Continuous Delivery 2016 – Kontinuierlich an die Pipeline andocken

Orientation in Objects GmbH

Weinheimer Str. 68
68309 Mannheim

www.oio.de
info@oio.de

Version: 1.1

Gliederung

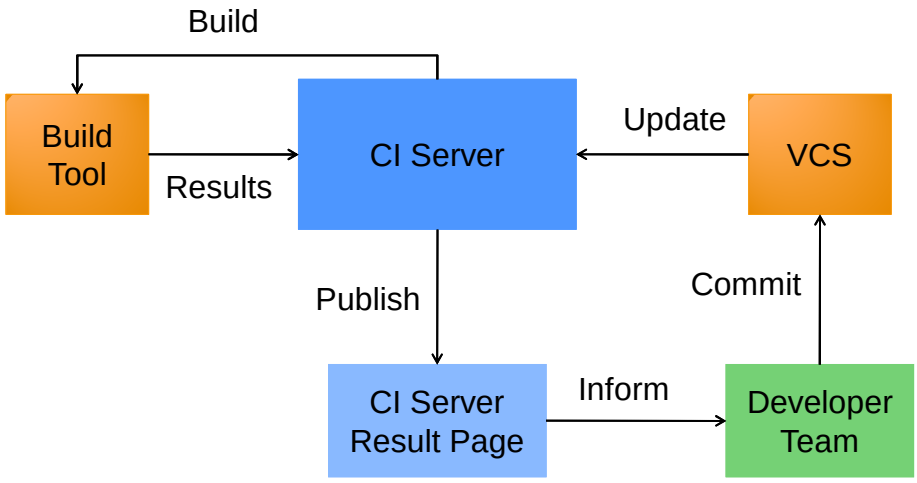
- Wer? Wie? Was?
- Match Made In Heaven
- Zusammenfassung

Gliederung

- **Wer? Wie? Was?**
- Match Made In Heaven
- Zusammenfassung

- Continuous Delivery

Been there, done that (1)



Been there, done that (2)

Orientation in Objects

- „Daily Build and Smoke Tests“ sind schon ein alter Hut
  - Erste Veröffentlichung von Steve McConnell im Jahre 1996
- „Continuous Integration“ Artikel von Martin Fowler im Jahre 2000
  - Klingender Name, „Key Practices“, erste Tools (CruiseControl)
- „Continuous Integration“ gehört heute zum guten Ton
  - Diverse Server, Definition von Best Practices und Patterns

Da war doch noch was? (1)

Orientation in Objects

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

First principle behind the Agile Manifesto
(http://agilemanifesto.org/principles.html)

Da war doch noch was? (2)

Orientation in Objects

Our highest priority is to ***satisfy the customer*** *through early and continuous delivery of valuable software.*

First principle behind the Agile Manifesto
(http://agilemanifesto.org/principles.html)

I Can't Get No Satisfaction

Orientation in Objects

- Kunde möchte Verfügbarkeit seiner Funktionalität
  - Kein Interesse, ob CI Server rot, grün, gelb oder blau ist.
- Zwischen gutem CI Build und Kunden Verfügbarkeit liegt Release
  - Oft manuell, zeitintensiv, kompliziert, viele Beteiligte, fehleranfällig
- Seltene Releases als Konsequenz
  - Frustrierend für den Kunden, dafür Stress bei Hotfix Releases

Buzzword Time (1)

Orientation in Objects

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

First principle behind the Agile Manifesto
(http://agilemanifesto.org/principles.html)

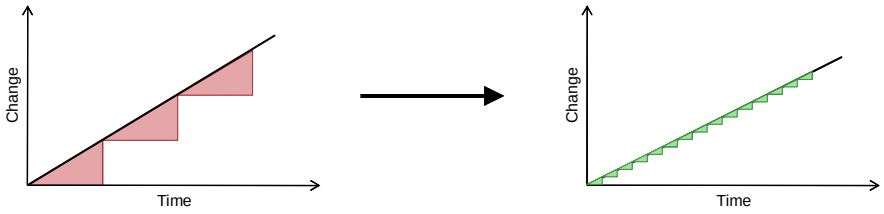
Buzzword Time (2)

Orientation in Objects

Our highest priority is to satisfy the customer through early and ***continuous delivery*** *of valuable software.*

First principle behind the Agile Manifesto
(http://agilemanifesto.org/principles.html)

Let Me Google That For You

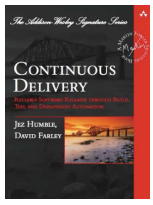


"[…] reduce the cost, time and risk of delivering changes, and ultimately value, to customers by allowing for more incremental changes to applications in production"

(Quelle: http://en.wikipedia.org/wiki/Continuous_delivery)

The Good Book

- Frühere Begriffsverwendung von Continous Delivery und Wurzeln
  - „Agile Manifesto" (2001)
  - „Deployment Pipeline" (2004 / 2005)
- Gleichnamiges Buch von Jez Humble & David Farley
  - Eigentliche Begriffsprägung (2010)
- Schwerpunktthemen „Automation" und „Collaboration"



Continuous Delivery Konzepte

Orientation in Objects

- “Create a Repeatable, Reliable Process for Releasing Software”
  - “If It Hurts, Do It More Frequently, and Bring the Pain Forward”
  - “Automate Almost Everything”
  - “Keep Everything in Version Control”
- “Everybody Is Responsible for the Delivery Process”
- “Done Means Released”
- Und wie soll das alles umgesetzt werden?

(Nach „Continuous Delivery“/J. Humble, D. Farley)

Gliederung

Orientation in Objects

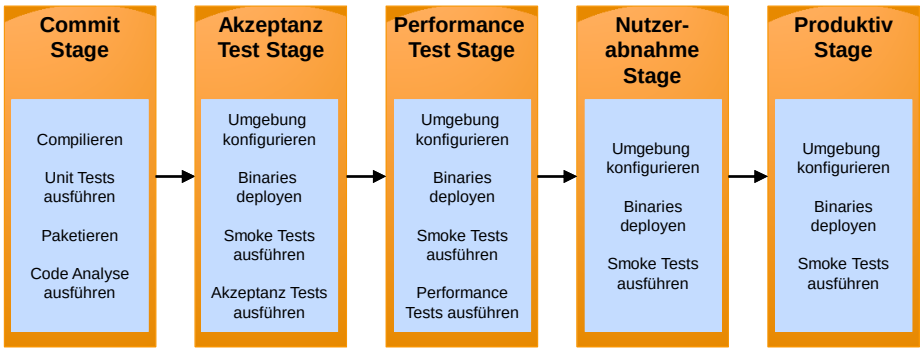
- **Wer? Wie? Was?**
- Match Made In Heaven
- Zusammenfassung

- Continuous Delivery
- **Deployment Pipeline**

Deployment Pipeline – Ein erster Blick

Orientation in Objects

- Zentrale Abstraktion „Deployment Pipeline“
  - Visualisierung aller Prozessteile für alle Beteiligten
  - Verbessertes Feedback während der Ausführung
  - Möglichkeit eines vollautomatischen Releases in alle Umgebungen



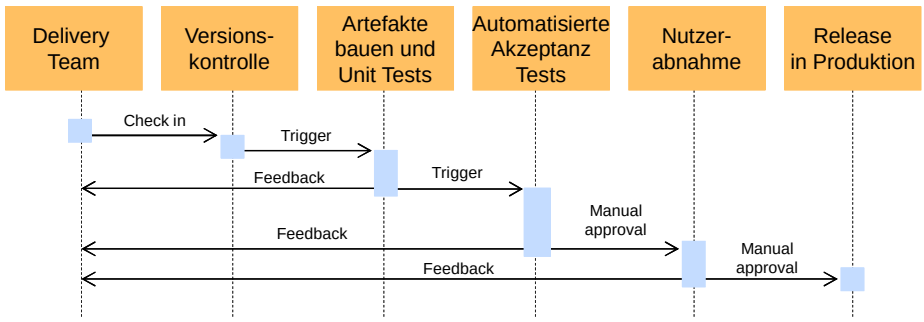
(Nach „Continuous Delivery“/J. Humble, D. Farley)

Deployment Pipeline – Bestandteile

Orientation in Objects

- Die **Deployment Pipeline**
  - Macht Status der Produktentwicklung sichtbar
  - Liefert Feedback zu jeder Änderung
  - Technisch-konzeptuelle Basis des Release Prozesses
- Die Pipeline besteht aus einer Folge von **Stages**
  - Commit Stage als zentrales Eingangs-Gate
  - Typische Stages: UAT, Performance Tests, Production Deployment
  - Stages verbunden durch Trigger (automatisch oder manuell)
- **Jobs** sind die Bausteine der Stages
  - „Unit of Work“
  - Bestehen aus Tasks wie Build, Deploy, Copy, Test, …

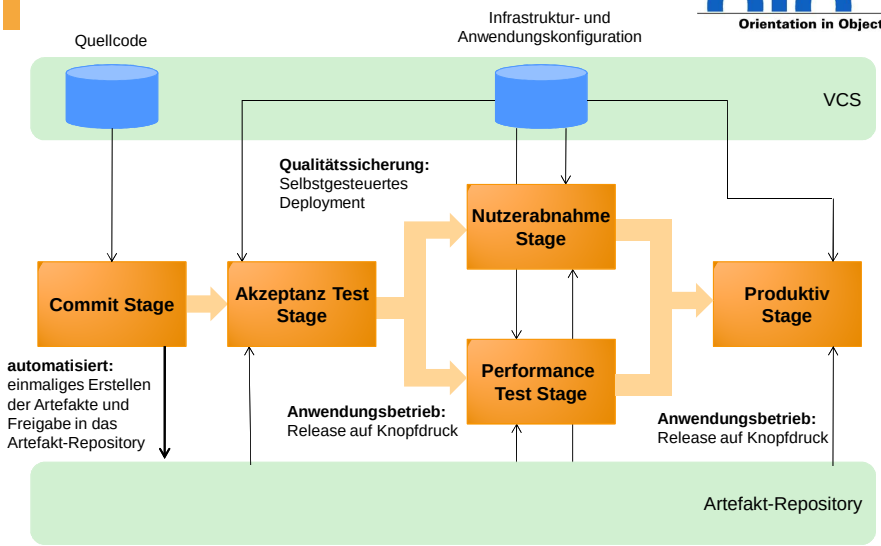
Deployment Pipeline – Sequenzdiagramm



- Jede Ressourcen Änderung startet neue Pipeline Instanz
- Erste Stage produziert alle Artefakte
- Durchlaufen aller Stages bis Fehlschlag („Stop the line“) oder …
- Pipeline Ende erreicht ist (letzte Stage führt Deployment aus)

(Nach „Continuous Delivery“/J. Humble, D. Farley)

Deployment Pipeline – Ein zweiter Blick



(Nach „Continuous Delivery“/J. Humble, D. Farley)

tl;dr

Orientation in Objects

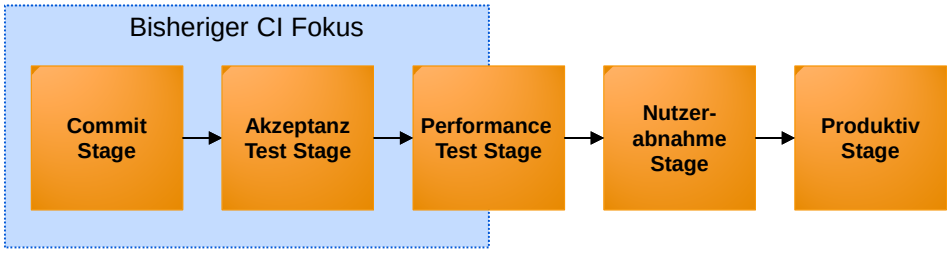
“A deployment pipeline is, in essence, an automated implementation of your application’s build, deploy, test, and release process.”

(“Continuous Delivery”, J. Humble, D. Farley)

Alle Theorie ist grau

Orientation in Objects

- *“The deployment pipeline has its foundation in the process of continuous integration and is in essence the principles of continuous integration taken to its logical conclusion.” (J. Humble, D. Farley)*



Tool Time

Orientation in Objects

- Jeder gängige CI Server bietet Pipeline Bausteine an
  - Und ist somit ein CD Server
- Konkrete Namen können variieren
  - Teils historische Gründe, teils Abgrenzung zur Konkurrenz
- Exemplarische Beispiele („*Your Mileage May Vary*“)
  - **Jenkins**: *Build Jobs, Build Steps, Post-build Actions, diverse Plugins*
  - **Bamboo**: *Build Plans, Stages, Jobs, Tasks*
- CI Server Unterschiede eher in anderen Bereichen, zum Beispiel
  - **Jenkins Pipeline Plugin**: *Erstellung von Pipelines per DSL*
  - **Bamboo Deployment Projects**: *Verwaltung von Artefakt Deployments*

Gliederung

Orientation in Objects

- **Wer? Wie? Was?**
- Match Made In Heaven
- Zusammenfassung

- Continuous Delivery
- Deployment Pipeline
- **Docker**

Was ist Docker?

Orientation in Objects

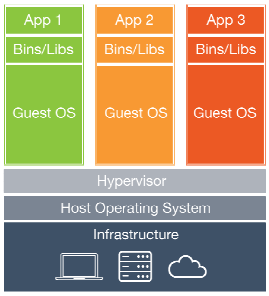
- *“Docker allows you to package an application with all of its dependencies into a standardized unit for software development.”*
- Open Source Projekt mit über 1.000 Mitwirkenden
  - Release als Open Source Projekt im März 2013
  - Mitwirkende Unternehmen sind u.a. Red Hat, IBM, Google, Cisco
- Kernabstraktion ist Container („*standardized unit*“)
  - Virtualisierung auf Betriebssystem Ebene („lightweight“)
  - Simuliert vollständig isolierte Umgebung („secure“)
  - Basiert auf verschiedenen Linux Kernel Features („open“)

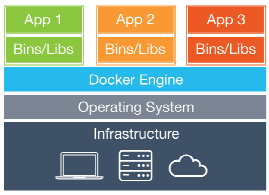
docker

Container vs. Virtual Machine (VM)

Orientation in Objects

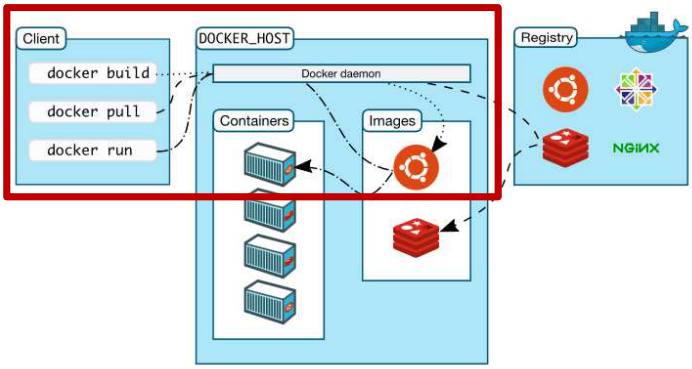
- Container bieten isolierte Umgebung auf dem selben OS
  - Container im Vergleich zu VMs kleiner und weniger ressourcenhungrig
- Virtual Machines benötigen jeweils komplettes OS
  - Virtual Machine, z.B. nötig um Docker auf Windows laufen zu lassen





(Quelle Graphik: https://www.docker.com)

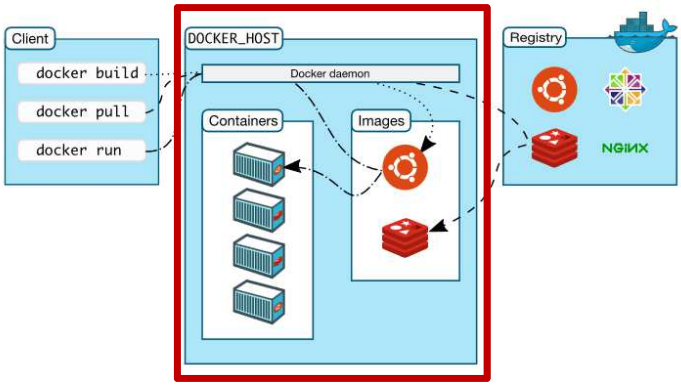
Docker Architektur – Daemon und Client



- Auf einer Docker Host Maschine läuft der Docker Daemon
- Per Docker Kommandozeilen Client kann Benutzer interagieren

(Quelle Graphik: https://docs.docker.com)

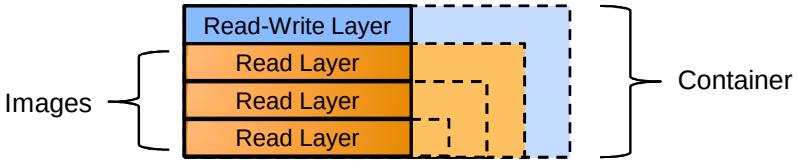
Docker Architektur – Container und Images



- Container enthalten alles um Anwendungen isoliert auszuführen
  - Können u.a. gestartet, gestoppt und gelöscht werden („*run component*“)
- Images sind „Read Only“ Templates für Container
  - z.B. mit Ubuntu, Apache HTTPD, Webanwendung („*build component*“)

(Quelle Graphik: https://docs.docker.com)

Docker Architektur – Image Layers



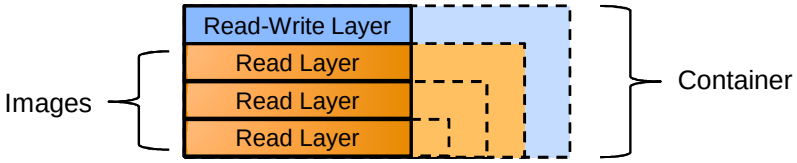
- Images bestehen aus hierarchischen „read only“ Schichten (Layers)
  - Jedes Image basiert auf Basis Image (z.B. Ubuntu oder Apache Image)
  - Image verändern heißt einzelne Schichten erneuern oder ergänzen
- Bestehendes Image kann mittels „Instructions“ verändert werden
  - Run Command, Add File, Launch Process, Create Env Variable, …
- Basis Image Angabe plus Instructions bilden ein „Dockerfile“

Docker Architektur – Dockerfile

```
# ANGABE DES BASIS IMAGE
FROM java:openjdk-8u66-jdk

MAINTAINER Steffen Schluff <steffen.schluff@oio.de>

RUN apt-get update && apt-get install -y wget git && rm -rf /var/lib/apt/lists/*

ADD https://get.docker.com/builds/Linux/x86_64/docker-1.8.3 /usr/local/bin/docker

RUN chmod +x /usr/local/bin/docker

COPY start.sh /usr/local/bin/start.sh

ENV BAMBOO_HOME /data/bamboo-agent-home

CMD ["/usr/local/bin/start.sh"]
```

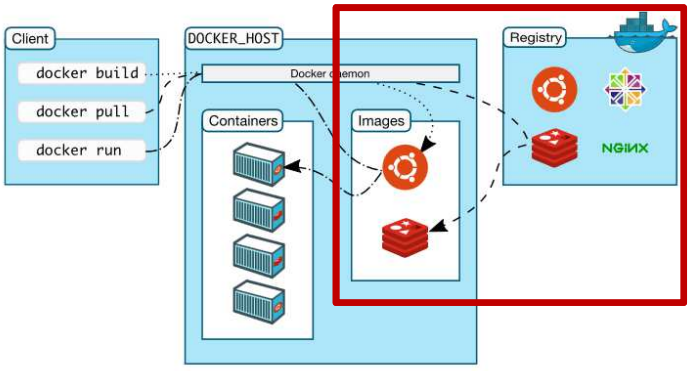
- Mittels „*docker build*“ kann aus Dockerfile ein Image erstellt werden
  - Angabe eines Context (Bezugsverzeichnis) ebenfalls notwendig

Docker Architektur – Container



- Mittels „*docker run*“ wird aus Image ein Container erzeugt
  - Image enthält Infos zu Container Inhalt und was ausführen ist
- Container „enthält“ Image Layers und eine obere „read write“ Layer
  - In diesem Container können nun Anwendungen ablaufen
- Container können sich in verschiedenen Zuständen befinden
  - Container müssen nicht laufen, sondern können auch gestoppt sein

Docker Architektur – Registries



- Registries verwalten Images und stellen diese in Repositories bereit
  - Dies können eigene oder fremde Images sein („distribution component“)
- „Docker Hub“ ist die offizielle Docker Registry
  - Enthält private, öffentliche und offizielle Repositories

(Quelle Graphik: https://docs.docker.com)

Docker Client Commands (Auszug)

Orientation in Objects

- *build* – Build an image from a Dockerfile
- *images* – List images
- *ps* – List containers
- *pull* – Pull an image or a repository from a registry
- *rm* – Remove one or more containers
- *rmi* – Remove one or more images
- *run* – Run a command in a new container
- *stop* – Stop a running container

Sittin' On The Dock Of The Bay

Orientation in Objects

- „Ich möchte keinen Monolithen sondern meine Dienste trennen.“
  - Datenbank und Web Server sollen in getrennten Containern laufen
  - Docker hat ein „*Linking System*“ zur Verknüpfung von Containern
- „Ich möchte aber meine Daten nicht in einem Container haben.“
  - Daten sollen von Containern weiter verwendet oder geteilt werden
  - Docker besitzt „*Data Volumes*“ zur Bewegdaten Verwaltung
- „Jetzt wird bestimmt die Konfiguration wieder so kompliziert.“
  - Setup und Verlinkung von Containern und Data Volumes soll leicht sein
  - „*Docker Compose*“ für zentrale Konfig und „single command launch“
- Außerdem Clustering mit Docker Swarm, eigene Docker Registry, …

Gliederung

Orientation in Objects

- Wer? Wie? Was?
- **Match Made In Heaven**
- Zusammenfassung

Mehr als die Summe seiner Teile

Orientation in Objects

- Container auf ersten Blick doch nur eine weitere Art der Paketierung
  - Einbindung in CI Server per Kommandozeile wie bei RPM, JAR, MSI, …
  - Docker ist aber mehr als nur ein weiteres Paket Format
- *"Docker is an open **platform** for building, shipping and running distributed **applications**."*
- Docker bietet komplettes Ökosystem nicht nur ein einzelnes Tool
  - Zum Beispiel „Official Repositories" auf Docker Hub
- Container als deploybares Artefakt enthalt vollständige Anwendung
  - Kein Mix mehr aus zu deployenden Binaries, externen Libs, Configs, …
- Was hat der Einsatz von Docker denn dann zur Folge?

Silver Bullets und Buzzwords (1)

Orientation in Objects

- Docker wird gerne mit bestimmten Themen „kombiniert“
  - Und jede erfolgreiche neue Technologie weckt Hoffnung auf „Wunder“
- Continuous Delivery / DevOps
  - Künftig deployen wir komplette Anwendungen als Container
  - Alle unsere Missverständnisse zwischen Dev und Ops sind gelöst

(Quelle: http://dev2ops.org/2010/02/what-is-devops/)

Silver Bullets und Buzzwords (2)

Orientation in Objects

- Immutable Infrastructure
  - Infrastruktur wird getrennt in Server und Daten
  - Server besitzen eine versionierte Definition (keine „in-place updates“)
  - Bei Änderungen wird alter Server entfernt und neuer gestartet
  - Docker kann das durch Dockerfiles und Data Volumes
  - Keine Snowflake Server mehr, Rollbacks und Scaling wird viel leichter
- Microservices
  - Docker bietet isolierte Container, die miteinander verlinkbar sind
  - Viele verbundene Dienste in verschiedenen Sprachen leicht erstellbar
- Docker kann bei diesen Themen helfen, löst aber nichts von alleine

Und was nun?

Orientation in Objects

- „Große“ Themen implizieren ein Container basiertes Deployment
- Docker basierter Produktionsbetrieb häufig nicht so leicht machbar
  - Strategische Entscheidung von großer Tragweite
  - Technologie ist noch relativ neu
  - Vorhandenen Applikationen müssen angepasst werden
- Ist Docker dann für „mich“ überhaupt spannend?
- *“Docker can help improve CI independently of whether your application will ultimately be deployed as a Docker image or not .“*

Das Runde muss ins Eckige (1)

Orientation in Objects

- Docker und CD Server können unterschiedlich zusammen arbeiten
- Bereitstellung von Build Agents oder Build Slaves
  - Buildverteilung ist mittlerweile de facto Standard
  - Mit Docker können schnell jeweils passende Agents erzeugt werden
  - CD Server Hersteller bieten teilweise fertige Agent Images an
- Docker Container als Teil des Build starten
  - Um Anwendung selbst für Tests zu starten (sofern in einem Container)
  - Um Infrastruktur für Integrationstest bereit zu stellen (Datenbanken, …)
  - Verwendete Images sind auch lokal von Entwicklern nutzbar

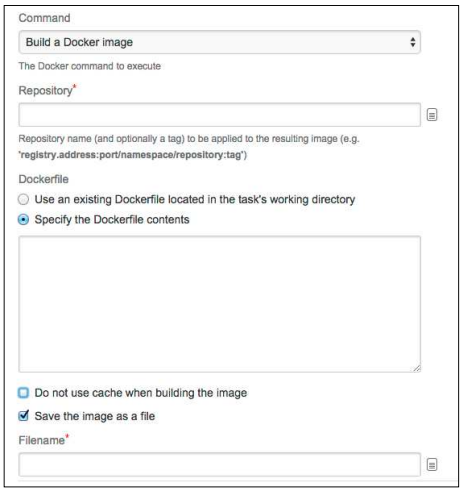
Das Runde muss ins Eckige (2)

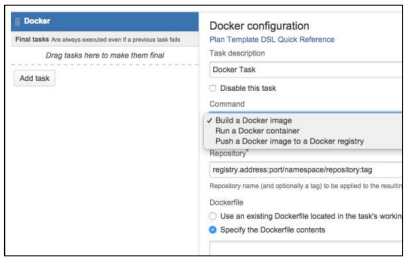
Orientation in Objects

- Docker Image bauen
  - Falls Anwendung als Docker Container ausgeführt werden soll
  - Inklusive Tagging (Build Nummer oder symbolischer Tag wie „latest“)
- Docker Image in Registry pushen
  - Falls Image anderen leicht zugänglich gemacht werden soll
  - Für eventuelles späteres Deployment der Anwendung
- Information aus Registry ob bestimmtes Image aktualisiert wurde
  - Image ist Quelldatei, falls Anwendung als Container in Produktion geht
  - Registries bieten Möglichkeiten bei Änderungen Webhooks aufzurufen
- All dies wird von den gängigen CD Servern direkt unterstützt

Demonstration

Orientation in Objects

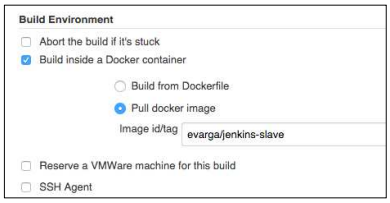




Your Mileage May Vary

Orientation in Objects

- Auch Jenkins kann mit Docker kombiniert werden
  - https://github.com/jenkinsci/docker-workflow-plugin/tree/master/demo




```
node('docker') {
 git 'https://git.mycorp.com/myproject.git'
 docker.image(,mycorp/ant:1.9.4').inside {
   sh 'ant dist-package'
 }
 archive 'app.zip'
}
```

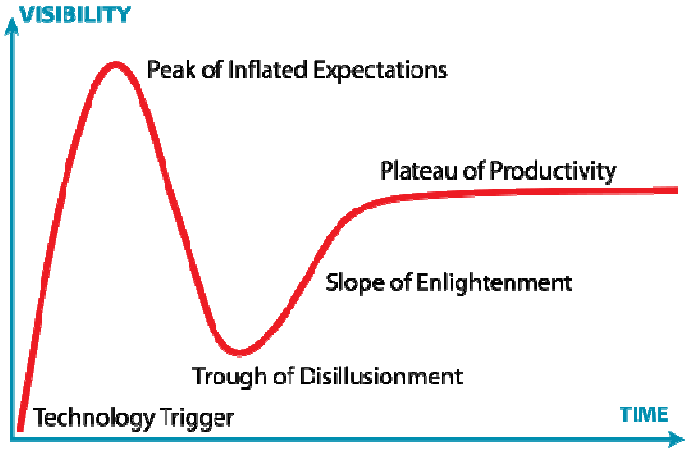

Gliederung

Orientation in Objects

- Wer? Wie? Was?
- Match Made In Heaven
- **Zusammenfassung**

Continous Delivery wird Fahrt aufnehmen
NEU – Jetzt mit Docker Beschleuniger

Orientation in Objects



(Quelle: http://en.wikipedia.org/wiki/Hype_cycle)

If you remember one thing

Orientation in Objects

"For anybody doing CI today, moving to Docker images represents low-hanging fruit that comes with very little disruption, but lots of advantages."

(Cloudbees Whitepaper: CD with Jenkins & Docker)

Links

Orientation in Objects

- Unterstand the Docker architecture
  - https://docs.docker.com/introduction/understanding-docker/
- Docker containers, Bamboo, and winning at continuous delivery
  - http://blogs.atlassian.com/2015/06/docker-containers-bamboo-winning-continuous-delivery/
- Configuring the Docker task in Bamboo
  - https://confluence.atlassian.com/bamboo/configuring-the-docker-task-in-bamboo-720411254.html
- Jenkins and Docker
  - https://www.cloudbees.com/continuous-delivery/jenkins-docker
- Docker Workflow Plugin Demo
  - https://github.com/jenkinsci/docker-workflow-plugin/tree/master/demo

Orientation in Objects

Fragen ?

Orientation in Objects GmbH

Weinheimer Str. 68
68309 Mannheim

www.oio.de
info@oio.de

Orientation in Objects

Vielen Dank für ihre Aufmerksamkeit !

Orientation in Objects GmbH

Weinheimer Str. 68
68309 Mannheim

www.oio.de
info@oio.de