TOOLS

Hudson is de nieuwkomer op het gebied van continuous build en continuous integration. Er zijn al veel tools op dit gebied beschikbaar. Hudson is de nieuwste en stijgt snel in populariteit. Wat zijn CI tools en wat maakt Hudson anders dan CruiseControll, Continuum of LuntBuild? En hoe werkt Hudson? Dit zijn de vragen die ik in dit artikel probeer te beantwoorden.

Build Automation met Hudson

Robbrecht van Amerongen
Projectmanager
AMIS Services/www.amis.nl

Waarom een buildtool gebruiken?

Integratie van code is altijd lastig. Zeker als het gaat om omvangrijke projecten, een groot aantal componenten en libraries en een veelvoud van ontwikkelaars. Het handmatig integreren, testen en opleveren van de code (artefacten) uit dergelijke trajecten is een secure en tijdrovende bezigheid. In de begindagen van J2EE had je geluk als er in je team één ontwikkelaar was die een build met constante kwaliteit kon maken. Helaas heeft deze ontwikkelaar ook wel eens een rotdag (of nog erger … vakantie). Deze activiteit wordt vaak ervaren als complex en tijdrovend en legt een beslag op het meest ervaren teamlid. Dit zorgt ervoor dat oplevering en integratie vaak tot het eind wordt uitgesteld. En dan komen pas de problemen met oplevering en integratie naar voren. Naast de afhankelijkheid van dit teamlid is er de onwenselijke afhankelijkheid van een lokaal ontwikkel-werkstation. Deze omgeving is frequent aan verandering onderhevig (libraries, systeem settings etc). Ook bevat deze omgeving instellingen die voor de ontwikkelaar relevant zijn maar niet geschikt voor de build. Voor het maken van een build moet er juist gebruik gemaakt worden van een omgeving die qua configuratie schoon en stabiel is.
Je kunt voor de oplevering van bedrijfskritische applicaties tegenwoordig niet meer afhankelijk zijn van één uniek teamlid en/of één werkstation. De introductie van tools als Make, Ant en Maven hebben het opleveren van code een stuk eenvoudiger gemaakt. Deze tools maken het mogelijk om onafhankelijk van een specifieke IDE, werkstation of specifieke ontwikkelaar een artefact (JAR, WAR, EAR) op te leveren. Hiermee is de build verplaatst van het werkstation van de ontwikkelaar naar de build server (continuous build server).

De complexe en veranderende omgevingen, grote aantallen afhankelijkheden en de grotere, soms internationaal opererende, ontwikkelteams maken frequente integratie noodzakelijk en zelfs onmisbaar. Op deze plaatsen spelen tools voor continue integratie een hoofdrol. Deze tools zorgen voor volledige automatisering van het compileren, integreren, testen en opleveren van artefacten. Ze maken het mogelijk om op vrijwel ieder gewenst moment een versie van het eindproduct op te leveren.

Hoe werken tools voor continue integratie

De tools voor continue integratie ondersteunen de software developement cycle vanaf genereren, build, test, package tot en met deployment in je corporate library, installatie op je applicatieserver en genereren van documentatie. Uitgangspunt is dat de code wordt opgeslagen in een source repository (bijvoorbeeld Subversion). Vanuit de codebase bouwt de CI tool door middel van een build script een artefact op een schone machine. Het team krijgt feedback van het resultaat van de buildactie (succes, fail of error). Het gebruik van een CI tool heeft de volgende voordelen:

- de build van de artefact wordt automatisch uitgevoerd.
- de build test zichzelf.
- de kwaliteit van de build is voorspelbaar.
- snelle feedback over de kwaliteit van de build en de kwaliteit van de opgeleverde artefacten.
- mogelijkheid tot verzamelen van build-data en data voor analyse van kwaliteit.
- ontwikkelaar kan zich richten op het creëren van business value in plaats van het build proces.
- snel, uniform en eenvoudig deployment van producten in je corporate library.
- snel, uniform en eenvoudig deployen van je applicatie op je J2EE container.

Aan de slag

Wat heb je nodig om een project op te nemen in een continue integratie?
Het project moet zijn opgenomen in een source repository. Vrijwel alle build tools ondersteunen Subversion en CVS.
Een build server met hierop de JDK versie die door de projecten gebruikt gaat worden. Sommige tools ondersteunen het gebruik van verschillende JDK versies naast elkaar.

Een gescript project. Bijvoorbeeld ANT, Maven (Maven is in veel buildtools defacto standaard).
Behalve compileren zorgt de buildtool onder andere ook voor: librarymanagement; valideren van interne consistentie, unit testing; packaging (Jar, War, Ear), deployment, functionele test, management van opgeleverde artefacts en genreren van documentatie.

De trigger voor het starten van de build is het committen in de source repository, een dagelijks tijdschema (b.v. iedere nacht om 1.00 uur) of een handmatige start via de web-interface.
Afhankelijk van de build-tool wordt er na afloop van de build cyclus een signaal (e-mail, IRC, rss) gegeven aan de ontwikkelaars. In dit signaal wordt het resultaat van de build doorgegeven. Bij een build-interval van een uur is dus iedere ontwikkelaar binnen een uur na inchecken op de hoogte van het resultaat van zijn actie (succes, fail of build error).

Wat doet Hudson

Hudson is buitengewoon gebruiksvriendelijk in installatie. De gedownloade Hudson.war is te starten met het commando java -jar Hudson.war. Door de ingebouwde Winstone servlet container fungeert de war ook direct als applicatieserver. Er is verder geen andere configuratie nodig om de Hudson draaiend te krijgen. Hudson draait ook binnen een servlet container (2.4/JSP 2.0) zoals GlassFish, Tomcat 5+ of JBoss. De configuratie, build-log's, statistieken en gegenereerde artefacts worden file-based opgeslagen. Je hoeft dus geen database te configureren.

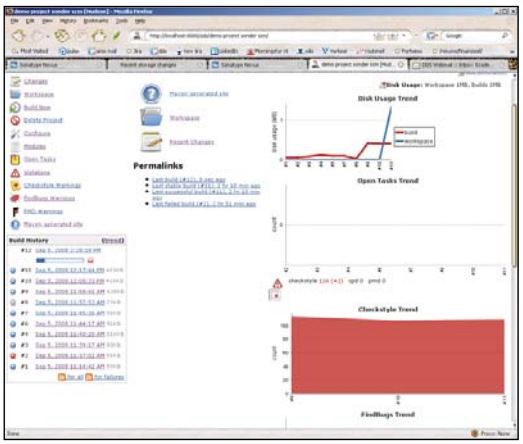
Na installatie is Hudson direct bruikbaar. Standaard staat gebruikersbeheer uit waardoor iedereen die de op de Hudson site komt direct alle rechten heeft. Voor gebruikersbeheer kan ook gebruik gemaakt worden van de gebruikerscontext van de servlet container of een externe LDAP. Hierna kan er een uitgebreide gebruikersautorisatie ingesteld worden met groepen en rollen. Dit kan zelfs per project worden ingesteld.

Voor je Hudson voor build kunt gebruiken moet je nog twee zaken instellen, te weten: de locatie van je JDK en de locatie van de build tool (MAVEN_HOME/ANT_HOME). Verschil met veel andere build tools is dat je hier verschillende versies tegelijk kunt opnemen. Hiermee kan je een hele range aan JDK's en build tools tegelijk gebruiken. Vervolgens is het nog handig om een SMTP adres op te geven zodat je als ontwikkelaar mails met build-status kunt ontvangen. Hierna kan je aan de slag.

Taken

In de basis verschilt Hudson niet erg veel met concurrerende OpenSource build systemen. Belangrijkste verschil is de grote hoeveelheid plugins die er beschikbaar zijn. Via het configuratiescherm zijn tientallen plugins te installeren. Met deze plugins kun je veel taken regelen die je anders in je build script moet configureren (de pom.xml of de build.xml).
Deze taken zijn:

- uitchecken van je sources uit scm.
- Uitvoeren van je tests, test rapportages, historische test rapportages.
- Rapportages over kwaliteit van de build en historische rapportages.
- Thresholds instellen voor controles op standaarden zoals code test coverage, coding standards, PMD/CPD errors.
- Genereren van todo lijsten.

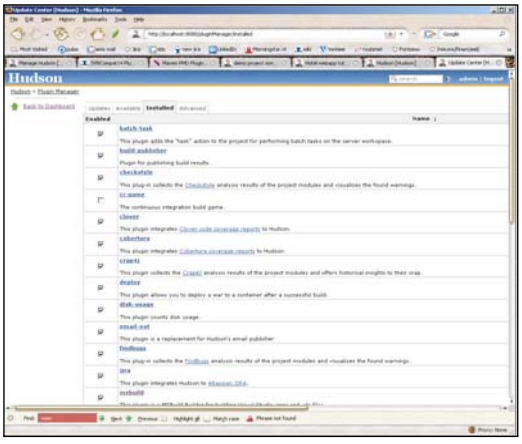


De interface van Hudson is wel voor verbetering vatbaar en heeft een grote leercurve

- Starten van een andere machine (b.v. vmware),
- starten applicatieserver,
- deployment war op applicatieserver.
- Deployment van artefacten in corporate repository.
- Bewaren van alle geproduceerde builds, testresultaten en documentatie.
- Fingerprinting van builds. Hiermee krijgt iedere build automatisch zijn eigen nummer en wordt zo vergelijkbaar.
- Aanbrengen van een TAG op een succesvolle build.

Werkt het allemaal?

Voor deze review gebruik ik een demo project van JavaWorld* (Java6, Maven 2 en subversion) en Hudson gedeployed in Tomcat6. Via de webinterface kun je hierna de JAVA_HOME en in mijn geval MAVEN_HOME opgeven en je kunt aan de slag. Om feedback per e-mail te krijgen kun je de url van de smtp server invullen.
Ik voeg het nieuwe Maven2 project toe door het opgeven van subversion url naar de pom.xml van dit project. Het is niet noodzakelijk (zoals bij Continuum) om je SCM url in je pom.xml op te nemen. (Dit kan bij herbruikbare pom.xml bestanden handig zijn.) Hierna zat het project in de Hudson Build.



De Hudson plugin manager

Via het configuratiescherm kan ik tientallen plugins downloaden. Ik kies voor de plugins: PMD, JIRA, Build Publisher, Checkstyle, Cobertura, deploy plugin, Subversion tagging, VMware plugin.
Met deze plugins wil ik mijn project bouwen, checken op fouten, publiceren van test resultaten, na build deployen in onze corporate library, een VMware omgeving starten en de gebouwde WAR hierin deployen. Na de build wordt JIRA bijgewerkt en de documentatie gepubliceerd op een website.

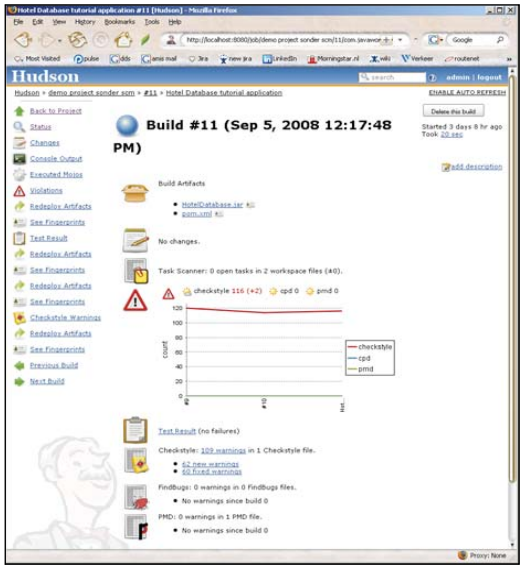
Na een reboot zijn alle plugins geïnstalleerd. De configuratie van de plugins is WEB-based. De uitleg bij de plugins is soms erg beperkt waardoor je een aantal malen moet experimenteren, wil je de plugin werkend krijgen. Helaas bleken, in de door mij gebruikte versie van Hudson, een aantal van de door mij gekozen plugins niet te werken (VMware en Subversion tag).

Een mooie feature is de mogelijkheid om verschillende JVM versies en build tools te configureren. Hiermee kun je een project builden in verschillende applicatie-configuraties (b.v. java4+Windows/Java5+Linux). Je kunt een matrix met deze combinaties opgeven waarna alle combinaties worden getest tijdens de build. Via slaves kun je ook Hudson instanties op andere omgevingen (b.v. Linux) koppelen en ook op die omgeving een builden.

Het starten en volgen van de build is erg eenvoudig. Het starten van de build is het aanroepen van een helder herkenbare url zoals bijvoorbeeld 'http://HudsonServerurl/job/myapplication/build'. Hiermee kan je ook vanuit andere tools een build triggeren.

De interface van Hudson is wel voor verbetering vatbaar en heeft een grote leercurve. Het is erg lastig je weg te vinden tussen de builds en de configuraties aangezien de schermen vrijwel identiek zijn. Er is geen duidelijke scheiding tussen het menu en de rapportages. Hierdoor is het niet duidelijk of een bepaald element nu bij het menu of bij een rapportage hoort. Aangezien het publiceren van de documentatie is geïntegreerd in de buildtool zullen ontwikkelaars deze site meer gaan bezoeken dan de sites van concurerende buildtools. Een heldere interface is hier dus

belangrijker. Het is niet moeilijk om het overzicht kwijt te raken in de vele schermen. Tot slot zijn de gebruikte iconen naar mijn smaak een beetje ouderwets en veel te groot. Het ontbreken van een heldere schermverdeling maakt dat je als gebruiker informatie niet terug kunt vinden op plaatsen waar je het zou verwachten.



Wat zijn de voordelen:

Hudson regelt veel zaken voor je die je anders in het reguliere buildscript moet opnemen. Het configureren van deze zaken is erg eenvoudig. Taken die je via Hudson kunt configureren:

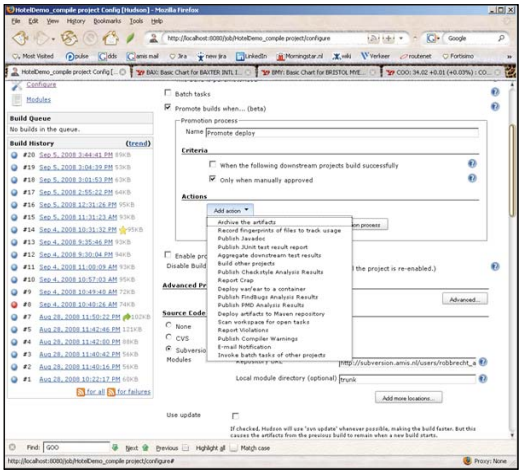
- Genereren en publiceren van test rapportage;
- Genereren en publiceren van documentatie;
- Publiceren van de gebouwde artefacten in de corporate artefact library;
- Koppelen van de build resultaten met JIRA;
- Onderscheid maken tussen builds met een buildnumber en terughalen van resultaten van oude builds;
- Promoten en installeren op een applicatieserver van één specifieke build;
- Managen van afhankelijkheden tussen projecten;
- Archiveren van de build resultaten;
- Het builden van je project op verschillende JVM versies en verschillende platformen;
- Instellen van volgorde van de builds;
- Distributie van build taken over verschillende build nodes (slaves);

Het onderbrengen van deze taken in Hudson zorgt ervoor dat je een schoon en efficiënt buildscript krijgt en de taken hierboven optimaal kunt managen. Het buildscript is dus puur gericht op het generen van de applicatie. Alle deployment en rapportage specifieke zaken hoef je niet meer in je buildscript op te nemen. Hierdoor kan je in je project de rol van ontwikkelaar en buildmanager/releasemanager duidelijker scheiden.

Promotion

Een van de krachtige features van Hudson is build-promotion. Dit is een handig mechanisme om handmatig (of zelfs automatisch) gebuilde versies apart te oormerken. De buildmanager kan de betreffende build aanmerken voor promotion. Aan deze promotion kunnen verschillende taken gekoppeld worden. Deze taken zijn te verdelen in de volgende groepen:

- publiceren van resultaten: Javadoc, test resultaten, Checkstyle, FindBugs, PMD, open tasks.
- voorwaardelijk starten van andere projecten: Het, bij succesvolle promotie, starten van een ander project of batch taak.
- deployment van de resultaten: het deployen van de project artefacts (jar,war,ear) naar de corporate library of installatie op een J2EE container.



Vergelijking met andere producten

Hieronder een korte vergelijking tussen Hudson, Continuum, Luntbuild en Cruisecontroll.

Cruise Control

- Een van de eerste build-tools en dat merk je (in een negatieve zin).
- Uitgebreide notificatie mogelijkheden.
- Complexe configuratie/geen web interface voor configuratie.
- Robuust van opzet.
- Kijkt vrijwel continu naar wijzigingen in het SCM systeem.
- Geen mogelijkheid tot dependency management.

Continuum

- Eenvoudige configuratie.
- Eenvoudig om nieuwe projecten toevoegen (voor ANT, Maven 1 en 2).
- Toevoegen van Shell scripts is meer werk.
- Voor kleine tot medium projecten een goede keuze.
- Gebruikersinterface is niet mooi te noemen.
- Geen applicatieserver nodig om te draaien.

Luntbuild

- Mooie grafische interface.
- Mogelijkheid tot het opslaan van artefacten.
- Er zit een goede installer bij.
- LDAP authenticatie.
- Goed dependency management .
- Mogelijkheid tot artefact management.
- Configuratie is te complex voor toepassing in startende teams.

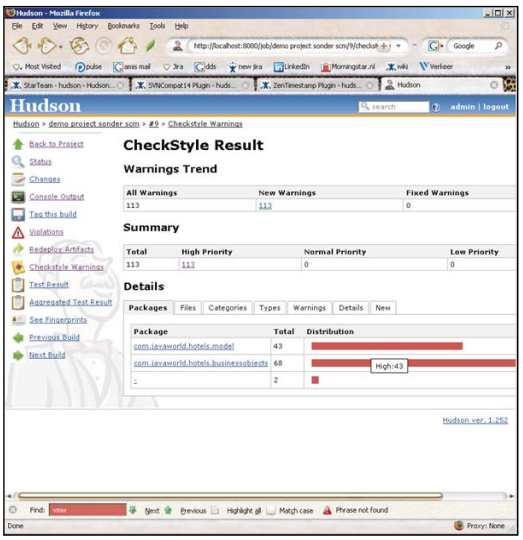
Hudson

- Heel eenvoudig in opzet en installatie.
- Achteraf taggen van build resultaten.
- Publiceert buildresultaten in Hudson zelf.
- Mogelijkheid tot specificeren van pre en post build acties.
- Volledig web-based configureerbaar.
- Nog niet helemaal volwassen (aantal zaken werken niet).
- Promotie van build versies.

Conclusie

In de praktijk heeft Hudson echt toegevoegde waarde in projecten met zeer complexe applicatieconfiguraties en een langdurige en resource-intensieve buildcyclus. Als je als organisatie slechts enkele kleinere projecten in de CI Server wilt opnemen, voldoen de andere producten ook prima. Als je echt heldere scheiding tussen je build en de applicatieontwikkeling wilt hebben is Hudson een aanrader. Naar mijn mening is keuze voor een specifieke build-tool triviaal. Het is veel belangrijker dat het ontwikkelteam een build-tool ook echt gebruikt. Als je echt een keuze moet maken dan is Hudson een goede keuze voor het merendeel van de projecten.

Hudson sluit met de plugins aan op de nu geldende standaarden op het gebied van software ontwikkeling en maakt geen onderscheid tussen koppelingen met commerciële (er is zelfs een



.NET build plugin!) en OpenSource producten. De huidige versie van Hudson is nog duidelijk in ontwikkeling en bevat nog een aantal onhandigheden en fouten die ik zie als kinderziektes. Hudson heeft een zeer actieve gebruikers community en een vrijwel wekelijkse release cycle. De grote hoeveelheid plugins en de omvang en activiteit in de gebruikerscommunity geven Hudson een goede toekomst. Dit geeft mij het vertrouwen dat de minpunten (gebruiksvriendelijkheid en de werking van enkele plugins) snel zullen verbeteren. Hiermee wordt deze tool echt volwassen. Ik zie Hudson over niet al te lange tijd de defacto standaard op het gebied van Continuous Build worden. «

Referenties

https://Hudson.dev.java.net/
http://www.javaworld.com/javaworld/jw-12-2005/jw-1205-maven.html?page=3