Arbeitsprotokoll Master-Thesis

# Arbeitspakete

Aufteilung der Arbeit in Arbeitspakete

## Exposé erstellen

Bis 20.02.2012: Einleitung, Fragestellung, Methodik und Vorgehensweise, Arbeits- und Zeitplanung, Gliederung erstellt.

23.02.2012: Leichte Überarbeitung des Exposés .

## Vergleich von Werkzeuge für Continuous Delivery

### Werkzeugklassen für CD

23.02.2012 Toolliste nach „DevOps in the Cloud“

* Application Containers:
  + Kontainer in der die Anwendung ausgeführt wird.
* Build Tools:
  + Werkzeuge die den Erstellungsprozess von Software als Vorgang unterstützen. Hierzu gehört die Kompilierung des Quellcodes sowie das verlinken an Bibliotheken.
* Code Review:
  + Manuelle Prüfung von Quellcode / Analytische
* Code Insight
* Continuous Integration
* Cloud IaaS
* Cloud PaaS
* Database
* Database Change Management
* Data Center Configuration Automation
* Dependency Management
* Deployment Automation
* Information Sharing
* Installer
* IDE
* Issue Tracking
* Multi-Type
* Password
* Protected Configuration
* Project Management
* Provisioning
* Reporting / Documantation
* Static Analyse
* System Monitoring
* Testing
* Version Control System

### Verfügbare Werkzeuge sind für CD

28.02.2012 Untersucht werden:

* Jenkins Build Pipeline Plugin (Deployment-Workflow)
* ThoughtWorks Go
* Etsy Deployinator (Deployment-Pipe)
* Opscode Chef (Infrastruktur)
* Racker Dreadnot (Deployment-Pipe / Workflow)
* UrbanCode DevOps Platform?

### Vergleich der Ansätze der einzelnen Werkzeuge

28.02.2012 Evaluationskriterien:

* Funktionalitäten für Continuous Delivery
  + Building
    - Build-Prozess wird durch Commit in ein Repository angestoßen
  + Deploying
    - Qualitätsabhängiges Deployment
    - Deployment von Datenbanken
    - Deployment in die Produktivumgebung
  + Testing
    - Codeanalyse
    - Funktionale Tests
  + Reporting
    - Trendanalyse
  + Automatisierter Aufbau und Konfiguration von Systemumgebungen von Staging- und Produktivumgebungen
    - Umgebungskonfiguration
    - Systeminstallation
  + Workflowunterstützung
    - Automatisierter Anschub von Bild-Prozessen, Test und Deployment nach Qualitätskriterien
  + Unterstützung von Versionsverwaltungen
    - Zentrale Versionsverwaltung für Quellcode und Konfigurationsdatein
    - Repository für Reports, Binärdatein, Metadaten
  + Reporting
    - Qualitätsaussage zu einzelnen Builds
    - Darstellung von Status der Delivery Pipeline

28.02.2012: Anmerkungen zu den einzelnen Werkzeugen

Jenkins Build-Pipeline-Plugin:

* Aufbau einer Build Pipeline
* Qualitätsschranken für Builds
* Verkettung von einzelnen Jobs (upstream/downstream-Dependencies???)
* Deployment in Produktivumgebung
* Status der Pipeline
* Status jeder Version innerhalb der Kette anhand der Revisionsnummer aus dem Versionskontrollsystem

29.02.2012: Qualitätsmerkmale für Software aus (Gernot Starke, Effektive Softwarearchitektur)

* -> verweis auf Wallmüller: Software-Qualitätsmanagement in der Praxis 2001 (Zuverlässigkeit, Effizienz, Wartbarkeit, Portabilität)
* -> DIN 9126-15 (<http://de.wikipedia.org/wiki/ISO/IEC_9126>) definiert:
  + Funktionalität
    - Angemessenheit
    - Richtigkeit
    - Interoperabilität
    - Ordnungsmäßigkeit
    - Sicherheit
  + Zuverlässigkeit
    - Reife
    - Fehlertoleranz
    - Wiederherstellbarkeit
  + Benutzbarkeit
    - Verständlichkeit
    - Erlernbarkeit
    - Bedienbarkeit
  + Effizienz
    - Zeitverhalten
    - Verbrauchsverhalten
  + Änderbarkeit
    - Analysierbarkeit
    - Modifizierbarkeit
    - Stabilität
    - Prüfbarkeit
  + Übertragbarkeit
    - Anpassbarkeit
    - Installierbarkeit
    - Konformität
    - Austauschbarkeit
* Szenarien:
  + Anwendungsszenarien
  + Änderungsszenarien
  + Stress- und Grenzszenarien

01.03.2012: Beurteilungskriterien für die Beurteilung der Werkzeuge (nach Starke)

Zielstellung: Evaluierung von Werkzeugen für eine Deployment-Pipeline:

1. Funktionsumfang
   1. Schnittstelle Versionskontrolle
   2. Schnittstelle Artefakt-Repository
   3. Commit Stage
      1. Kompilieren des Quellcodes
      2. Commit Tests
      3. Assemble
      4. Code Analyse
   4. Acceptance Stage
      1. Konfigurieren der Umgebung
      2. Deployen der Binärdatein
      3. Smoke Test
      4. Akzeptanztest
   5. Self-Service Deployment für Tester
      1. Konfigurieren der Umgebung
      2. Deployen der Binärdatein
      3. Smoke Tests
   6. Kapazitätstest
      1. Konfigurieren der Umgebung
      2. Deployen der Binärdatein
      3. Smoke Test
      4. Kapazitätstest
   7. Produktionsumgebung
      1. Konfigurieren der Umgebung
      2. Deployen der Binärdatein
      3. Smoke Test
   8. Report
      1. Auswertung und Beurteilung der Builds anhand der Testergebnisse, Build-Laufzeiten
2. Funktionalitäten
   1. Liefern die Funktionen die erwarteten Ergebnisse?
   2. Sind die Funktionen für den Aufbau einer Deployment-Pipeline geeignet?
   3. Kann das System mit der bestehenden Infrastruktur zusammenarbeiten?
   4. Wird der Zugriff unberechtigter verhindert?
3. Zuverlässigkeit
   1. Wie verhält sich das System bei Fehlerzuständen?
4. Benutzbarkeit
   1. Ist der Aufwand für Benutzer, das Konzept des Systems zu verstehen gering?
   2. Ist der Aufwand die Bedienung des Systems zu erlernen gering?
5. Effizienz
   1. Sind die Verarbeitungszeiten bei der Funktionsausführung gering?
   2. Welche Betriebsressourcen werden zur Erfüllung der Funktionen benötigt?
6. Änderbarkeit
   1. Wie hoch ist der Aufwand das Versagen des Systems bei Fehlerzuständen zu diagnostizieren?
   2. Wie hoch ist der Aufwand das System zur Fehlerbeseitigung zu verbessern?
7. Übertragbarkeit
   1. Wie hoch ist der Aufwand das System an verschiedene Umgebungen anzupassen?
   2. Wie hoch ist der Aufwand das System zu installieren?

01.03.2012: Bewertungskriterien für Continuous Delivery nach Paul M. Duvall (DZone Refcardz Continuous Delivery)

Generelle Beurteilung nach Visualisierung, Feedback und kontinuierlichen Ausliefern:

* Visibility: Alle Aspekte von bauen, ausliefern, testen der Software und diese in die Produktivumgebung bringen kann, sind allen Teammitgliedern visuell zugänglich.
* Feedback: Alle Teammitglieder können schnell auf Probleme und Fehler reagieren und aus diesen lernen und diese schnell beseitigen.
* Continually Deploy: Ein automatisierter Prozess, der in jede Version einer Software in jede Umgebung ausliefern kann.

02.03.2012: Beurteilung nach Pattern für Continuous Delivery:

* Konfigurationsmanagement
  + Konfigurierbare Fremdanwendungen: Fremdanwendungen können leicht konfiguriert, deployt und automatisiert werden?
  + Konfigurierbarer Katalog: Existiert ein Katalog mit allen Einstellungen, wie dieses Einstellungen und Speicherorte geändert werden können? Dieser Katalog ist Teil des Build-Prozesses?
  + Hauptstamm: Gibt es nur ein minimales Merging und wird hauptsächlich auf dem Hauptstamm entwickelt?
  + Tägliches Merge: Werden Änderungen am Hauptstamm werden auch an die Branches weitergegeben?
  + Geschützte Konfiguration: Werden Konfigurationen an einem gesicherten und über einen Fernzugriff erreichbaren Ort wie eine Datenbank, Verzeichnis oder Regestry-Datei gehalten?
  + Repository: Werden alle Dateien, ausführbarer Quellcode, Konfigurationen, Gastumgebungen und Daten in einer Versionsverwaltung gehalten?
  + Kurzlebige Branches: Werden Branches nur wenige Tagen oder weniger als eine Iteration zugelassen?
  + Ein-Befehl-Umgebung: Gibt es einen einzigen Befehl um ein ausgeschektes Projekt zu bauen und zu deployen?
  + Nur einen Weg in die Produktion: Unterliegt das gesamte System dem Konfigurationsmanagement? Kann jede Änderung am System auf einen bestimmten Punkt über die Versionsverwaltung zurückgestellt werden?
* Continuous Integration
  + Schranke für Build: Wird das bauen schnell beendet, wenn eine Qualitätsgrenze wir Architekturelle?
  + Häufiges Commit: Wird mindestens einmal am Tag durch jedes Teammitglied commited?
  + Kontinuierliches Feedback: Gibt es ein automatisiertes Feedback an alle Team-Mitglieder?
  + Continuous Integration: Wird jede Änderung in der Versionsverwaltung gebaut und getestet?
  + Stopp der Produktion: Werden Fehler im Lieferprozess schnell behoben? Werden keine Änderungen in die Versionsverwaltung gegeben, wenn es zu einem solchen Fehler gekommen ist?
  + Unabhängiges Build: Sind Build-Skripte von der DIE losgelöst? Werden die Build-Skripte durch ein CI-System ausgeführt?
  + Dashboard: Existiert eine übersichtliche und aggregierte Aufbereitung über Build-Zustände? Ermöglicht diese Übersicht ein qualitatives Feedback?
* Testing
  + Automatisierte Test: Werden automatisierte Test bezüglich der Verifikation und Validierung
  + Isolierte Testdaten: Werden für Testzweck Datenbank-Transaktionen genutzt und diese anschließen wieder zurückgerollt?
  + Parallele Tests: Werden verschiedene Teste parallel auf unterschiedlicher Hardware ausgeführt?
  + Stumpftests: Werden Abhängigkeiten zu externe Systemen simuliert?
* Deployment Pipeline
  + Deployment Pipeline: Existiert eine automatisierte Pipeline für build, deploy, test und release?
  + Value-Stream Map: Existiert eine Karte die den Strom von der Versionsverwaltung
* Build und Deployment Scripting
  + Dependency Management: Sind alle Abhängigkeiten zentralisiert um Fehler in den Klassenpfaden sowie Wiederholungen zu vermeiden?
  + Einheitliche Sprache: Existiert eine einheitliche Skriptsprache, um allen Teammitgliedern Änderungen am Liefersystem zu ermöglichen?
  + Externalisierte Konfiguration: Sind alle variablen Werte von Build und Deployment Properties externalisiert und werden Änderungen aufgezeichnet.
  + Fail Build: Sind die Abläufe so gestaltet, das Prozesse, die häufig scheitern zu erst ausgeführt werden können?
  + Fast Build: Gibt es ein schnelles Feedback, sofern ein Build scheitert?
  + Scripted Deployment: Basieren alle Deploymentprozesse auf Scripten und unterliegen einer Versionsverwaltung?
  + Unified Deployment: Wird für jedes Deployment dasselbe Skript verwendet? Unterliegt die Konfiguration einer Versionsverwaltung?
* Auslieferung und Verteilung der Anwendung
* Infrastruktur und Umgebung
* Daten
* Inkrementelle Softwareentwicklung
* Zusammenarbeit

05.03.2012: Installation VM

06.03.2012: Konfiguration der VM

### Systeme die bei adesso derzeit für Continuous Integration und Deployment eingesetzt werden

* Jenkins CI Server
* Chef (opscode)

### Werkzeuge die sich in die bestehende Systemlandschaft von adesso integrieren lassen

## Umsetzung einer Deployment Pipeline

### Schematische Darstellung einer Deployment Pipeline

23.02.2012: Ablauf

1. Build: Quellcode wird aus der Versionsverwaltung geholt. Anschließend wird der Quellcode kompiliert, getestet und eine Codeanalyse durchgeführt. Die Ergebnisse werden mit Berichten und Metadaten in ein Repository übertragen.
2. Akzeptanz-Tests: Für den Akzeptanztest werden Konfigurationsdateien aus der Versionsverwaltung geladen und die Testumgebung damit konfiguriert. Die Binärdateien werden aus dem Repository geladen und das System wird installiert. Der Anwendung werden Smoke- und Akzeptanztests unterzogen. Berichte und Metainformationen des Akzeptanztests werden in das Repository übertragen.
3. Abnahme und Lasttest / Produktivumgebung: Manuelles Anstoßen von Abnahme- und Lasttest. Grade bei Lasttests wird mit einer längeren Laufzeit zu rechnen sein, dieses können sich mehrere Stunden hinziehen. Sind Abnahme- und Lasttests erfolgreich absolviert, kann die Anwendung durch anstoßen eines Deployment-Skriptes auf die Produktivumgebung gebracht werden.

### Elemente der Delpolyment Pipeline

### Werkzeuge und Konfiguration

## Master-Thesis

### Theoretische Grundlagen

### Abschnitte schreiben

# Fragen

27.02.2012: Welche Randbedingungen sind bei einem Deployment zu beachten? (DB, Konfiguration Load Balancer bei Web-Anwendungen, etc.)

27.02.2012: Welche Tools eignen sich für Akzeptanztests?

Selenium-Test und Sonar Code-Review sind bei adesso bekannt.

27.02.2012: Was kann das Jenkins-Build-Pipeline-PlugIn?[[1]](#footnote-1)

02.03.2012: Wie kann das ITIL Change-Management eine CD unterstützen? Welche Abhängigkeiten zu ITIL Change-Management sollte eine Deployment-Pipeline berücksichtigen?

ITIL-Release-Management

„Das Release-Management ist der Prozess, der für die Planung des zeitlichen Ablaufs und die Steuerung des Übergangs von Releases in Test- und Live-Umgebungen verantwortlich ist. Das wichtigste Ziel des Release-Managements ist es sicherzustellen, dass die Integrität der Live-Umgebung aufrechterhalten wird und die richtigen Komponenten im Release enthalten sind. Das Release-Management ist Teil des *Release-und-Deployment-Management*-Prozesses.“ (Wikipedia: Change Management (ITIL))

# Literatur

Interessante Literatur zum Thema aus der Adesso-Bibliothek

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Tilly, Jesse | Ant: The Definitive Guide | O'Reilly | Dortmund | 0596001843 |
| Duvall, Matyas & Glover | Continuous Integration: Improving Software Quality and Reducing Risk |  | Dortmund | 978-0321336385 |
| Kunnumpurath Meeraj M. | JBoss 3.2 Deployment and Administration | Apress | Berlin | 1590592816 |
| Popp | Konfigurationsmanagement mit Subversion, Ant und Maven |  | Berlin | 978-3898644167 |
| Fewster & Graham | Software Test Automation | Addison Wesley | Dortmund | 0201331403 |
| Nguyen Hung Q. | Testing Applications on the Web | Wiley | Dortmund | 047139470x |
| Thaller, Georg E. | Software-Test Verifikation und Validation | Heise | Dortmund | 3882291834 |
| Biskup, Thomas | Spring Praxishandbuch: Integration und Testing (Broschiert) |  | Dortmund | 978-3939084396 |
| Rätzmann | Software Testing | Galileo Press | Köln | 3898422712 |
| Link, Johannes | Unit Tests mit Java | dpunkt Verlag | München | 3898641503 |

Interessante Literatur zum Thema im Handel

|  |  |  |
| --- | --- | --- |
| Jez Humble (Autor), David Farley (Autor) | Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation (Addison-Wesley Signature) | <http://www.amazon.de/Continuous-Delivery-Deployment-Automation-Addison-Wesley/dp/0321601912/ref=sr_1_2?ie=UTF8&qid=1330346006&sr=8-2> |
| Paul M. Duvall (Autor), Steve Matyas (Autor), Andrew Glover (Autor) | Continuous Integration: Improving Software Quality and Reducing Risk (Martin Fowler Signature Books) | <http://www.amazon.de/Continuous-Integration-Improving-Software-Signature/dp/0321336380/ref=sr_1_1?ie=UTF8&qid=1330346158&sr=8-1> |
| Simon Wiest (Autor) | Continuous Integration mit Hudson/Jenkins: Grundlagen und Praxiswissen für Einsteiger und Umsteiger [Broschiert] | <http://www.amazon.de/Continuous-Integration-mit-Hudson-Jenkins/dp/3898646904/ref=sr_1_3?ie=UTF8&qid=1330346158&sr=8-3> |
| Gunther Popp (Autor) | Konfigurationsmanagement mit Subversion, Maven und Redmine: Grundlagen für Softwarearchitekten und Entwickler | <http://www.amazon.de/Konfigurationsmanagement-Subversion-Maven-Redmine-Softwarearchitekten/dp/3898645215/ref=sr_1_11?ie=UTF8&qid=1330346158&sr=8-11> |

Interessante Literatur zum Thema in der Hochschulbibliothek

???

1. <https://wiki.jenkins-ci.org/display/JENKINS/Build+Pipeline+Plugin> [↑](#footnote-ref-1)