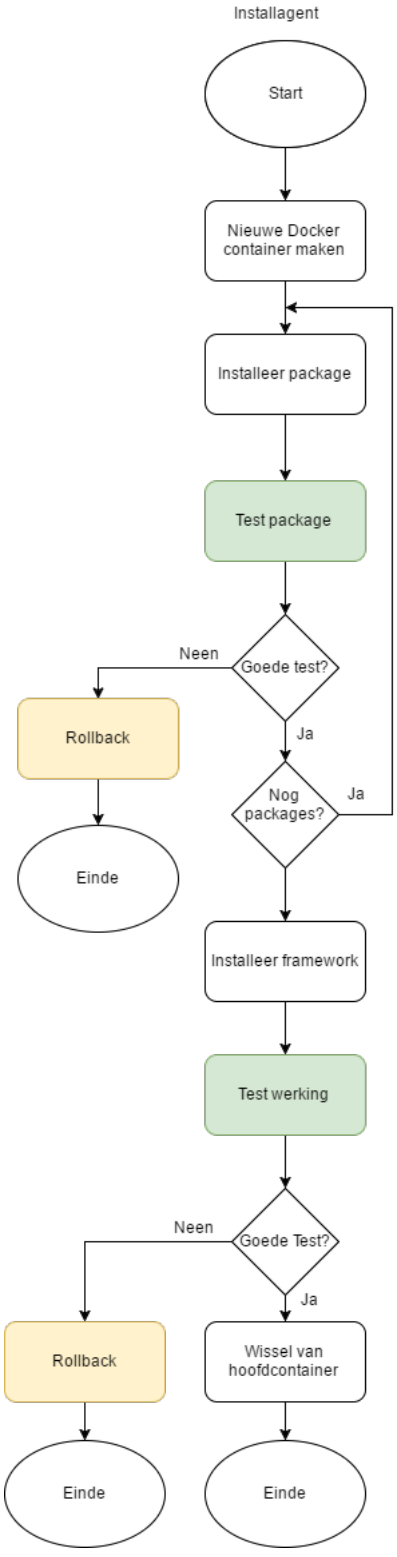
Figuur A.2: Flowchart van een agent



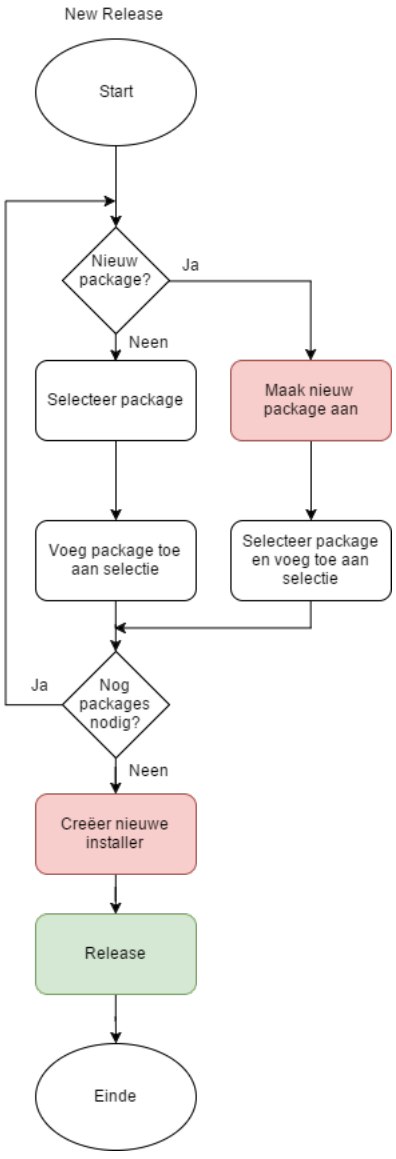
Figuur A.3: Flowchart voor het ophalen/controleren van berichten



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



Figuur A.5: Acties van de installagent



Figuur A.6: Flowchart van het creëren van een nieuwe release

Bijlage B

Beschrijving van deze masterproef in de vorm van een wetenschappelijk artikel

Bijlage C

Poster

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