

13-1-2017

Implementatie Ontwikkelstraat

IPSENH



RZN
REPOSITORY ZONDER NAAM

Versiebeheer

Versie	Datum	Omschrijving	Wie
0.1	28-11-2016	Opzet document, verantwoording gemaakt	Anton
0.2	28-11-2016	Invulling Implementatie	Anton
0.3	29-11-2016	Invulling afbeelding ontwikkelstraat	Sander
0.4	1-12-2016	Invulling OTAP, Issue tracking, Changes, Backlog	Anton
0.5	19-12-2016	Invulling OTAP met SCRUM, Notificatie, deployscript	Anton
0.6	22-12-2016	Aanpassing JSLint -> TSLint met reden	Anton
0.7	08-01-2017	Configuraties ontwikkelstraat	Zairon
0.8	09-01-2017	configuraties ontwikkelstraat	Zairon
0.9	13-01-2017	SonarQube hoofdstukken toegevoegd.	Sander
1.0	13-01-2017	Document afgerond voor oplevering	Sander

Inhoud

Ontwikkelstraat	3
Flow	3
Implementatie	3
OTAP bij de ontwikkelstraat	4
OTAP met SCRUM.....	5
Ontwikkelstraat om inzicht te geven.....	5
Ontwikkelstraat Systeem Architectuur	6
De flow van OTAP (DTAP)t.o.v. de server architectuur.....	6
Specificaties Acceptance / Production-server.....	6
Hulp bij het opzetten/aansluiten van de ontwikkelstraat en de begeleiders	7
Wijzigingen t.o.v. onderzoeksrapport en verantwoording	7
Ontwikkelstraat	7
Code review, issue tracking, backlog.....	7
Uitleg configuratie ontwikkelstraat.....	8
Globale Configuratie Server	8
Globale Configuratie Lokale Machine	8
Configuratie Internet Information Services.....	9
Configuratie Jenkins en tools	10
Configuratie Front-end Job.....	11
Configuratie Back-end Job.....	17
Bijlage installaties	18
NodeJS installatie	18
Jenkins Installatie	18
GIT installatie.....	18
JetBrains WebStorm installatie	18
Google Chrome installatie	19
Visual studio installatie.....	19

Ontwikkelstraat

Flow

Onderstaande afbeelding geeft een beeld van onze ontwikkelstraat voor de TravelPlanner applicatie.



Implementatie

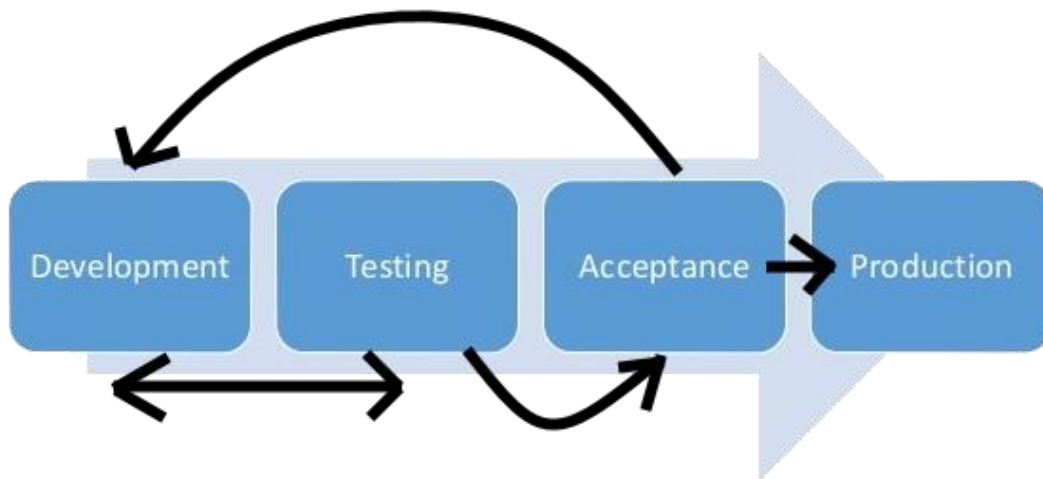
In dit hoofdstuk wordt per tool beschreven hoe deze geïnstalleerd en geconfigureerd zijn.

- Lokaal wijzigingen maken, deze test je lokaal en als alle tests slagen wordt er een Pull Request gemaakt voor de nieuwe feature. (git)
- Code Review wordt gedaan over de Pull Request
- Merge wordt uitgevoerd
- Continuous Integration ontvangt nieuwe code -> Voert tests uit en code analyse
- Continuous Integration zet build klaar
- Handmatige stap om de build op de server te deployen

Als er tijdens het testen of bouwen iets mis gaat wordt er automatisch door Jenkins een notificatie gestuurd naar een van de developers. Dit hebben wij geconfigureerd via email. Het is ook mogelijk om dat via Slack te doen maar dit hebben wij niet als communicatie platform gebruikt tijdens dit project.

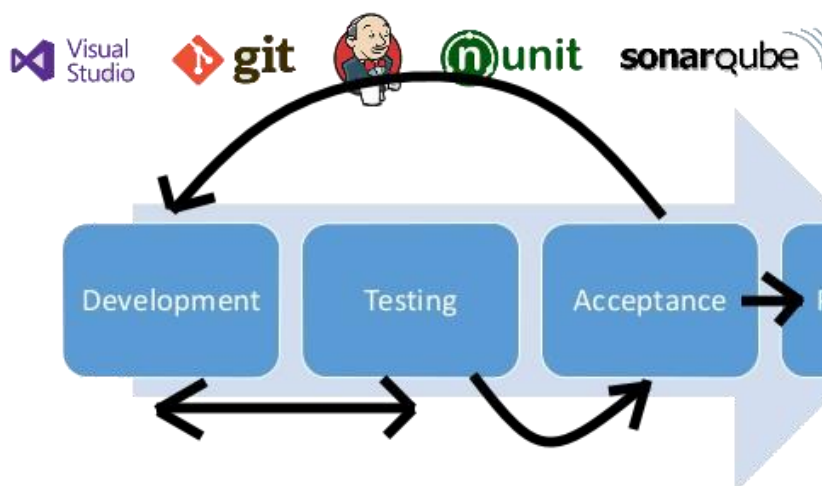
OTAP bij de ontwikkelstraat

In combinatie met de verschillende onderdelen van de ontwikkelstraat maken we gebruik van een zogenoemde OTAP opzet. OTAP staat voor Ontwikkelen Testen Acceptatie Productie (in het Engels DTAP). Hierbij wordt er voor iedere stap een aparte omgeving opgezet. Het doel hiervan is om te zorgen dat er niets dat nog niet getest is, of dat nog niet geaccepteerd is aan de wijde wereld (productie) te leveren. Hier zijn echter uitzonderingen omdat je soms feedback wilt van gebruikers. Op dat moment kan je (deels, bijvoorbeeld door het op slechts een klein gebied uit te rollen) live gaan.



Het idee is dat er vaak vanaf Development naar Testing wordt gegaan. Zodra hier resultaten uitkomen kan er dan weer terug worden gegaan naar Development, of door worden gegaan naar Acceptance. Als er bij Acceptance iets niet in orde is dan wordt er teruggekeerd naar Development. Als het allemaal geaccepteerd wordt dan kan er worden doorgedaan naar Production.

De ontwikkelstraat is functioneel vooral in het gedeelte vanaf Development -> Testing -> Acceptance



Vervolgens is het gaan naar Production een handmatige stap, we voeren hiervoor een zelf gemaakt innovatief scriptje uit dat de inhoud van een map kopieert naar de locatie die uiteindelijk live gezet wordt door bijvoorbeeld Apache.

OTAP met SCRUM

OTAP gaat goed hand-in-hand met SCRUM. Zodra er iets tussen een van de verschillende stappen van OTAP mis gaat komt er een nieuwe item op de Backlog van het SCRUM-team. Het item zal dan de eerst volgende sprint aan bod komen.

De fasen van SCRUM komen erg overeen met die van OTAP. De kolom "Doing" sluit goed aan op de O van OTAP (Ontwikkelen)

De kolom "Testing/Verification" sluit aan op de T en de A van OTAP (Testing en Acceptance)

De kolom "Done" sluit aan op de P van OTAP (Productie) want daar eindigt de code als alles goed verlopen is.

Ontwikkelstraat om inzicht te geven

De ontwikkelstraat dient niet alleen als automatisering maar ook als informatiebron voor de ontwikkelaars en de product owner. Het dashboard van Jenkins en Sonar Qube verschaft een hoop informatie over de efficiëntie en effectiviteit van de ontwikkelstraat en het ontwikkelproces. Voor de front-end hebben we hiervoor TSLint.

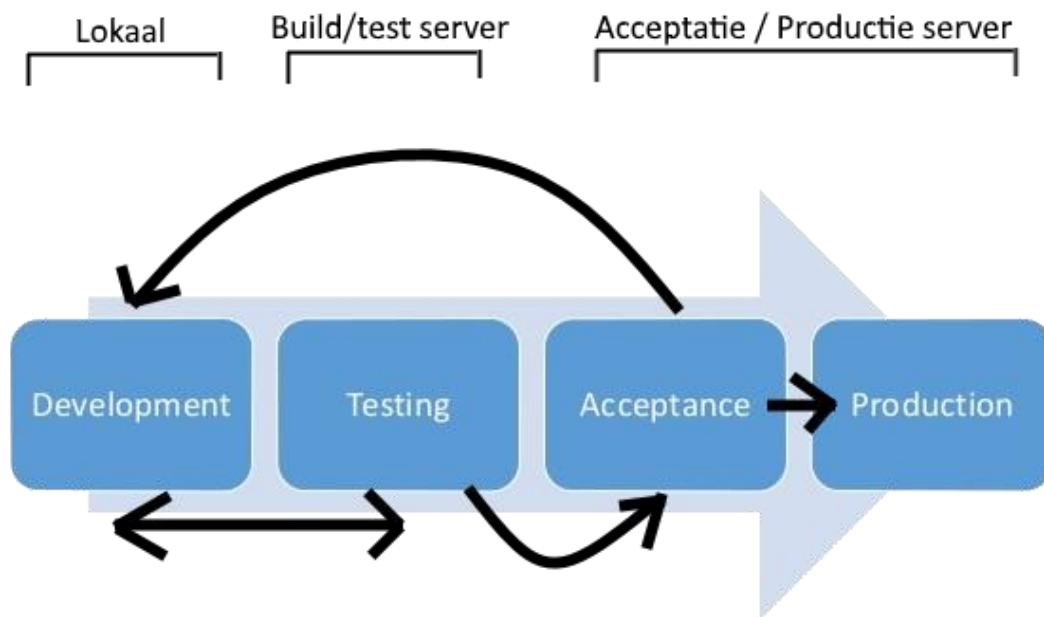
Jenkins geeft informatie over de build: Duurt het lang? Slagen de builds? Zitten er fouten in de tests? Hiermee kunnen we efficiëntie en effectiviteit meten

Sonar Qube geeft informatie over de code zelf, kan de code beter? Zitten er onnodige duplicaties in de code? Kan het efficiënter of effectiever?

TSLint geeft informatie over de effectiviteit van de code. Er wordt hierbij gekeken naar onnodige code zoals imports die niet gebruikt worden, tekens die ontbreken en conventie afwijkingen.

Ontwikkelstraat Systeem Architectuur

Om de ontwikkelstraat te ondersteunen met systemen hebben wij de volgende architectuur voor ogen.



De flow van OTAP (DTAP)t.o.v. de server architectuur

Lokaal schrijven wij nieuwe code en testen dit.

Dan maken wij een Pull Request. Als deze wordt geaccepteerd komt deze op de testing server (de Continuous Integration vraagt telkens de nieuwste versie van de code) en deze wordt dan nog een keer getest. De CI zet dan een gebouwd pakket klaar dat kan worden gezet op de acceptatie server (handmatige stap, is veiliger en meer controle). Vervolgens wordt dit door een script op de acceptatie/productie-server gezet. De acceptatie-versie van de code zal draaien op een andere poort dan de echte productie-versie (die zal op poort 80 draaien, standaard http-poort).

Specificaties Acceptance / Production-server

Gewenste specificaties voor de acceptance / productie-server zijn als volgt:

- 2 tot 4 Cores, minstens 2 Ghz
- 4GB RAM
- Minimaal 50GB Opslagruimte
- Windows Server

Hulp bij het opzetten/aansluiten van de ontwikkelstraat en de begeleiders

Om de begeleiders te helpen hebben Laurens en Zairon de begeleider Geert-Jan geholpen met het opzetten van de ontwikkelstraat. Hierbij zijn er uitgebreid vragen gesteld over onze ontwikkelstraat en de configuratie hiervan.

Wijzigingen t.o.v. onderzoeksrapport en verantwoording

Ontwikkelstraat

Er is een nieuw onderdeel toegevoegd aan de ontwikkelstraat. Het onderdeel dient niet om de codekwaliteit te verhogen, maar om gevonden bugs en andere fouten bij te kunnen houden. Dit onderdeel heet Issue Tracking. Voor issue tracking maken we gebruik van Team Foundation Services.

Ook de OTAP-opzet hadden wij nog niet omschreven in het onderzoeksrapport. Dit is echter wel iets dat zeer nuttig is.

Bij het onderzoeksrapport hebben wij het niet gehad over meerdere servers. Dit is wel iets dat nodig is omdat niet alles op dezelfde server kan draaien (build, test, deploy, acceptatie, productie)

In het onderzoeksrapport hebben we aangegeven dat we Spint gingen gebruiken. Dit hebben we gewijzigd naar TSLint omdat Angular2 met TypeScript werkt. Niet met normale JavaScript.

Code review, issue tracking, backlog

In het onderzoek hebben we aangegeven dat we code reviews doen in GitHub. Dit doen we voor de Angular2 Front-end, maar voor de C# back-end doen we dit in TFS. Dit doen we omdat 42WindMills hun code wordt gepubliceerd naar TFS en dat de hoofd repository is.

Voor issue tracking gebruiken we ook TFS voor de back-end en voor de front-end gebruiken we GitHub. De issue tracking hebben we niet behandeld in het onderzoeksrapport. Issue tracking gebruiken we om wijzigingen en feature aanvragen in een overzichtelijke manier te weergeven.

Voor de product backlog gebruiken we Trello dit omdat we op TFS maar 1 account hebben en het hierdoor lastiger is om te zien voor wie welk product backlog item / taak is.

Uitleg configuratie ontwikkelstraat

De ontwikkelstraat draait vooral om de Continuous integration tool “Jenkins”. Zoals hierboven al is gezegd vraagt Jenkins constant aan de code repository of er wijzigingen zijn. Als deze er zijn dan neemt Jenkins actie. Dit doet Jenkins natuurlijk niet uit zichzelf, hier laten we zien wat we hebben gedaan om de juiste stappen uit te laten voeren.

Globale Configuratie Server

Eerst moest Jenkins worden geïnstalleerd (voordat Jenkins geïnstalleerd wordt moet de Java SDK geïnstalleerd zijn) en de nodige tools voor de applicatie. Voor de front-end was dit NodeJS (De Node Package Manager met name). Met NPM kunnen we angular-cli installeren, deze zullen we gebruiken bij de build proces om de verschillende tools te gebruiken die we nodig hebben bij de front-end. Voor de back-end was het iets meer namelijk ook nog MSBuild, NUnit en een C# compiler. Dit hebben we uiteindelijk gedaan door Visual Studio te installeren op de server. Deze installeerde automatisch de andere benodigdheden. Als webserver hebben we IIS geïnstalleerd. We installeren ook GIT op de server om te kunnen pullen van de repository. We hebben ook een webbrowser nodig om de tests te draaien op onze server, we gebruiken hiervoor Chrome. Bij de bijlage installaties wordt de installatie procedures hiervoor uitgelegd.

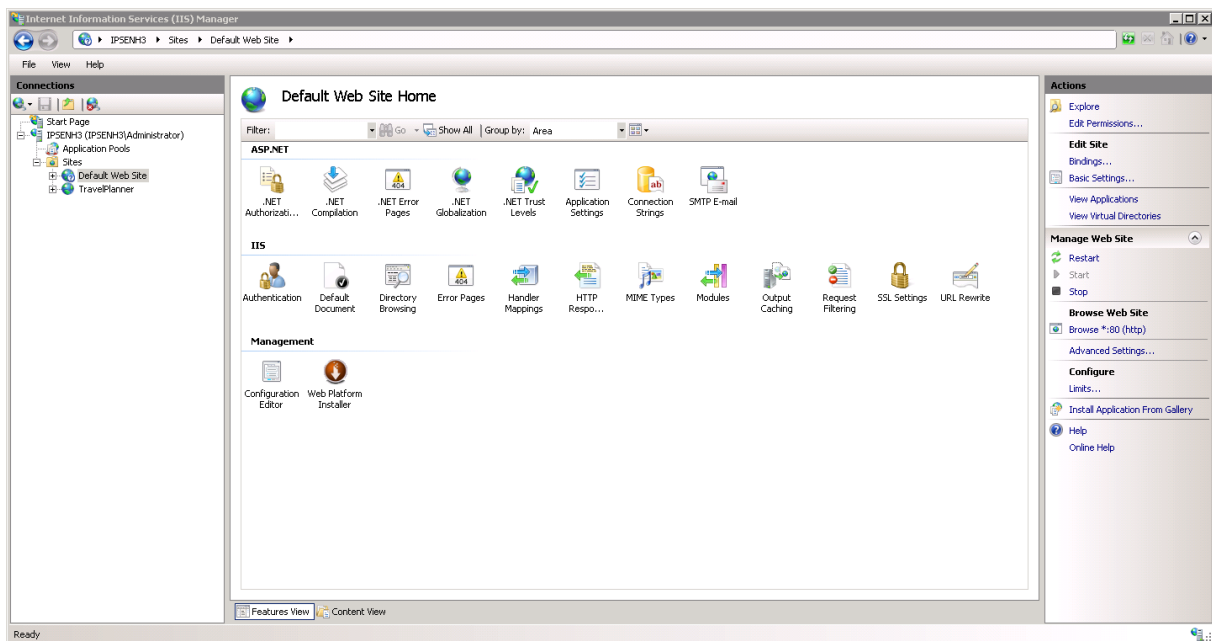
Globale Configuratie Lokale Machine

Op onze lokale machine installeren we de JetBrains WebStorm IDE, GIT, Google Chrome, Visual Studio en NodeJS. Bij de bijlage installaties wordt de installatie procedures hiervoor uitgelegd.

Configuratie Internet Information Services

We draaien onze applicatie op IIS op onze server. Hiervoor gebruiken we de webserver role in server manager. We openen de server manager en gaan naar roles, hier zullen we de webserver role toevoegen. Als we op add roles klikken kunnen we kiezen voor web server (IIS) en deze installeren. Nu is IIS geïnstalleerd op de server, maar we moeten ook de IIS Management console installeren om IIS te configureren. In server manager kunnen we naar add role services gaan en deze installeren. Nu kunnen we de IIS Manager vinden in Administrative Tools.

In de IIS Manager gaan we naar IPSENH3 -> Sites -> Default Web Site, en we zetten index.html tot bovenop in de lijst. Nu kan de Angular applicatie meteen geladen worden als we naar localhost gaan op de server. Deze wordt geladen uit "C:\inetpub\wwwroot", dus hier wordt de angular applicatie gedeployed.



De TravelPlanner applicatie en de API worden ook met behulp van IIS gedeployed. Deze bestanden zijn te vinden in C:\Users\Administrator\Desktop\TP_Jenkins_Build_PublishedWebsites.

Configuratie Jenkins en tools

We hebben eerst de Java SDK geïnstalleerd zodat we vervolgens Jenkins kunnen installeren en gebruiken. Jenkins draait op port 8080, dit laten we ook zo staan. Deze port hebben we wel open gemaakt met de firewall zodat we Jenkins kunnen benaderen buiten de server. Nadat de installatie is voltooid kunnen we navigeren naar localhost:8080. We moeten eerst Jenkins unlocken met een key, Jenkins geeft zelf de locatie van de key. Na deze stap kunnen we kiezen welke plugins we nodig hebben, Jenkins bevat veel nuttige plugins. Jenkins installeert standaard ook enkele plugins voor ons, en andere plugins die we ook nodig hadden hebben we zelf erbij gezet. We hebben de volgende plugins geïnstalleerd en gebruikt: Git Plugin, Nunit Plugin, MSBuild Plugin en SonarQube Plugin.

Vervolgens krijgen we een scherm te zien om een nieuwe gebruiker (Admin) aan te maken, we vullen de gewenste gegevens in en deze bevestigen. Hierna is Jenkins klaar om te gebruiken.

Het is aan te raden om Jenkins te installeren als een service, dit kunnen we doen na het installeren van Jenkins zoals eerder beschreven. We navigeren dan naar "Manage Jenkins" en hier kunnen we kiezen om Jenkins te installeren als een service. Dit doen we om Jenkins in de achtergrond te laten draaien en het automatisch starten bij startup.

In "Global Tool Configuration" configureren we de tools die we zullen gebruiken, we kunnen een path geven naar de locatie van de tool, of Jenkins deze tool automatisch laten installeren. We hebben in dit geval een path gegeven naar de verschillende tools. We willen ook dat we een notificatie krijgen als een build faalt, emails zijn goed hiervoor. In "Configure System" kunnen we de e-mail notificatie configureren met een SMTP-server, we hebben "smtp.gmail.com" gebruikt op port 465.

S	W	Name	Last Success	Last Failure	Last Duration	Fav
S		rzn-angular	8 days 19 hr - #53	40 min - #55	7 min 18 sec	
S		TravelPlanner	2 hr 28 min - #118	2 hr 32 min - #116	2 min 32 sec	

Bij de default pagina van Jenkins krijg je de jobs te zien, blauw betekent dat de laatste build succesvol was en rood betekent dat de laatste build niet succesvol was. We hebben een job voor de front-end en de back-end.

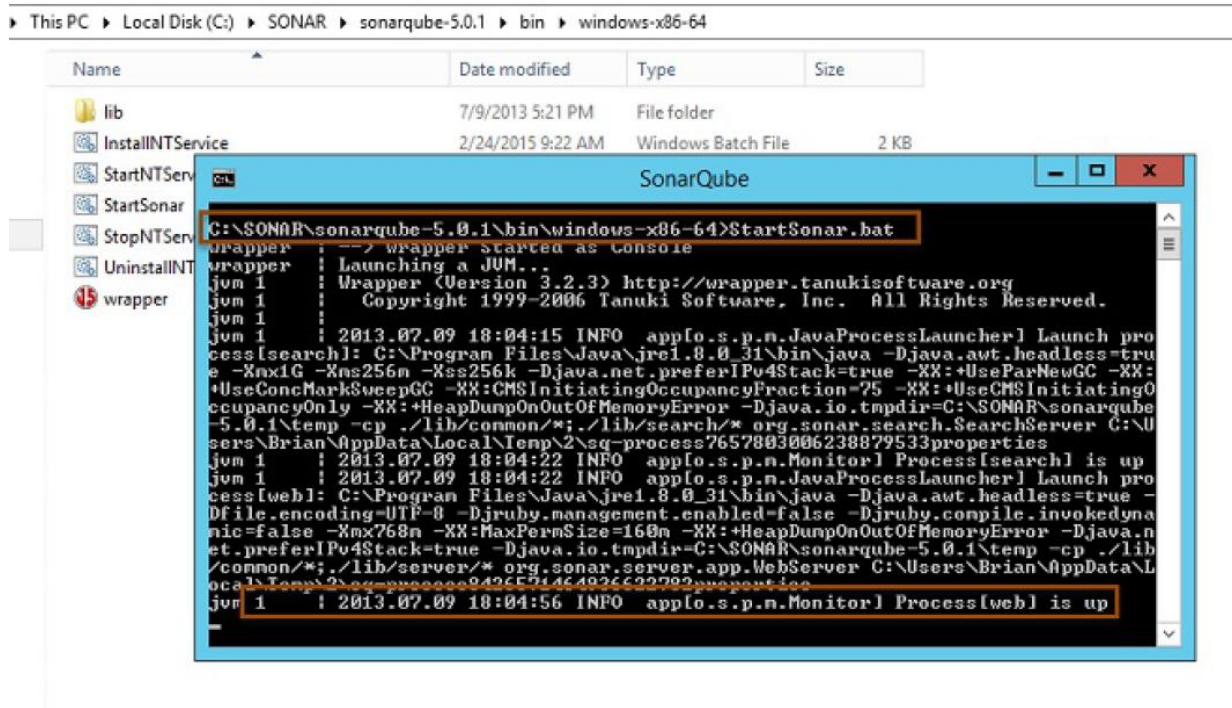
Configuratie SonarQube

Voordat je met de configuratie begint, volg eerst de installatie van SonarQube in het hoofdstuk 'SonarQube installatie'.

Je hebt nu twee uitgepakte mappen op de locatie C:\SonarQube. Open pgAdmin 4 en maak een nieuwe database aan met de naam sonar. Open het configuratiebestand van de SonarQube server. Deze is te vinden in C:\SonarQube\sonarqube-x.x\conf\sonar.properties. Bij het stukje over user credentials vul je als gebruikersnaam postgres in en als wachtwoord het wachtwoord wat je tijdens de installatie hebt gekozen. Dit was bij ons Ipsenh3289. Onder het PostgreSQL gedeelte hebben we de jdbc.url uitgecommentarieerd en als url het volgende opgegeven:
jdbc:postgresql://localhost/sonar. Sla dit bestand op.

Download vervolgens de SonarQube C# plugin van <https://www.sonarsource.com/why-us/products/languages/csharp.html> en plaats het uitgepakte .jar bestand uit in C:\SonarQube\sonarqube-x.x\extensions\plugins.

Voer vervolgens het volgende commando uit in een terminal: cd C:\SonarQube\sonarqube-x.x\bin\{OSVERSIE}. De OSVERSIE is het operating systeem waarop SonarQube draait. In ons geval was dit Windows 64 bit. Het commando werd: cd C:\SonarQube\sonarqube-x.x\bin\windows-x86-64. Start daarna het bestand StartSonar.bat en wacht tot de SonarQube server gestart is.



De SonarQube server is nu gestart en is te vinden op poort 9000. Er kunnen nu projecten worden gescand. Laat de server aan staan, anders werkt de scanner niet meer.

Er volgen nu drie stappen om een scan te krijgen:

1. Start de SonarQube begin phase door het volgende commando uit te voeren
MSBuild.SonarQube.Runner.exe begin /key:{SonarQube project key} /name:{SQ project name} /version:{SQ project version}.
 - a. /key: de key waar het project aan verwant is (mag leeg zijn)
 - b. /name: De naam van het project. In ons geval is het TravelPlanner
 - c. /version: De versie van het project
2. Start het normale build traject. Omdat we dit niet vanuit Visual studio bouwen doen we het met een ander terminal. Er zijn dus nu 3 terminals geopend. De SonarQube server, de SonarQube begin phase en het nieuwe scherm om de build te starten. De build start je met het volgende commando: C:\Program Files (x86)\MSBuild\14.0\Bin\msbuild C:\Users\Administrator\.jenkins\workspace\TravelPlanner\TravelPlanner.sln. De build wordt nu gestart en uitgevoerd.
3. Start nu de SonarQube end phase door het volgende commando uit te voeren: MSBuild.SonarQube.Runner.exe end. Als dit voltooid is kun je je project terugvinden in de webinterface van SonarQube (localhost:9000).

Configuratie Front-end Job

Voor de front-end maken we een nieuwe job in Jenkins. Hiervoor gebruiken we een “Freestyle project”. Deze job gaat verschillende taken voor ons automatisch uitvoeren, en dit is ook de essentie van een integration tool.

We geven het project een naam en bij “Source Code Management” geven wij de gewenste git repository, we voegen toe een credential (username en password) en de branch waaruit we een pull request willen doen als er veranderingen zijn. De repository is:
<https://github.com/antonsteenvoorden/rzn-angular>

The screenshot shows the 'Source Code Management' configuration page in Jenkins. The 'Git' radio button is selected. Under 'Repositories', the 'Repository URL' is set to 'https://github.com/antonsteenvoorden/rzn-angular/' and the 'Credentials' dropdown is set to 'antonsteenvoorden (jenkino)'. There are buttons for 'Advanced...', 'Add Repository', and 'Add Branch'. Under 'Branches to build', the 'Branch Specifier (blank for \'any\')' is set to '*/development'. There is a button for 'Add Branch'. The 'Repository browser' is set to '(Auto)'. There is an 'Add' button for 'Additional Behaviours'. The 'Subversion' radio button is unselected.

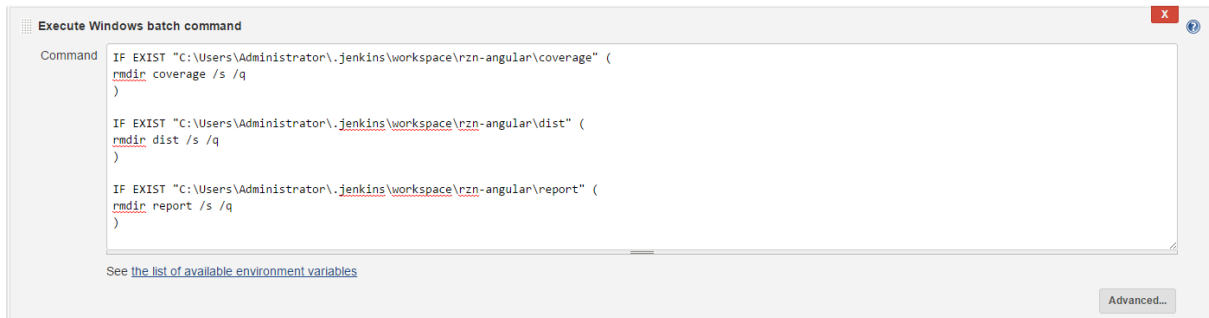
Bij “Build Triggers” gebruiken we de poll SCM optie. Hierin zetten we “H/5 * * * *”, dit stukje code zorgt ervoor dat Jenkins elke 5 minuut zal controleren als er veranderingen zijn in de gegeven branch van de repository. Als er veranderingen zijn dan wordt een pull request uitgevoerd. Vervolgens gaat de build proces starten.

The screenshot shows the 'Build Triggers' configuration page in Jenkins. The 'Poll SCM' checkbox is checked. The 'Schedule' field is set to 'H/5 * * * *'. Below the schedule field, it says 'Would last have run at maandag 9 januari 2017 10:55:04 uur CET; would next run at maandag 9 januari 2017 10:55:04 uur CET.' There is an 'Ignore post-commit hooks' checkbox which is unselected.

Bij “Build” zetten we alle taken die uitgevoerd moeten worden tijdens het build proces. We gebruiken “Execute Windows Batch Commands” voor alle stappen van de build proces. Hierin vullen we de gewenste commando’s om een taak uit te voeren. Voor de meeste taken gebruiken we de angular-cli, dit is een command line interface voor angular. Het build proces start meteen nadat Jenkins een pull request heeft gedaan na het vinden van veranderingen in de repository. Bij het falen van een taak stopt de build proces meteen en er wordt een email gestuurd naar de ontwikkelaars en die kunnen dan in de logs zien wat fout ging. Alleen voor het linten van de code geldt dit niet, dit hebben we bewust gedaan omdat lint fouten geen problemen zou leveren bij het draaien van de angular applicatie.

We voeren 9 losse taken uit. De taken die uitgevoerd worden zijn de volgende:

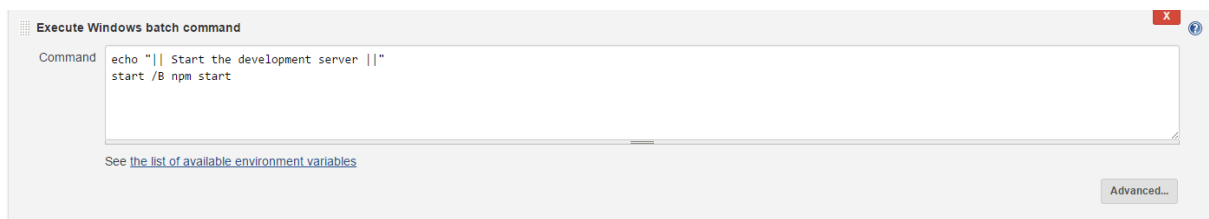
1. We beginnen eerst met het verwijderen van enkele folders en hun inhoud, de oude coverage, report en dist folder. Deze worden opnieuw gemaakt bij de komende stappen.



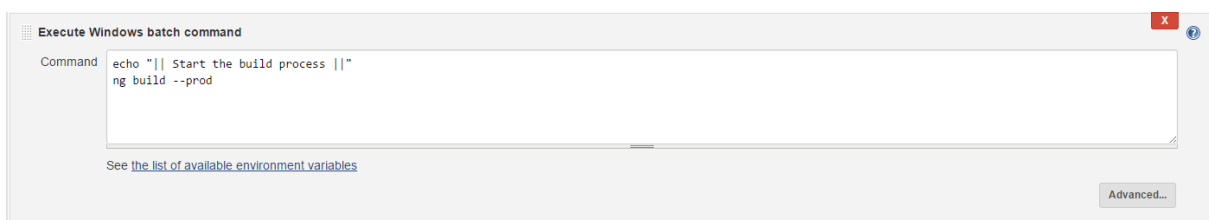
2. "npm install" gaat alle dependencies installeren die nodig zijn voor de angular-cli project. Deze dependencies zitten in de "package.json" file in het project.



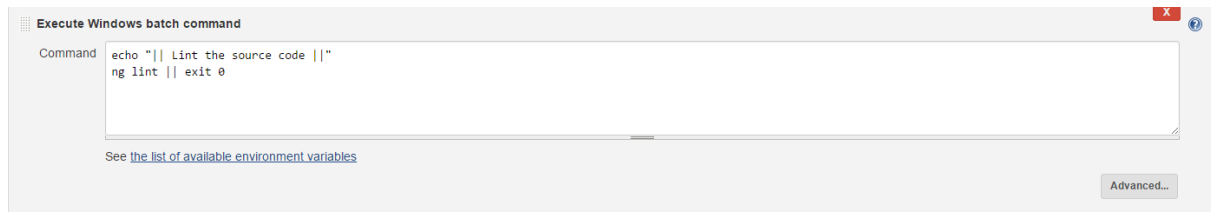
3. Met "npm start" starten de development server waarin de angular applicatie gaat draaien. Deze willen we draaien in de achtergrond om door te kunnen gaan naar de volgende taken, om dit te doen gebruiken we "/B". We starten de development server om later hiermee tests te kunnen uitvoeren.



4. Om een build te maken van onze front-end applicatie gebruiken we angular-cli. Met "ng build --prod" maakt een build van de angular-cli applicatie. Deze geeft als output een "dist" folder in het project. De inhoud van deze folder kunnen we deployen naar een webserver.



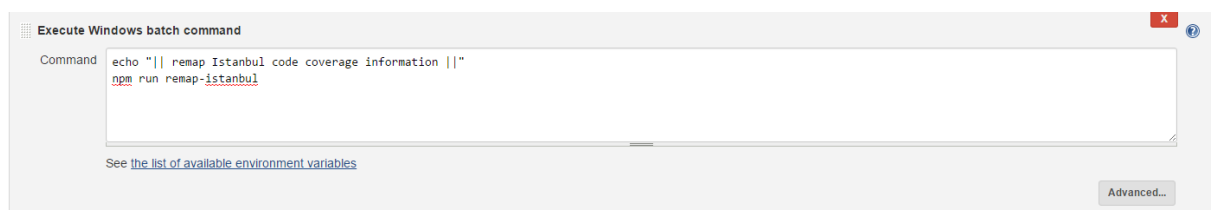
5. “ng lint” gaat onze project linten (static code analysis). Hiervoor wordt TSLint gebruikt, deze tool komt met de angular-cli. Zoals eerder gezegd willen we dat bij het falen van deze stap, het build proces niet stopt. Hiervoor zetten we een “exit 0”, dit zorgt ervoor dat er altijd een false teruggestuurd wordt. Een false betekent dat er geen fouten gevonden zijn en een true wel fouten zijn gevonden. Door altijd een false terug te geven stopt de build proces nooit bij deze stap.



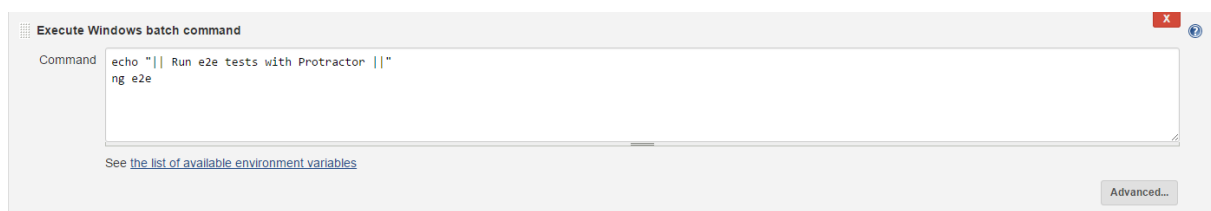
6. “npm run test” gaat unit tests uit voeren. Deze haalt op alle bestanden die beëindigen met “.spec.ts”, in deze bestanden zijn de tests beschreven. Om de tests te draaien wordt Karma (test runner) en Jasmine (testing framework) gebruikt. Deze tools komen ook met de angular-cli project. Tevens wordt hier de coverage en report van test resultaten gecreëerd.



7. Een remap tool voor Istanbul code coverage. De output is een html-file die we kunnen openen en de code coverage bekijken.



8. “ng e2e” gaat end-to-end tests uitvoeren. Hiervoor wordt Protractor gebruikt, ook een tool dat met angular-cli project komt. Deze haalt alle bestanden die beëindigen met “e2e-spec.ts”. Hier wordt ook een report van de test gecreëerd, met screenshots erbij.

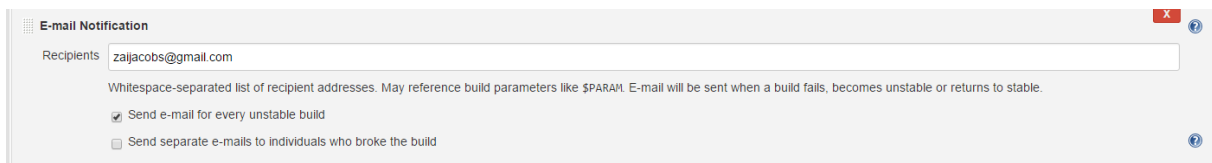


9. Als alle vorige taken uitgevoerd zijn zonder enige fouten, wordt deze laatste stap uitgevoerd. Deze stap zal de build output deployen naar IIS op de server.



Door elke stap los van elkaar in een windows batch command te zetten hebben we ervoor gezorgd dat de build meteen stopt bij het vinden van een fout, omdat het op die moment een true zal retourneren voor die stap. Een true betekent dat iets in het proces fout ging.

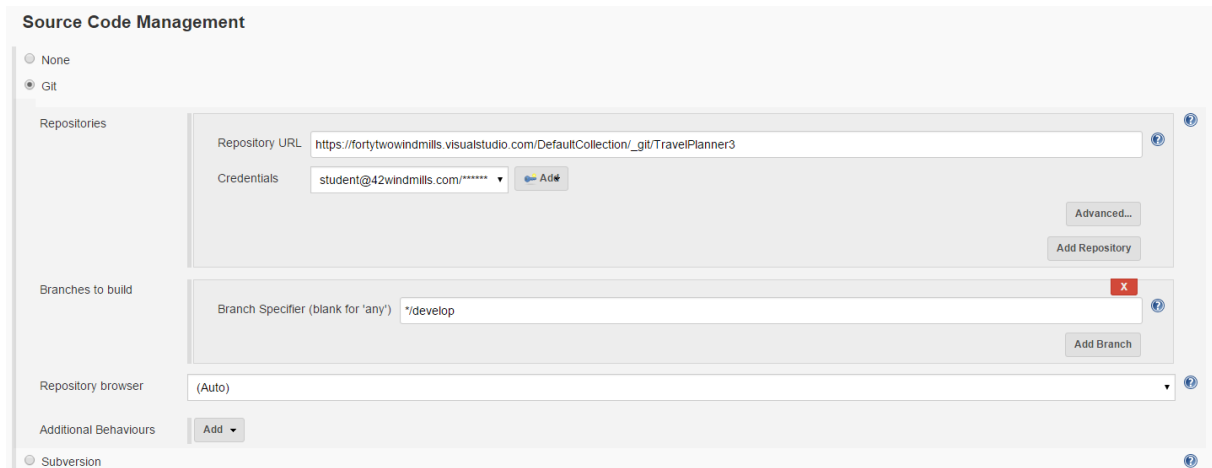
Bij “Post-build Actions” zetten we de email adressen waarvan we een notificatie willen sturen als een build faalt. Tevens wordt er ook een notificatie gestuurd van de eerste succesvol build als vorigen gefaald zijn.



Configuratie Back-end Job

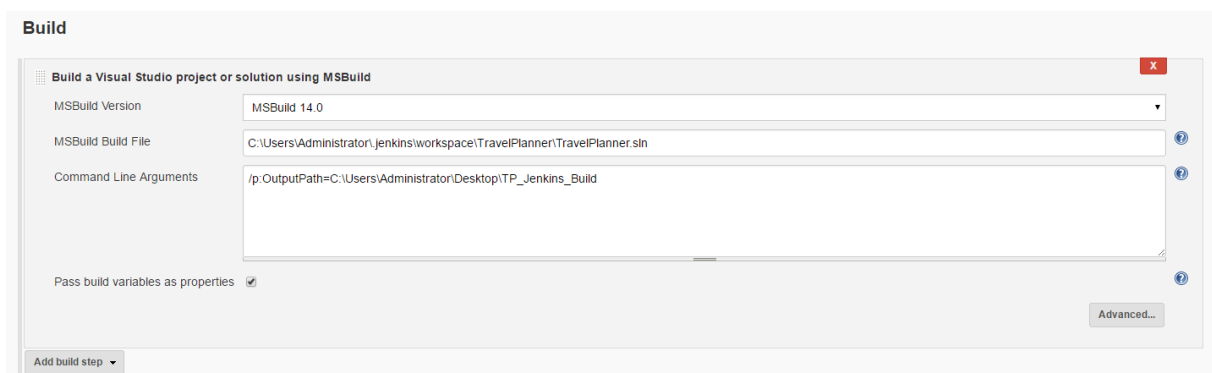
Voor de back-end maken we een nieuwe job in Jenkins. Hiervoor gebruiken we een “Freestyle project”.

We geven het project een naam en bij “Source Code Management” geven wij de gewenste git repository, we voegen toe een credential (username en password) en de branch waaruit we een pull request willen doen als er veranderingen zijn. De repository is:
https://fortytwowindmills.visualstudio.com/_git/TravelPlanner3



Bij “Build Triggers” gebruiken we de poll SCM optie. Hierin zetten we “H/5 * * * *”, dit is precies hetzelfde als gedaan bij de front-end.

Bij “Build” kiezen we Build a Visual Studio Project or solution using MSBuild. We kiezen de gewenste MSBuild versie en geven een path naar de “.sln” build file die in onze project zit. Als command line arguments geven we de path waar we de output willen zetten.



We zetten eventueel in “Post-build Actions” de email adressen van de ontwikkelaars om notificaties te krijgen.

Bijlage installaties

NodeJS installatie

We downloaden de laatste versie van NodeJS op "<https://nodejs.org/en/>". We krijgen een executable die we op onze server zullen installeren. Na het voltooien van deze installatie kunnen we meteen "npm" gebruiken, dit is een package manager voor Javascript.

Met "npm" gaan we vervolgens angular-cli globaal installeren op onze server installeren, dit doen we met "npm install angular-cli". Tevens zullen we dit ook op onze lokale machine moeten installeren. De verschillende tools die we zullen gebruiken bij onze front-end applicatie krijgen we als dependencies in onze project, deze installeren we door in onze project een "npm install" te doen. Deze tools zijn Protractor, Jasmine, Karma, TSLint en Remap-Istanbul.

Jenkins Installatie

Voordat we Jenkins installeren hebben we Java SDK 8 nodig, we downloaden de laatste versie van "<http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>" en deze installeren op onze server. Vervolgens gaan we Jenkins downloaden van "<https://jenkins.io/>", we kiezen hier de LTS release. In de locatie waar deze is gedownload openen we cmd en voeren uit "java -jar jenkins.war", het installatie proces zal nu starten. Als deze voltooid is kunnen we op localhost naar port 8080 Jenkins benaderen.

SonarQube installatie

Voordat SonarQube geïnstalleerd kan worden moeten er eerst een aantal andere programma's gedownload worden. De eerste is Java 7 runtime of hoger. Deze is te downloaden van <https://www.java.com/nl/download/>. Daarna moet er een databaseomgeving gekozen worden.

Aangeraden is PostgreSQL, omdat deze gemakkelijk te installeren en configureren is. Het gaat hier om PostgreSQL versie 9.x. Download hierbij ook pgAdmin 4. Deze zit standaard in het installatiepakket. PostgreSQL is te downloaden vanaf <https://www.enterprisedb.com/downloads/postgres-postgresql-downloads#windows>. Selecteer de juiste gegevens voor de download. Download en installeer PostgreSQL met de standaardinstellingen.

Er is ook een webbrowser nodig. Hiervoor voldoet Google Chrome. Zie voor de installatie van Google Chrome het hoofdstuk 'Google Chrome installatie'.

Download vervolgens de laatste SonarQube server vanaf <https://www.sonarqube.org/downloads/>.

Pak de SonarQube bestanden uit in de map C:\SonarQube.

Download de laatste SonarQube scanner voor MSBuild vanaf <https://github.com/SonarSource-VisualStudio/sonar-scanner-msbuild/releases/download/2.2/sonar-scanner-msbuild-2.2.0.24.zip> en pak deze uit in de map C:\SonarQube.

Je bent nu klaar met de installatie van SonarQube.

GIT installatie

We downloaden de laatste versie van GIT van "<https://git-scm.com/downloads>". Deze installeren we vervolgens op onze lokale machine en onze server.

JetBrains WebStorm installatie

We downloaden de WebStorm IDE van "<https://www.jetbrains.com/webstorm/>" en installeren deze op onze lokale machine.

Google Chrome installatie

We installeren de Chrome webbrowser op onze lokale machine en server, deze vinden we op "<https://www.google.com/chrome/browser/desktop/>". Chrome zal gebruikt worden om unittests en e2e tests uit te voeren op onze server, en lokaal als we deze willen doen.

Visual studio installatie

Het maakt in principe niet uit welke versie van Visual Studio er gedraaid wordt. We hebben tijdens het ontwikkelen gebruik gemaakt van Visual Studio 2014 en hoger. Dit raden wij dan ook aan om te gebruiken. De installatie van Visual Studio spreekt voor zich. Een kopie kan worden gedownload vanaf <https://www.visualstudio.com>. Kies hier voor de Community Edition. Tijdens de installatie kunnen alle instellingen op blijven zoals ze zijn.