Steven Schenk 0894490 & Cees-Jan Nolen 0902130 | INFDEV016B | 14 januari 2016

Opdracht C

Software delivery

## http://www.neosllc.com/wp-content/uploads/Software-Delivery.jpg

Stakeholders

Dit document is gemaakt om een verbinding te vormen tussen de Product owner, de stakeholders en het ontwikkel team. In dit document zijn alle requirements opgenomen die wij toe zullen passen in ons systeem, zodat er geen verwarring ontstaat.

Bij het project zullen de volgende stakeholders betrokken zijn:

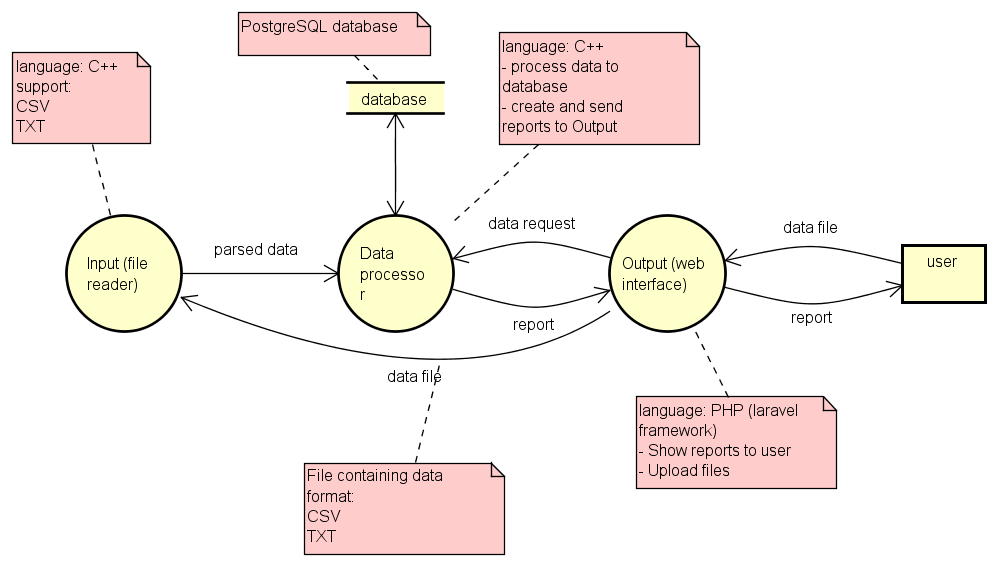
* The project leader
* The potential users of the system
* CityGis

Systeem ontwerp

Het systeem zal verdeeld worden in 3 processen:

1. Input-system.
2. Data-processing system.
3. Output-providing system.

Al deze processen samen vormen het hele systeem. De afbeelding hieronder geeft een mooie grafische weergave hoe ons systeem eruit zal komen te zien. Het geeft de grensen aan tussen de sub-systemen en geeft een weergave van de dataoverdracht.



**Figuur 1: Context diagram and system boundary**

Systemenvironment

Het systeem zal gehost worden op de servers van Hogeschool Rotterdam dat onderhouden wordt door het development team. De server maakt gebruik van de volgende hard- en software:

Hardware:

* CPU: Intel Xeon e5-2650 @2.00 Ghz
* Memory: 3953 Megabytes
* Disk: 15 Gigabytes
* OS: Ubuntu 14.04.3

Software:

* C++ programming language
* PHP inc. Laravel framework
* PostgreSQL
* Git inc. Github
* Homestead\*
* Vagrant\*
* Composer\*

\*: These three software-systems are used to accommodate development in the Laravel framework.

Plan deployment pipeline

Technologie

Voor het deployen van onze software maken we gebruik van de tool Jenkins. Hiermee kunnen we onze software automatisch testen, compilen en deployen. Jenkins geeft ons rapportages terug over onze code. Hiermee kunnen wij kijken of onze code bug vrij is en ook van goede kwaliteit is. Omdat dit elke uur wordt uitgevoerd krijgen wij een trend lijn waarmee we kunnen reflecteren of we voorruit gaan of juist achteruit. Ook kunnen we hiermee de stabiliteit van ons project meten en hierdoor garanderen.

Server configureren

Voordat we onze code kunnen deployen op de server, moet de server eerst alle benodigde software draaien waar ons project van afhankelijk is. Omdat het veel tijd kost om dit elke keer met de hand te doen hebben we hiervoor een script gemaakt. Dit script hoef je eenmalig te runnen waarna de server volledig geconfigureerd is voor ons project

Dependencies:

* Git
* LibPqxx (Postgres driver C++)
* UnitTest++ (Unit tests voor C++)
* CMake
* G++ (C++ compiler)
* CppCheck (Style checking C++)
* PostgreSQL (database)

Stappenplan Jenkins

Wij laten Jenkins de volgende bouwstappen uitvoeren:

Build

1. Ga naar de directory (home/USER/projects)
2. Git clone de nieuwste versie van de master branch (cooperatio/Project56)
3. Ga naar de juiste work directory (/home/USER/projects/Project56)
4. Compile de source code, zowel main processen als Unit tests (g++ met flags: -std=c++11 –pthread –lpqxx –lpq en voor UnitTest ook nog –lUnitTest++)
5. Alles moet nu up to date en gecompiled zijn

Test

1. Ga naar de juiste directory (/home/USER/projects/Project56)
2. Execute de gecompileerde Unit test
3. Voer de CppCheck uit over de c++ code d.m.v. command (cppcheck)

Release

1. Nginx instellen voor deployment van website (nginx.conf file copy naar nginx directory en daar nginx.conf overschrijven)
2. PostgreSQL instellen, juiste users aanmaken die ons systeem gebruikt
3. Database aanmaken in PostgreSQL
4. Tabellen aanmaken in PostgreSQL door midden van php artisan migrate (in de laravel directory)

Deployment

1. Kopieer de website files van de geclonde git repository naar root directory van Nginx (/var/www/laravel)
2. Run composer update op de website root directory
3. Kopieer de gecompileerde bestanden naar de juiste plek
4. Start het input proces
5. Start nginx
6. Start postgres

Release plan

Doel van het project

Het project heeft als doel om een systeem te maken dat snel data kan verwerken. De data wordt geleverd in csv format. Deze moet zo snel mogelijk geparsed worden en in een database gestopt worden. Daarna moet het systeem deze data kunnen aggregeren tot verslagen, zodat de gebruiker deze verslagen kan downloaden.

Indeling

Timeline (les)week 7-10

In deze fase zullen we vooral onderzoeken. We gaan ons verdiepen in de platformen en programmeertalen. Hierna zullen we gebruik maken van UML om blauwdrukken van ons systeem te maken. Dit zullen we vooral doen met classen diagrammen. Er zal in deze periode ook al kleine stukjes code geprogrammeerd worden. Dit zal vooral in het output proces zijn. De design van de site zal helemaal ontworpen zijn en eventueel enkele functionaliteiten ingebouwd.

Timeline (les)week 11-14

In deze fase zullen alle processen ontwikkeld worden tot bruikbare componenten. Ieder proces moet geïsoleerd kunnen werken en moet op meerde servers kunnen functioneren die voldoen aan de hardware eisen.

Timeline (les)week 15-16

In deze periode zullen de onafhankelijke processen aan elkaar worden gekoppeld en het systeem zal gevormd worden tot een geheel. Daarnaast zal er ook grondig getest worden en zal de code refactored worden waar nodig is. In deze fase worden de puntjes op de i gezet. Hierna moet het product volledig deployment ready zijn.

Rolverdeling en verantwoordelijkheden

|  |  |
| --- | --- |
| Persoon | Taken/verantwoordelijkheden |
| Steven Schenk | * Processing proces * Output proces (prototype) * UML diagrammen * Aanspreekpunt voor begeleider |
| Robert Kraaijeveld | * Processing proces * UML diagrammen |
| Cees-Jan Nolen | * Input proces * UML diagrammen |
| William de Visser | * Output proces * UML diagrammen |
| Robin Bakker | * Output proces * UML diagrammen |

Environments

Voor ons project hebben we eigenlijk drie verschillende environments. Een environment voor het ontwikkelen van de software, een environment voor het testen van de software, en een environment waar we onze software op deployen.

developing environment

Om makkelijk software te ontwikkelen is het belangrijk dat we allemaal een goede ontwikkelomgeving hebben. We maken allemaal gebruik van IntelliJ en van C-lion. Omdat we in C/C++ programeren moesten we ook allemaal beschikking hebben over een C++ compiler. Voor de front end gebruiken we het laravel framework. Ook dit moet goed geconfigureerd worden alvorens je het kan gebruiken.

Omdat deze environment geen virtuele environment is en draait op ieder zijn eigen laptop/desktop, verschil deze environment bij ieder ook. Hierdoor is het niet mogelijk om allemaal hetzelfde te hebben en kan de één tegen problemen aanlopen, waar de ander weer niet tegen aan loopt.

Deployment/Testing environment

De test environment was een virtuele server waarop een Ubuntu 14 image was geïnstalleerd. Daarnaast alle benodigde software om ons project draaiend te krijgen. Eigenlijk is de test omgeving een weerspiegeling van de deployment omgeven. Daarom kunnen we voor de deployment omgeving dezelfde image gebruiken als voor de test omgeving.

Benodigde software dependencies:

* LibPqxx 4.0 (Postgres driver C++)
* UnitTest++ 1.5.0 (Unit tests voor C++)
* CMake 2.8.12.2
* G++ 4.8.4 (C++ compiler)
* CppCheck 1.61 (Style checking C++)
* PostgreSQL 9.3.10 (database)
* Git 1.9.1

Het upgraden van software

Update proces

Als je het systeem op de server wilt aanpassen door middel van een update dan is het natuurlijk belangrijk dat dit gebeurd zonder dat de user dit in de gaten heeft. Dit betekent dus dat er geen tolerantie is voor downtime. Omdat ons systeem draait op maar één enkele virtual machine wordt dit natuurlijk erg lastig.

Het mooiste zou zijn als je 2 of meer virtual machines hebt draaien die elk hetzelfde proces draaien. Als je dan een update uitbrengt kan je eerst virtual machine A uitzetten en updaten, terwijl de users gewoon nog virtual machine B gebruikt. Daarna kan je A weer aanzetten en B updaten. Vanwege gebrek aan tijd is dit voor ons niet mogelijk. Daarom zullen wij proberen om met zo weinig mogelijk downtime een update uit te voeren.

Wat bij ons wel mogelijk is om twee keer hetzelfde proces te runnen, tegelijkertijd. Waar het hier op neer komt is dat we heel makkelijk de nieuwe versie kunnen compilen, en daarna laten draaien. Op dit moment draaien dus twee dezelfde processen, weliswaar één proces met de nieuwste update, tegelijkertijd. Op het moment dat de oude versie al zijn werk gedaan heeft kan deze dus gesloten worden. Op deze manier hopen wij zonder downtime toch een update uit te kunnen brengen.

Stappen plan:

1. Haal de nieuwste source code van github
2. Compile de nieuwe source code
3. Zet de gecompileerde software op de juiste plek
4. Start de nieuw gecompileerde software
5. Sluit het oude software proces zodra deze niet meer gebruikt wordt

Het afstemmen van de update

Omdat processen van elkaar afhankelijk zijn is het altijd een beetje tricky om zomaar een nieuwe update uit te brengen. Proces A roep bijvoorbeeld proces B aan. Maar als je proces B update waardoor proces A proces B op een andere manier moet aanroepen, moet je dus ook per se proces A aanpassen. Anders werkt het systeem niet meer na de update.

Alvorens een update uitgebracht wordt moet er dus goed overlegd worden wat hier de consequenties voor zijn, en hoe je deze uit de weg gaat. Daarnaast zal een nieuwe update eerst uitvoerig getest moeten worden in een test omgeving. Pas als zeker is dat alles goed blijft functioneren na de update, zal deze uitgebracht worden. Hierbij moet je goed rekening houden dat als je Proces A update en hiervoor moest ook proces B een aanpassing krijgen, dat je alle twee deze processen tegelijk update.

Inrichten van de server

automatisch script

Het inrichten van een server zal gebeuren door middel van een automatisch script. De script maakt de nodige users aan, installeert alle benodigde dependencies, configureert alle software, cloned het project van git en compiled het waar nodig is. Zodra het script klaar is met runnen moet de server voor 100% klaar zijn en moet alles in de lucht zijn. Het script kan meerdere keren worden uitgevoerd zonder dat dit problemen oplevert.

Om dit te behalen zal ons script er als volgt uitzien:

1. Creëren van alle benodigde users (Nginx user, postgres user)
2. Installeren van de software (Nginx, PostgreSQL, PHP5, Git, Composer, Laravel, g++, libpqxx, libpq)
3. Software services toekennen aan specifieke user (postgres -> PostgreSQL, nginx -> nginx) en juist permissies toekennen
4. Project clonen van github
5. Compilen van de source code
6. Het project starten