

CI/CD

Continuous Integration

Continuous Delivery / Deployment

DevOps

Pipelines

CI

Continuous Integration

Continuous Integration

- Änderungen in kleinen Inkrementen einchecken
- Oft mergen
(Soll Merge-Hell verhindern)
- Bei jedem Eincheckvorgang Tests laufen lassen
(Unit- und kurze Integration-Tests)

CD

(Continuous Delivery)

Continuous Deployment

Continuous (Delivery) Deployment

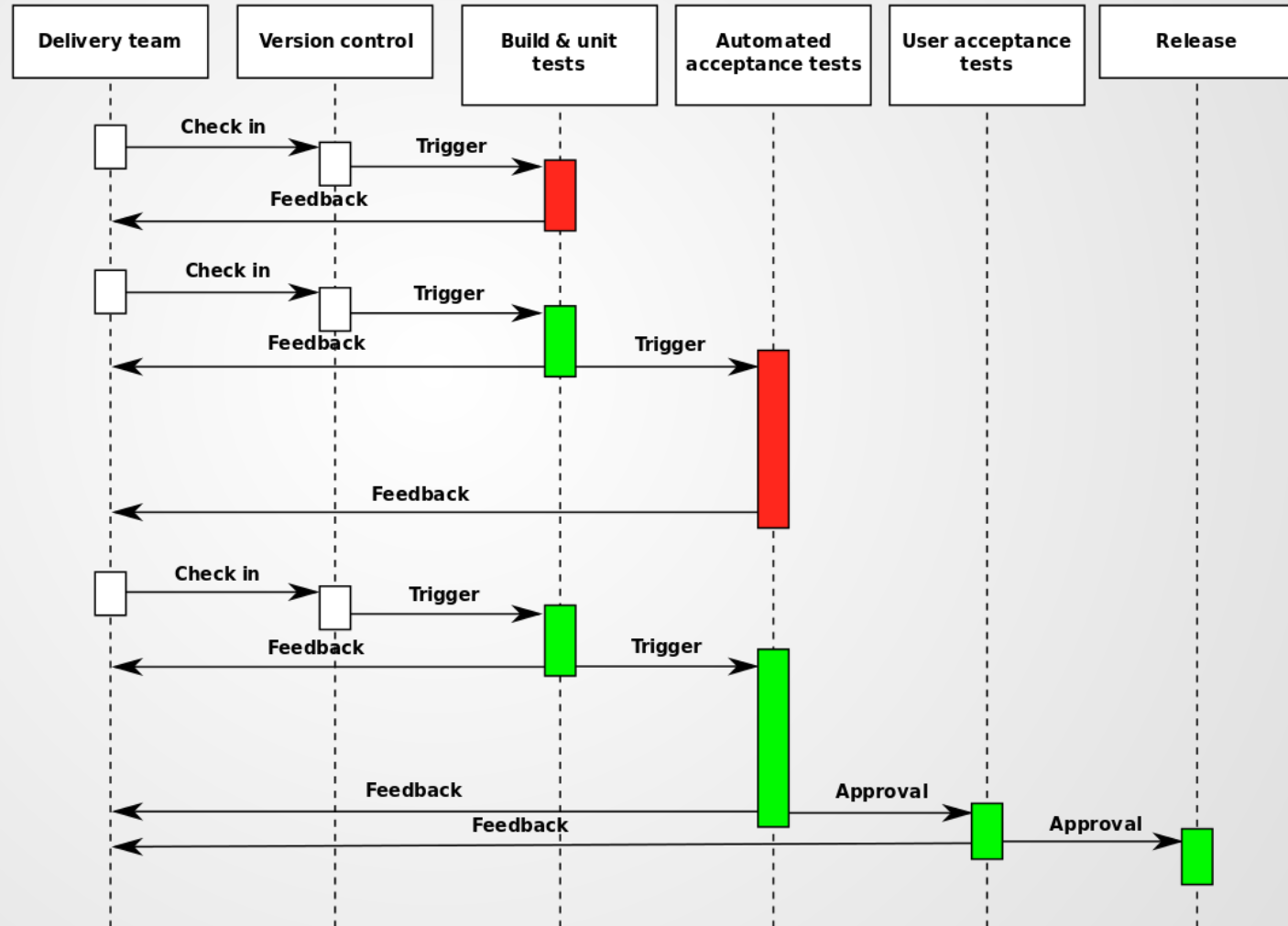
- Oft neue Versionen bauen
- Jede Version ist ein ernstzunehmender Release Candidate
- Automatisierte Tests laufen nach jedem Checkin
- Kann auch manuelle Tests beinhalten

Unterschied Delivery/Deployment

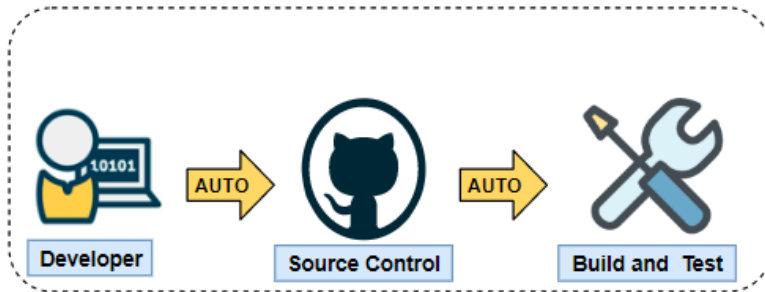
- "Continuous Delivery" hat **manuelles** Deployment
- "Continuous Deployment" hat **automatisches** Deployment

Continuous (Delivery) Deployment

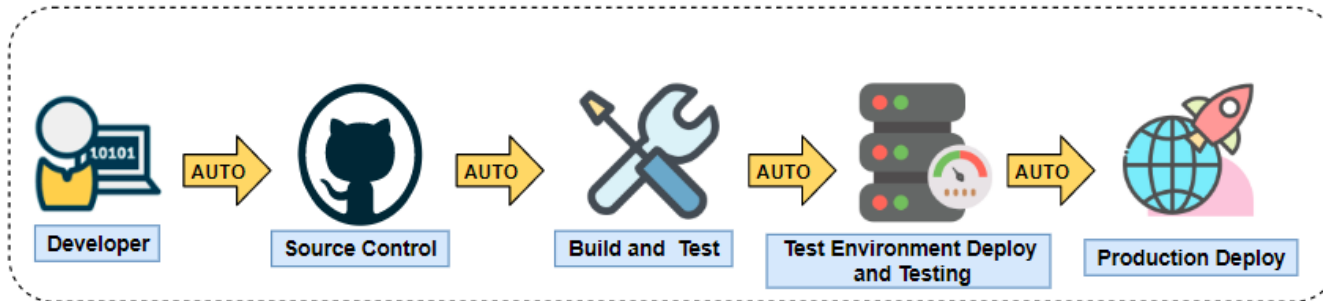
teilautomatische Pipeline...



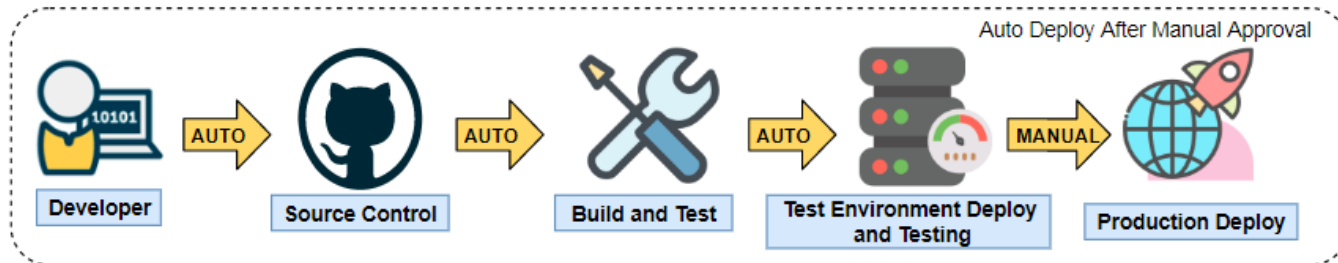
Continuous Integration



Continuous Deployment



Continuous Delivery



RELEASE

Continuous Integration

- Continuously integrating to a shared repository every day

BUILD

Continuous Delivery

- Always ready to ship

Continuous Deployment

- Continuously Deploying / Shipping

DevOps

DevOps

(Development / Operations)

- Kulturelle Änderungen innerhalb der Softwarefirma
- Automatische Builds
 - CI/CD sind notwendig
- Zusammenarbeit verschiedener Teams
 - Developers
 - Operations
 - Quality Assurance
 - Management
 - ...
- um ein Produkt **automatisiert** zu releasen **und zu warten**

DevOps

- Soll den Systems Development Lifecycle verkürzen (Betrieb und Wartung gehört da auch dazu)
- Soll Softwarequalität erhöhen
- Kommt von agiler Softwareentwicklung
- Schaut in kleineren Firmen so aus, dass die Developer alles machen :)

Prinzipien

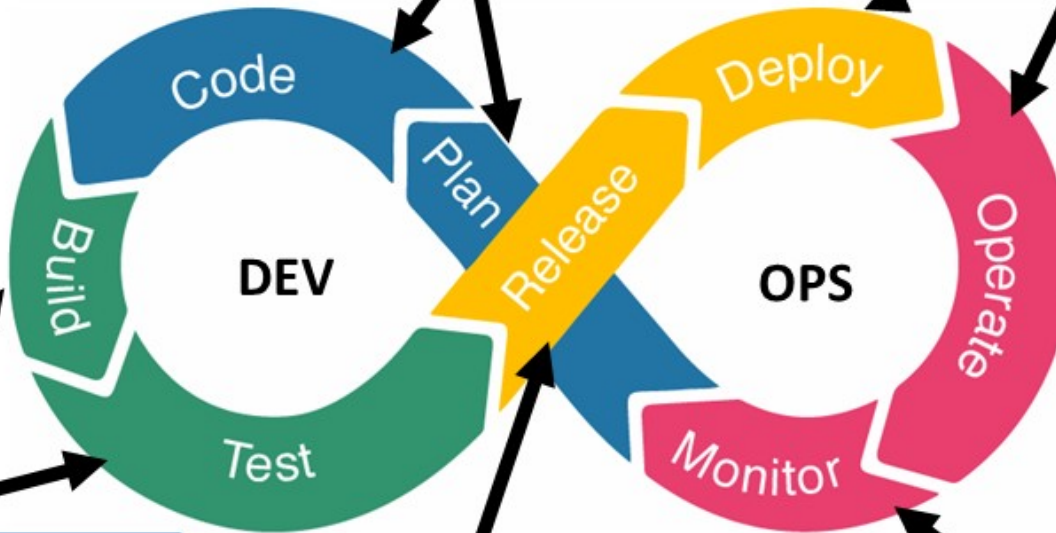
- Shared (Collective) Ownership
(gehört allen; alle sind verantwortlich)
- Workflow Automation
- Rapid Feedback

Continuous Development:

Jira, Confluence, Bitbucket, Git, Svn, VSTS, etc.

Continuous Deployment:

Puppet, Chef, Ansible, SaltStack, etc.



Continuous Testing:

Zephyr, TestComplete, SoapUI, Selenium, TestNG, JUnit/PHPUnit

Continuous Integration:

Jenkins, Bamboo, Hudson, etc.

Continuous Monitoring:

Splunk, ELK Stack, Nagios, Sensu, NewRelic

CI/CD

Pipeline

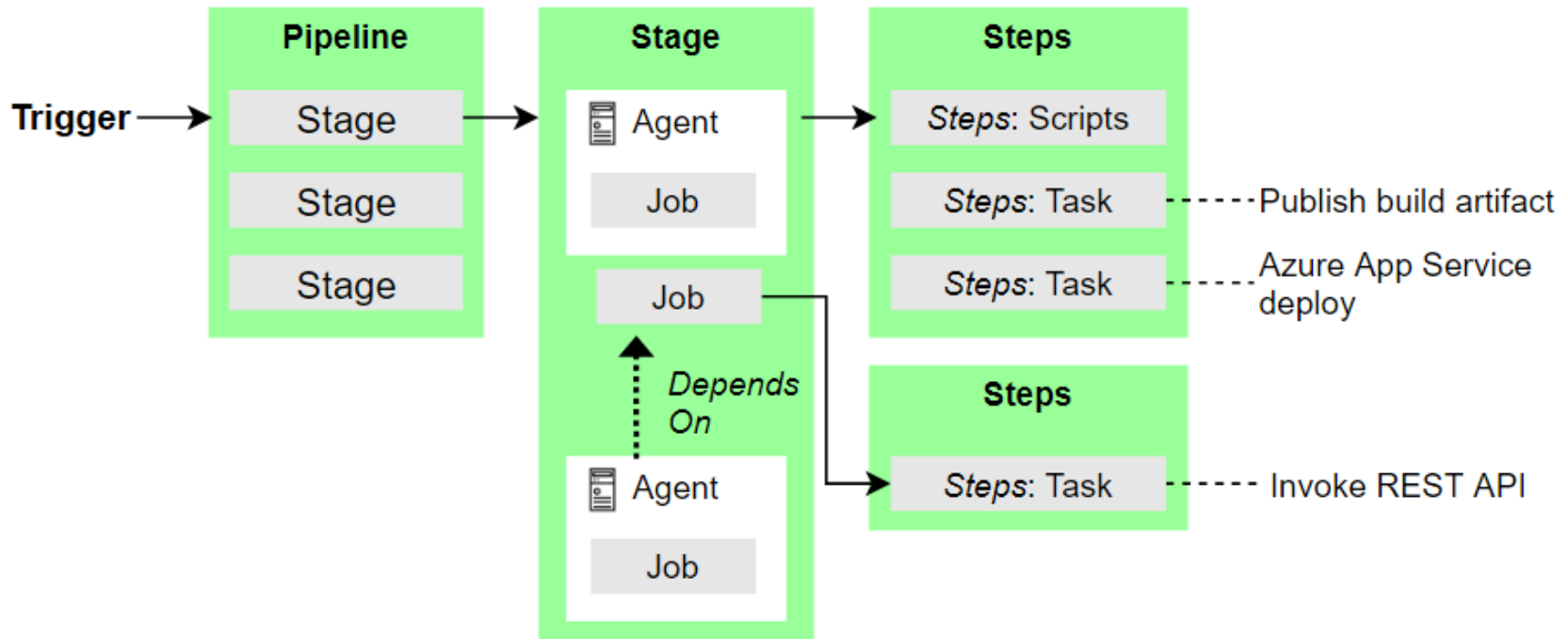
Build vs. CI/CD Pipeline

- Genau wie es für **Dependency Management** eigene Tools gibt, ...
 - Maven, Gradle, NPM, ...
- ... gibt es auch für **Build Pipelines** eigene Tools ...
 - MS-Build, Maven, ANT, NPM, ...
- Am Ende dieser Build Pipeline habt ihr ein kompiliertes Programm

CI/CD Pipeline

- In einer **CI/CD Pipeline** wird **gebaut** (mehrere Build Pipelines werden gestartet) und **deployed**
- Tools:
 - Gitlab (lokal oder in der Cloud)
 - Github Actions
 - Travis
 - ...
- Checkin triggert eine Pipeline
 - Baut die Software
 - Testet die Software
 - Baut verschiedene Release-Builds
 - Deployed die Release-Builds nach Staging und/oder Production
- Kann **sequentielle (Stages)** und **parallele (Jobs)** Teile haben

CI/CD Pipeline

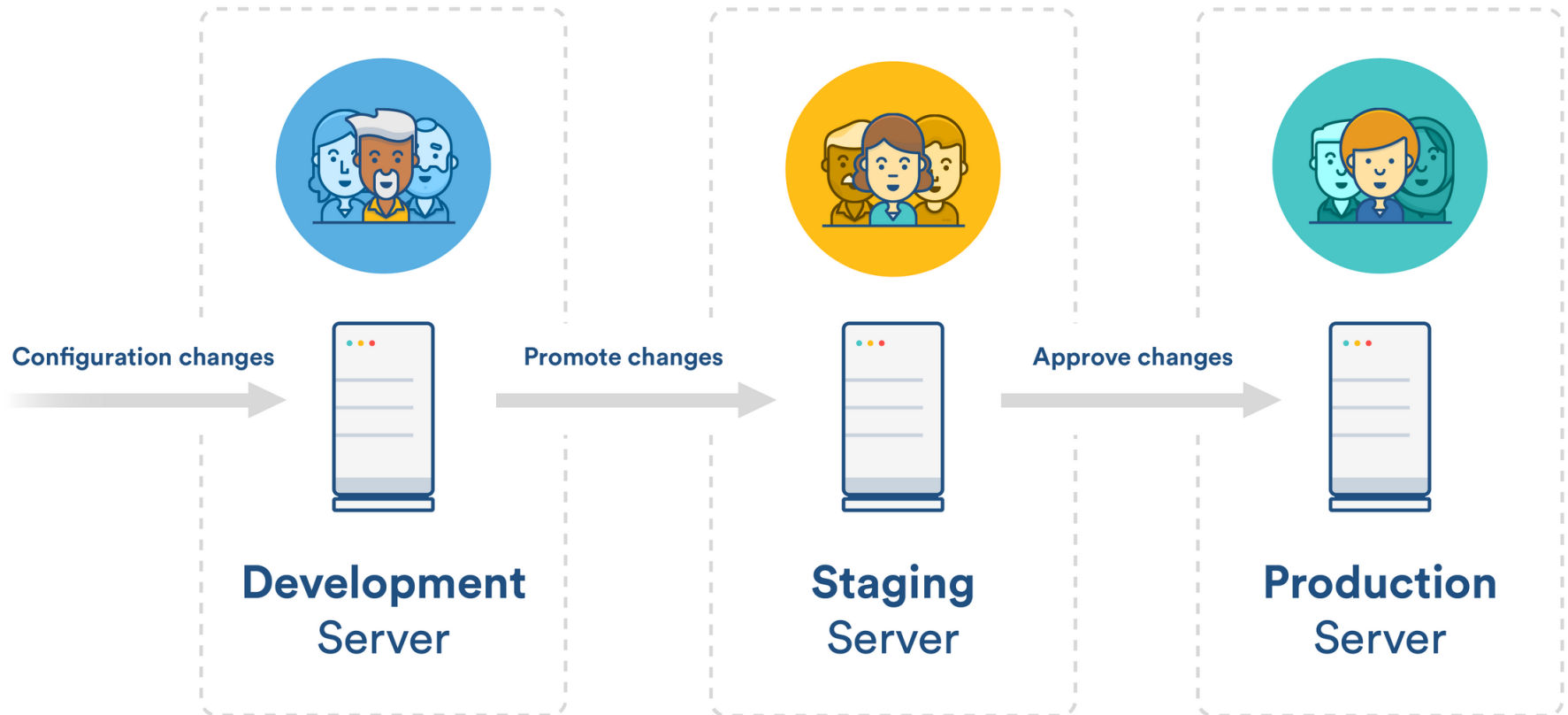


- **Trigger** ist meistens ein **Checkin**
- Pipeline reagiert oft **unterschiedlich** auf Push am **Master** / feature **Branch**, etc...
- Die **Agents** sind meist **Docker-Container**

Build

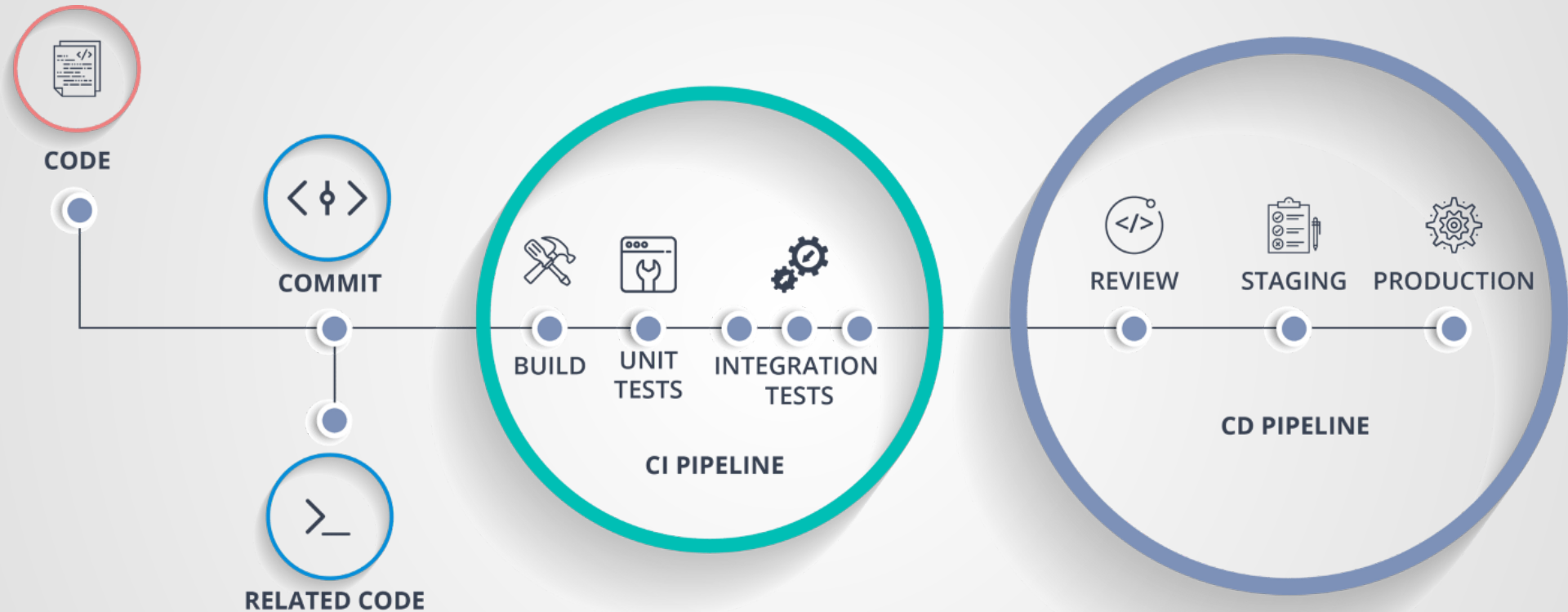
Environments

- Build Environments



- oft läuft auf Staging das Sprint-Ziel (für Reviews, ...)
- Am Beginn des Sprints Staging == Production

- Ein Tool kann für alle Environments genommen werden
- Oft sind das aber auch unterschiedliche Tools



CI/CD

Pipeline

Example

CI/CD Pipeline (Build)

Git repository

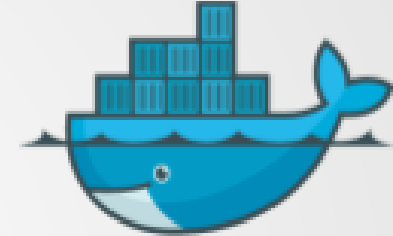


myorg/myrepo

Travis CI build



Docker registry

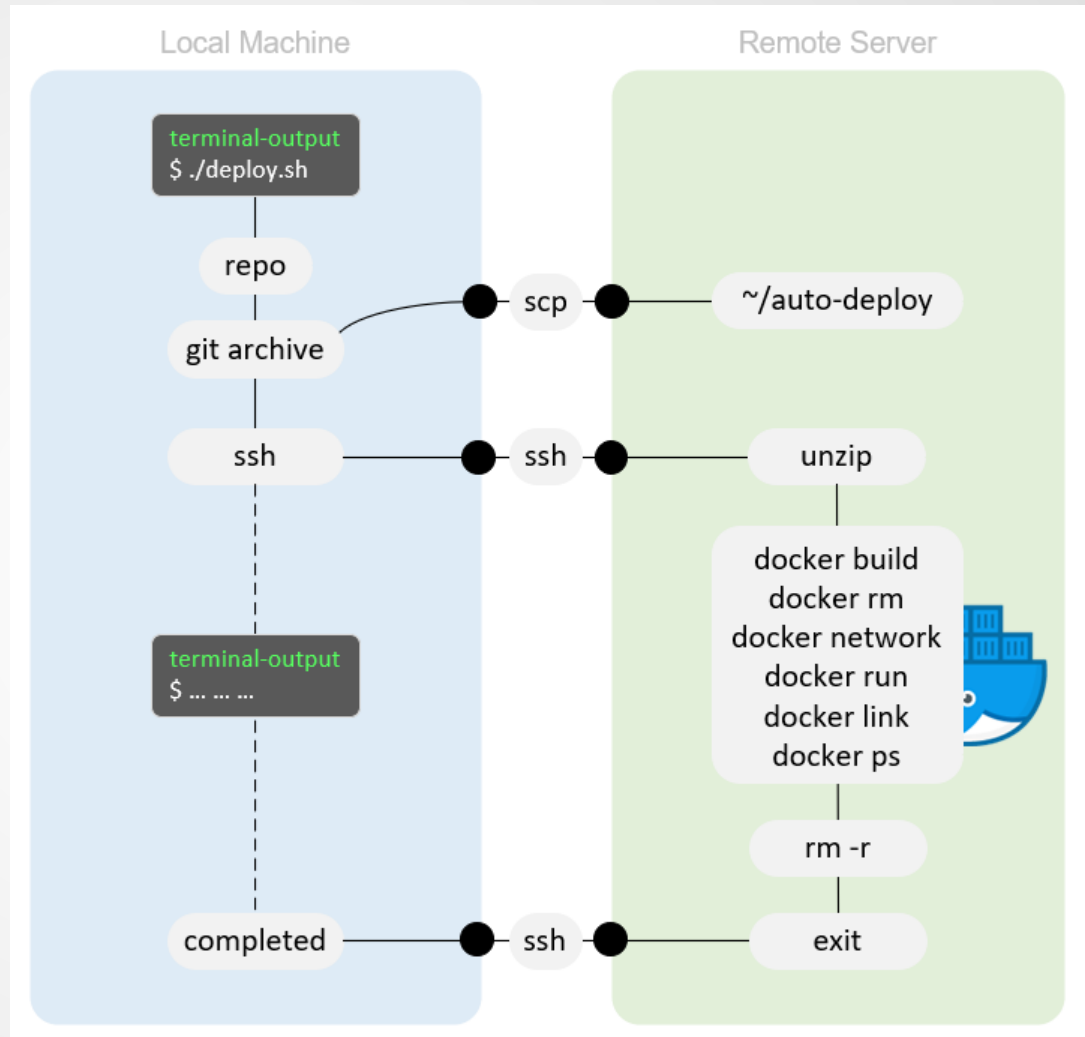


myorg/myimage

- **Push** to Repository
- **Build** is triggered (Github Webhook)
- **Tests** are run (Travis)
- **Image** is **built** (Travis)
- **Image** is **tagged** (Travis)
- **Image** is **pushed** to **Docker.com** (Travis)
 - public or private (login required to download)

CI/CD Pipeline (Deploy)

- SSH to target server
- copy scripts
- execute scripts



- makes target server download and execute the images previously updated

```
1 # Real-life example of a .travis.yml file.
2 branches:
3   only:
4     - master
5 language: java
6 os: linux
7 git:
8   quiet: true
9 jobs:
10   include:
11     - arch: amd64
12       env: BUILD_ARCH=amd64
13     - arch: arm64-graviton2
14       virt: vm
15       group: edge
16       env: BUILD_ARCH=arm64
17     - stage: before-deploy-finish
18       provider: script
19       skip_cleanup: true
20       script: ./$TRAVIS/before_deploy.sh
21       env: BUILD_PUBLISH=true
22     - stage: deploy
23       provider: script
24       skip_cleanup: true
25       script: ./$TRAVIS/deploy.sh
26       env: DOCKER_REGISTRY=false
27     - stage: publish
28       provider: releases
29       skip_cleanup: true
30       overwrite: true
31       api_key: $GITHUB_API_KEY
32       file: $ARTIFACT_ID.$POM_VERSION.zip
33       on:
34         tags: true
35       env: DOCKER_REGISTRY=false
36 ...
```

```
1 # continued...
2 ...
3 - stage: publish
4     provider: releases
5     skip_cleanup: true
6     overwrite: true
7     api_key: $GITHUB_API_KEY
8     file: $ARTIFACT_ID.$POM_VERSION.zip
9     on:
10         tags: true
11         env: DOCKER_REGISTRY=false
12
13 before_install:
14     - git clone https://github.com/UnterrainerInformatik/Travis-Scripts.git travis
15     - source travis/functions.Java.sh
16     - tr_setProjectSubdir Java
17     - source $TRAVIS/before_install.sh
18 install:
19     - source $TRAVIS/install.sh
20 before_script:
21     - source $TRAVIS/before_script.sh
22 script:
23     - source $TRAVIS/script.sh
24     - source $TRAVIS/before_deploy.sh
```

Beispiel für eine Pipeline für einen Multi-Architecture-Build...

✓ master next

Commit 48a6002

Compare 3f70344..48a6002

Branch master

Gerald

→ #75 passed

Ran for 10 min 14 sec

Total time 10 min 10 sec

a day ago

Restart build

Build jobs

View config

✓ test

2 min 55 sec

✓ # 75.1

AMD64

Xenial

</> Java

BUILD_ARCH=amd64

2 min 30 sec

✓ # 75.2

arm6...

Xenial

</> Java

BUILD_ARCH=arm64

2 min 50 sec

✓ before-deploy-finish

1 min 6 sec

✓ # 75.3

AMD64

Xenial

</> Java

BUILD_PUBLISH=true

1 min 6 sec

✓ deploy

1 min 51 sec

✓ # 75.4

AMD64

Xenial

</> Java

DOCKER_REGISTRY=false

1 min 51 sec

✓ publish

1 min 53 sec

✓ # 75.5

AMD64

Xenial

</> Java

DOCKER_REGISTRY=false

1 min 53 sec

Output für den Step #75.2...

✓ master next

Commit 48a6002

Compare 3f70344..48a6002

Branch master

Gerald

</> Java

arm64-graviton2

BUILD_ARCH=arm64

#75.2 passed

Ran for 2 min 50 sec

a day ago

Restart job

Debug job

Job log

View config

Remove log Raw log

```
1 Worker information
6
7 Build system information
142
143
144 Installing SSH key from: default repository key
146 Using /home/travis/.netrc to clone repository.
147
148 $ git clone --depth=50 --branch=master --quiet https://github.com/UnterrainerInformatik/java-cms-data-
151
152
153 Setting environment variables from repository settings
154 $ export DEPLOY=true
155 $ export DEPLOYMENT_SERVER=lan [secured zettl at
```

Deployments

... how to deploy?

Deployments

Wie bekomme ich mein Programm auf einen Rechner beim Kunden?

- Mit **Datenträger** hinfahren und manuell installieren
- Kunden ein **Installationsprogramm** schicken
- Kunden ein **Installationsprogramm** herunterladen lassen
- Kunden eine **Docker-Image** verwenden lassen

Deployments

Wie bekomme ich mein Programm auf einen Rechner beim Kunden?

Auf den **Server** via **SSH** und eventuell **VPN-Tunnel** verbinden und ...

- ... ein **Image raufkopieren** und starten
- ... ein Skript raufkopieren und starten, das das Image herunterlädt und startet (**docker-compose**)

Deployments

... configuring

(Environment Variables)

Deployment Configuration

- Deployments haben verschiedene Variablen, die sich je nach Kunde oder Installationsort ändern können
 - **IP** des Servers, Adresse des **Backends**, Adresse des **SQL Servers**, **Datenbankname**, **Passwort** der Datenbank, **SMS-Gateway** Zugangsdaten, **Mail**-Zugangsdaten, **Ports** auf denen der Server laufen soll, **Name** des Containers, **Frontend-URL** für Reverse-Proxy, ...
- Diese realisiert man meist mit Environment-Variablen

Deployment Configuration

- Environment-Variablen sind bequemer, weil
 - Sie an verschiedenen **Orten** stehen können und verschiedene **Prioritäten** haben können
- Beispiel Quarkus:

```
(400) System properties like `myapp -Dquarkus.datasource.password=thisisnotarealpwd` (LOWEST PRIO)
(300) Environment variables
(295) .env file in the current working directory
(260) Quarkus Application config file in $PWD/config/application.properties
(250) Quarkus Application config file application.properties in classpath
(100) MicroProfile Config file META-INF/microprofile-config.properties (HIGHEST PRIO)
```

- Damit kann ich die unterschiedlichsten Deployments bedienen
(Entwicklermaschine, Dev-Server, Staging-Server, Deployments)

Deployment Configuration

- Environment-Variablen sind bequemer, weil
 - Es auf jeder Ebene schon **1000 fertige Mechanismen** gibt, die es mir erlauben von überall aus auf diese Variable **zuzugreifen...**
 - ... oder sogar sie an einer bestimmten Stelle **zu setzen**.
 - Denn manche Dinge weiß ich erst NACH dem Einchecken (z.B. **Semantic Version**)
 - Manche Dinge will ich im Build-Prozess belassen und nicht nach außen sichtbar machen (welche **ARCH** ist das gerade **AMD64** oder **arm64**)

Deployment Configuration

- Environment-Variablen sind bequemer, weil
 - es überall Möglichkeiten gibt **Passwörter** oder **Zertifikate geschützt** zu setzen, ohne dass sie im Klartext im Repo landen



References

- https://en.wikipedia.org/wiki/Continuous_integration
- https://en.wikipedia.org/wiki/Continuous_delivery
- <https://en.wikipedia.org/wiki/DevOps>
- <https://quarkus.io/guides/config-reference>
- <https://towardsdatascience.com/continuous-integration-continuous-delivery-myths-pitfall-and-practical-approach-aaec22edacc5>
-