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

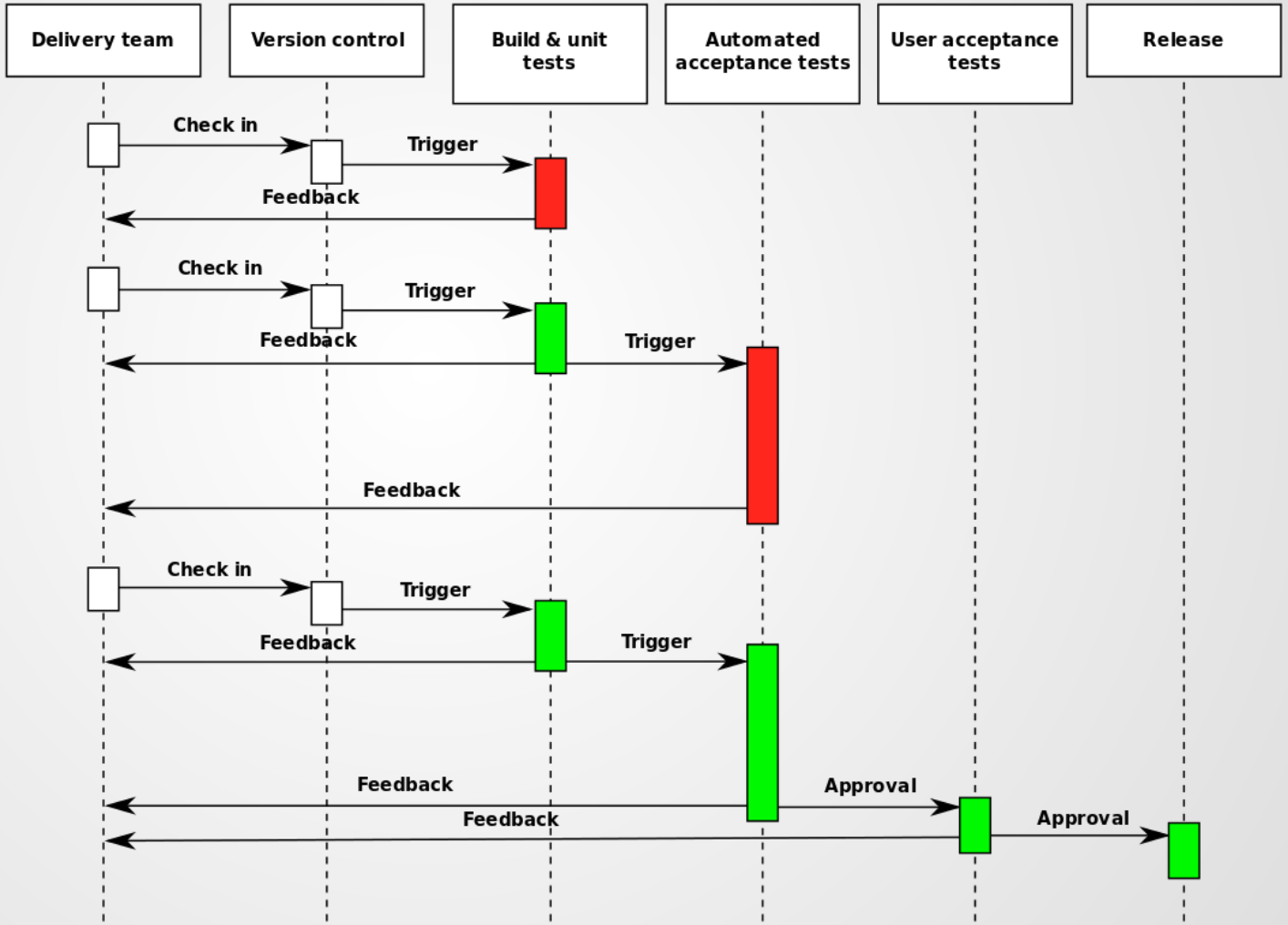
- Oft neue Versionen bauen
- Jede Version ist ein ernstzunehmender Release Candidate
- Automatisierte Tests laufen nach jedem Checkin
- Kann auch manuelle Tests beinhalten

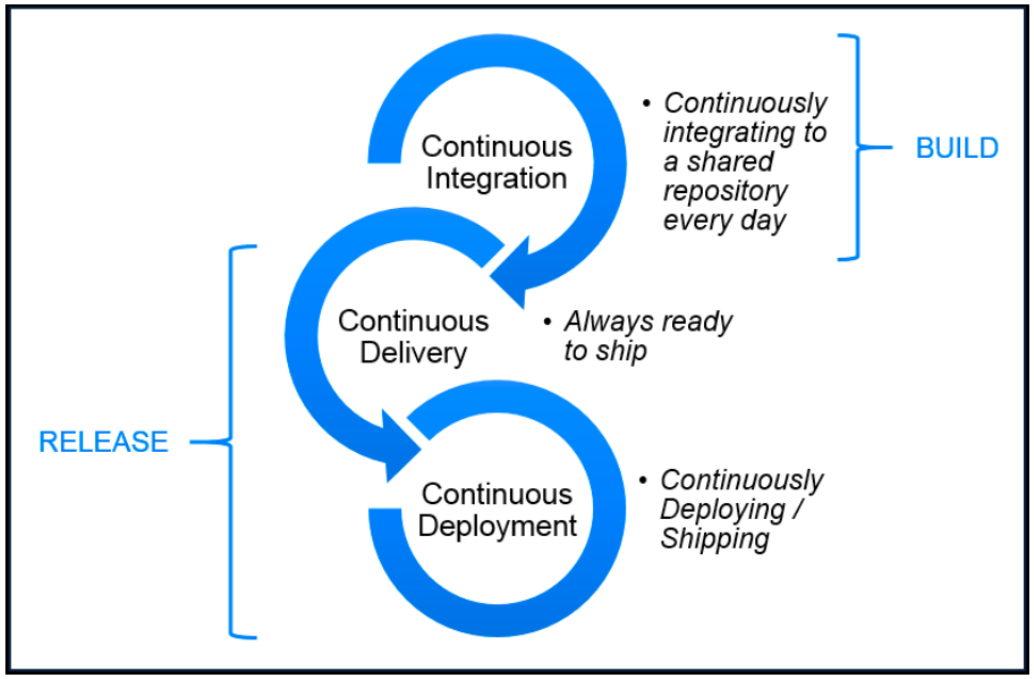
Unterschied Delivery/Deployment

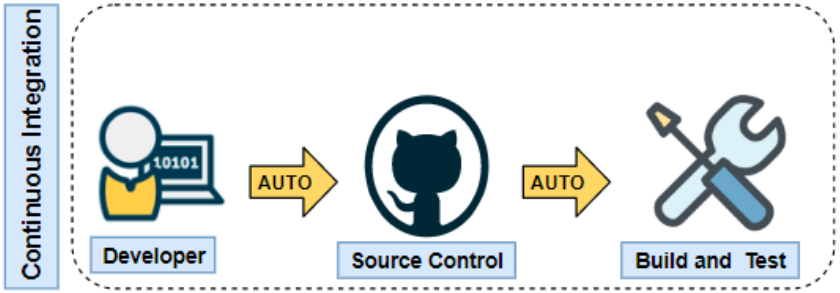
- "Continuous Delivery" hat **manuelles** Deployment
- "Continuous Deployment" hat **automatisches** Deployment

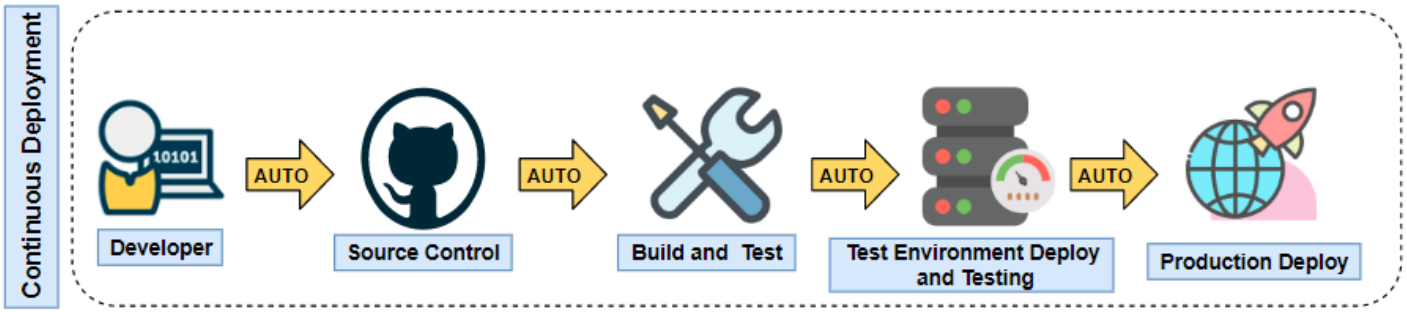
Continuous (Delivery) Deployment

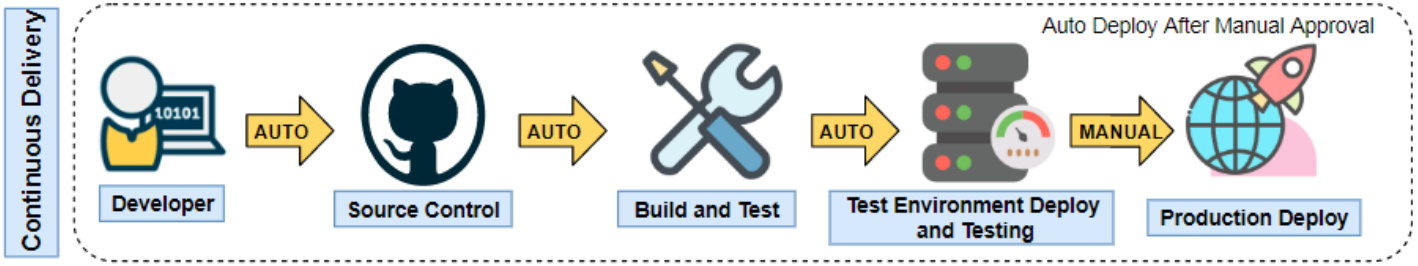
teilautomatische Pipeline...




Continuous Integration
• Continuously integrating to a shared repository every day
BUILD
Continuous Delivery
• Always ready to ship
RELEASE
Continuous Deployment
• Continuously Deploying / Shipping


Continuous Integration
10101
AUTO
AUTO
Developer
Source Control
Build and Test


Continuous Deployment
10101
AUTO
AUTO
AUTO
AUTO
Developer
Source Control
Build and Test
Test Environment Deploy and Testing
Production Deploy


Continuous Delivery
Auto Deploy After Manual Approval
10101
AUTO
AUTO
AUTO
MANUAL
Developer
Source Control
Build and Test
Test Environment Deploy and Testing
Production Deploy

DevOps

DevOps
(Development / Operations)

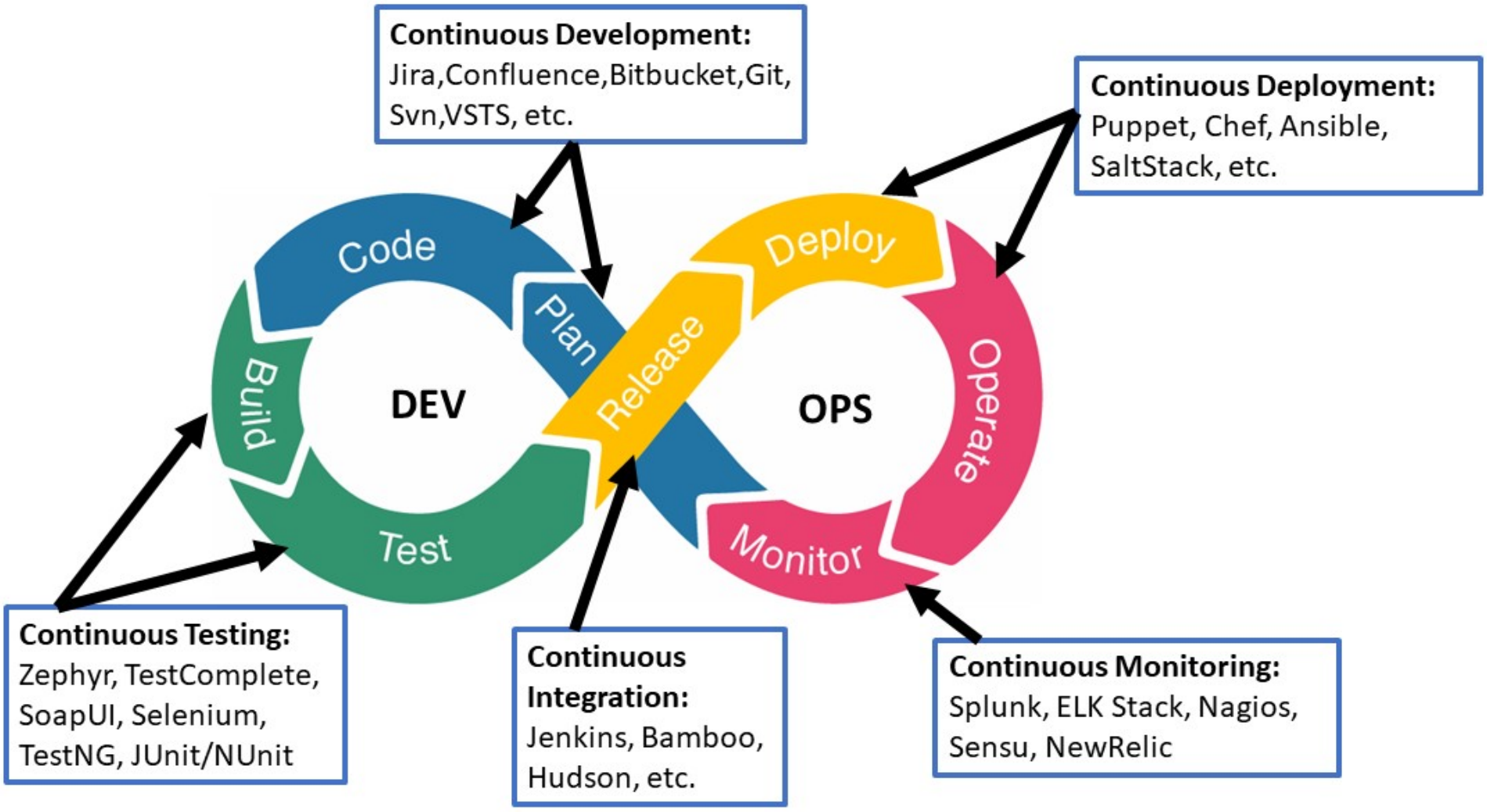
- Kulturelle Änderungen innerhalb der Softwarefirma
- Automatische Builds
  - CI/CD sind notwendig
- Zusammenarbeit verschiedener Teams
  - Developers
  - Operations
  - Quality Assurance
  - Management
  - ...
- um ein Produkt **automatisiert** zu releasen **und zu warten**

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.
Code
Plan
Build
DEV
Release
Deploy
Operate
OPS
Test
Monitor
Continuous Testing:
Zephyr, TestComplete,
SoapUI, Selenium,
TestNG, JUnit/NUnit
Continuous
Integration:
Jenkins, Bamboo,
Hudson, etc.
Continuous Monitoring:
Splunk, ELK Stack, Nagios,
Sensu, NewRelic

CI/CD Pipeline

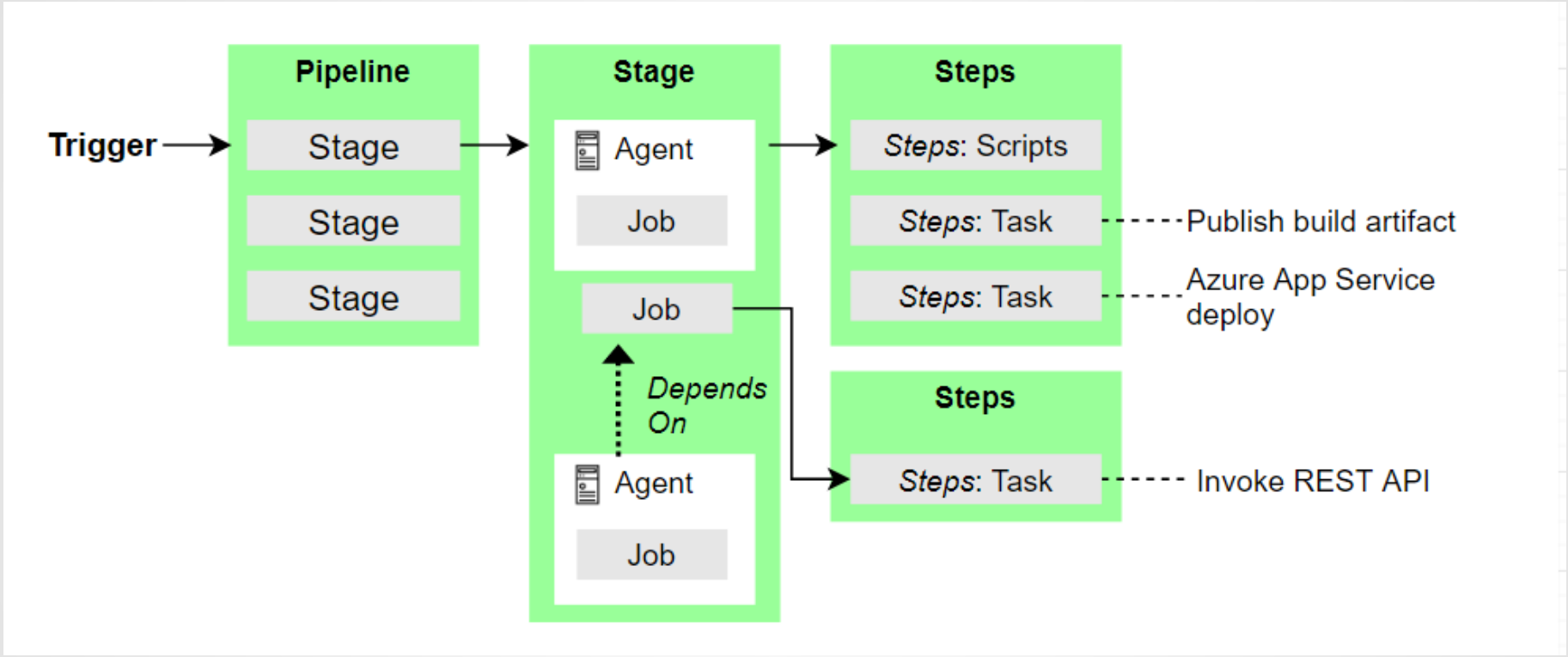
Build vs. CI/CD Pipeline

- 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

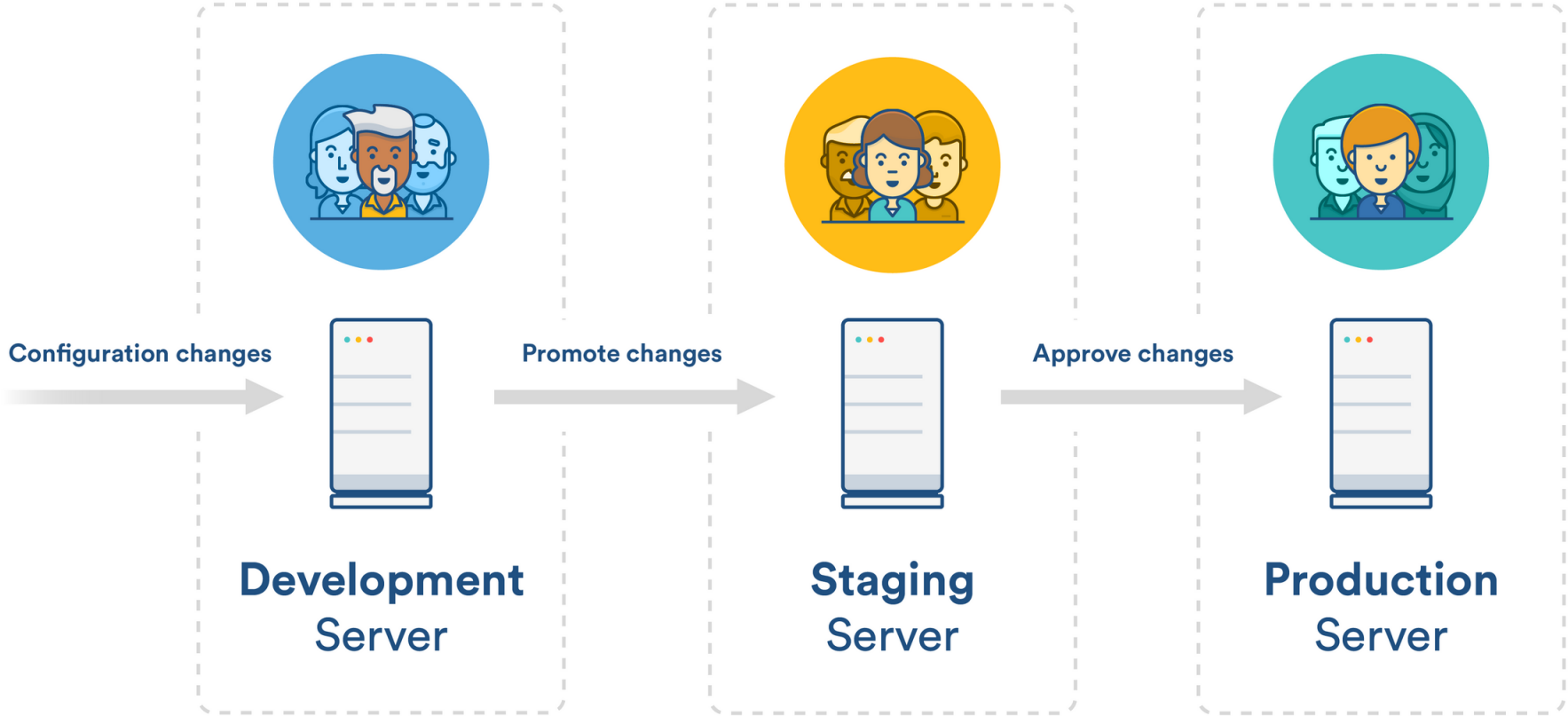
CI/CD Pipeline



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

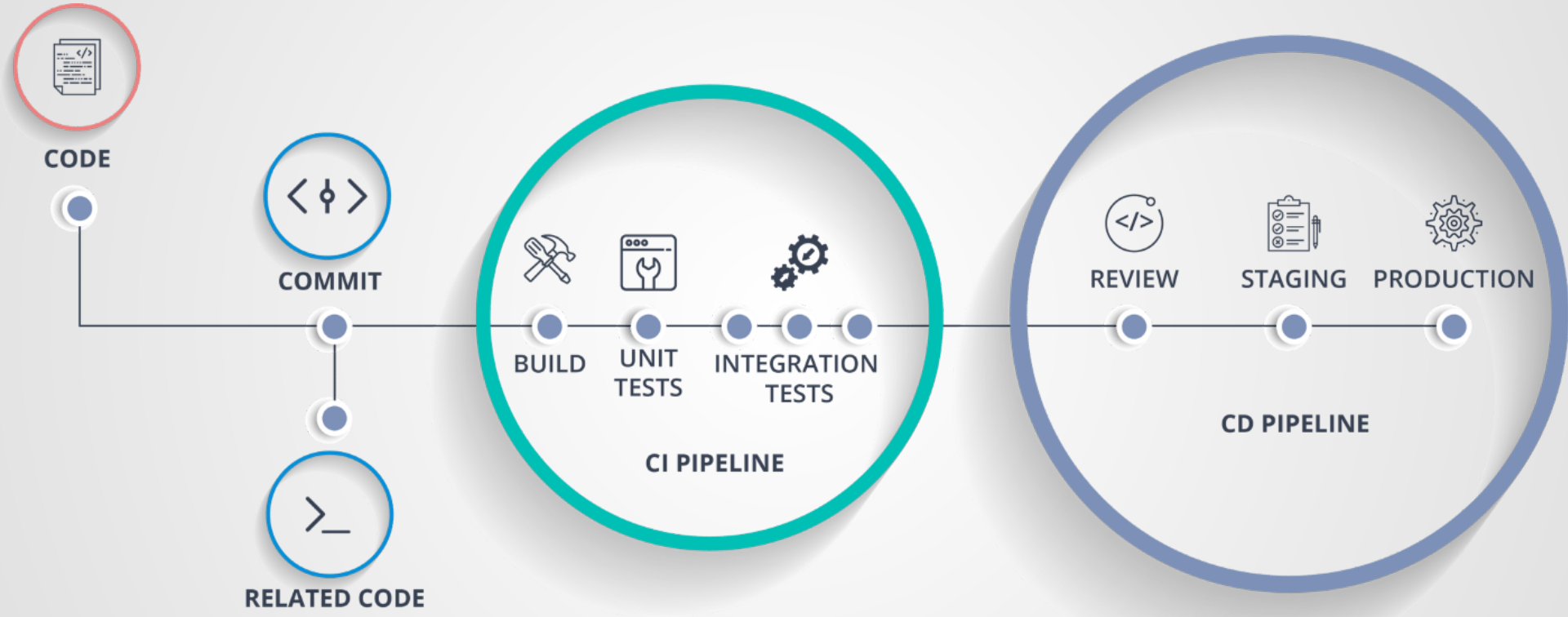
Build Environments

- Build Environments



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

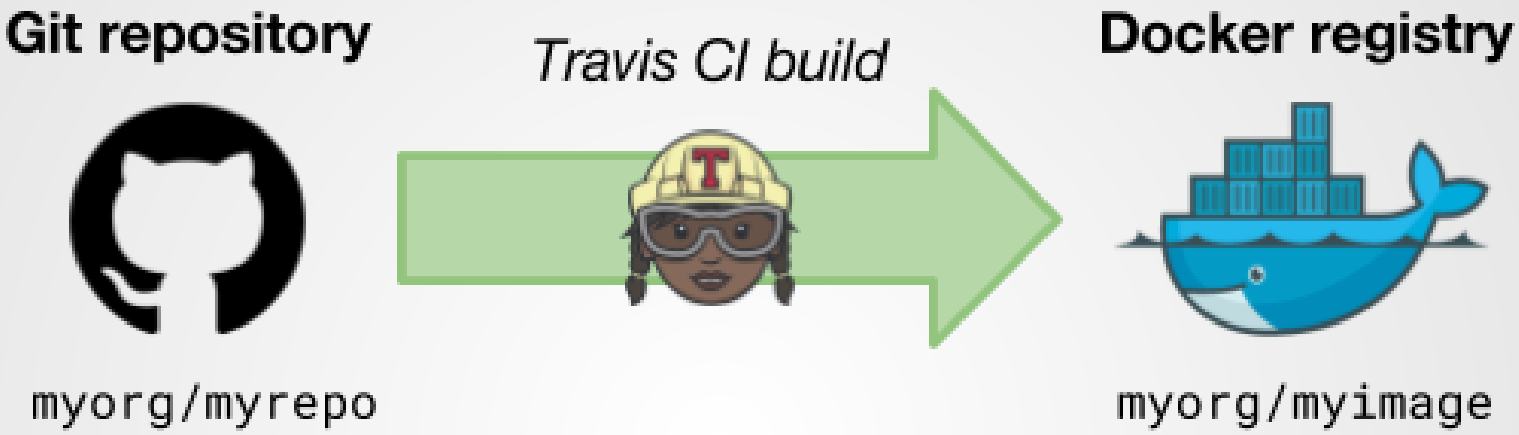
- Ein Tool kann für alle Environments genommen werden
- Oft sind das aber auch unterschiedliche Tools



CI/CD Pipeline

Example

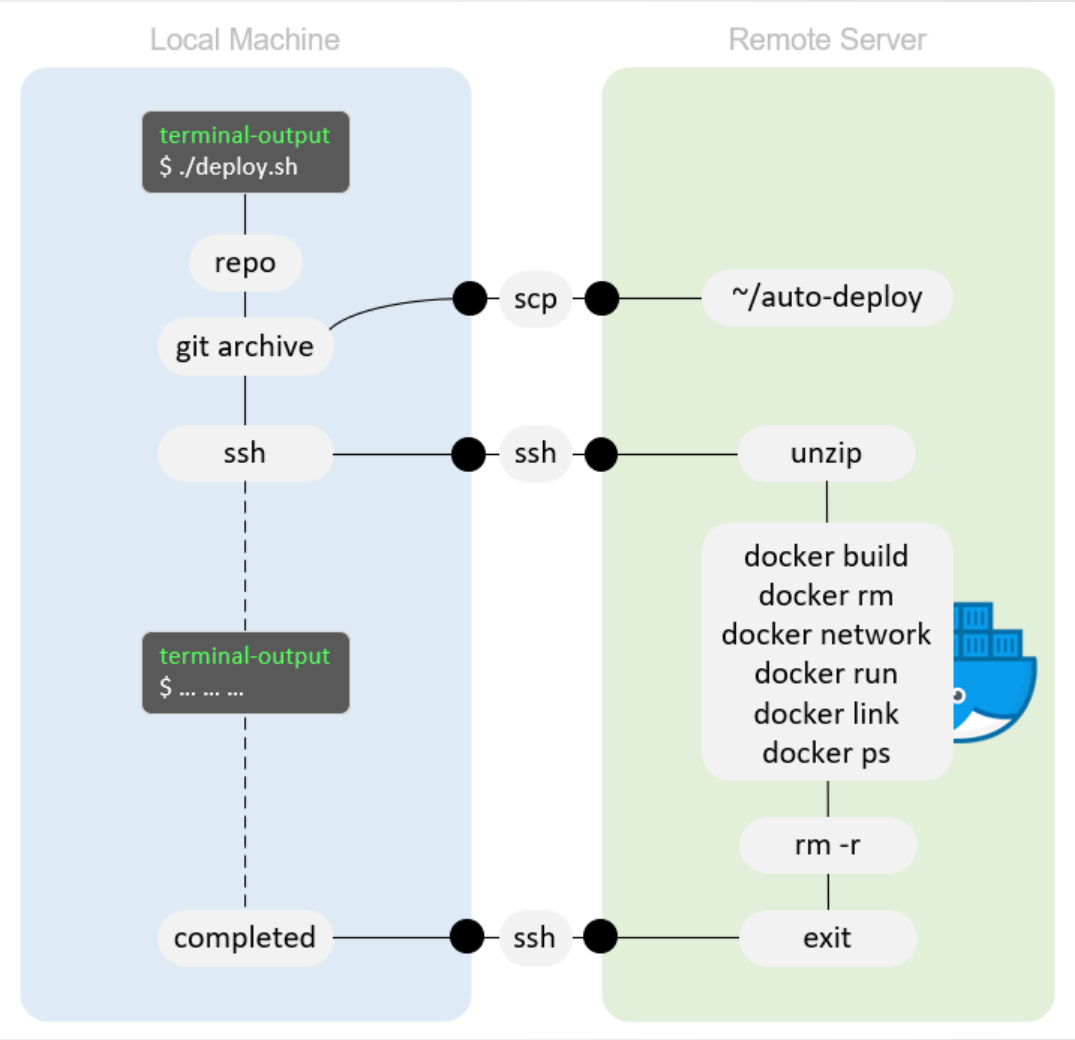
CI/CD Pipeline (Build)



- **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



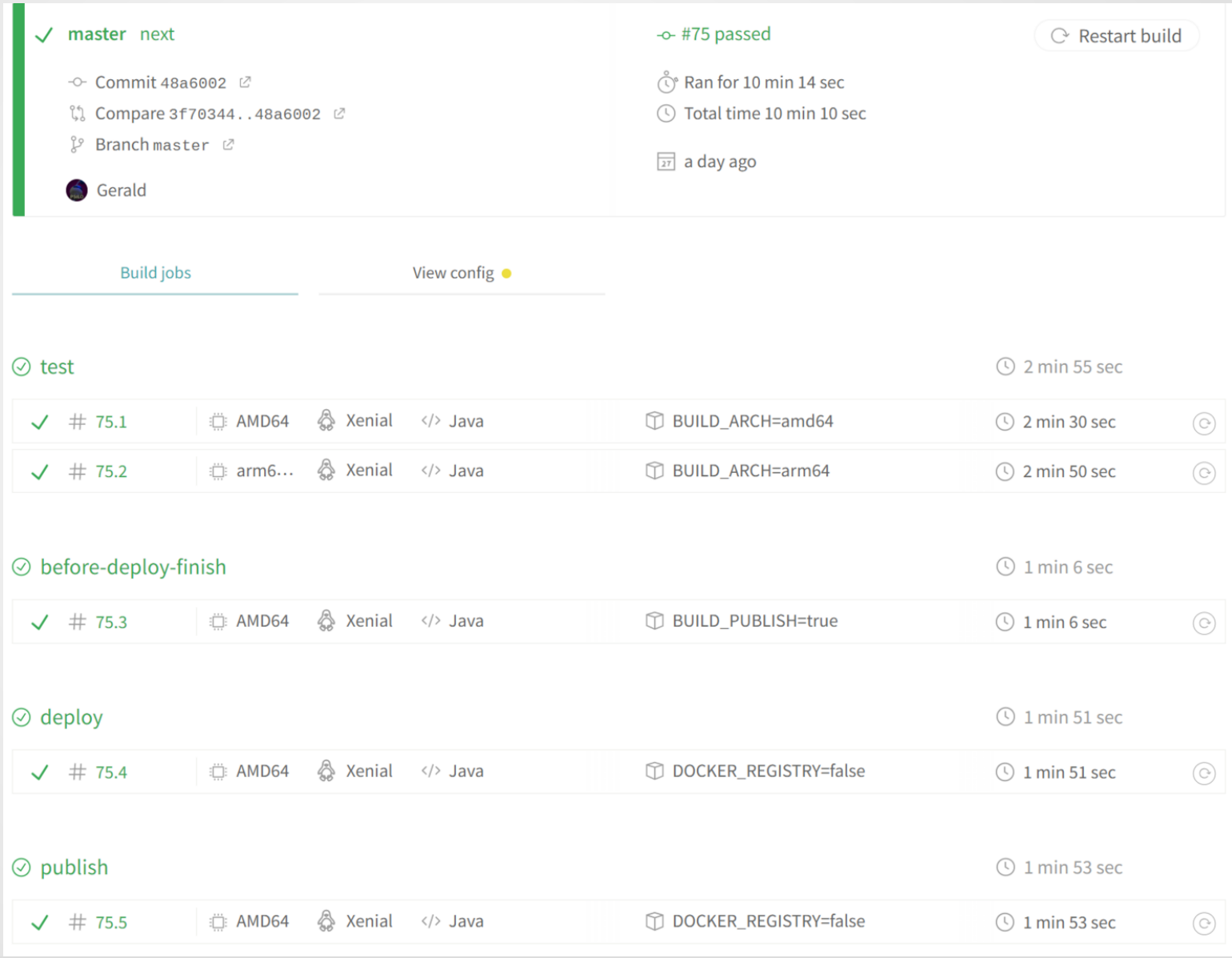
- makes target server download and execute the images previously updated

```yaml
# Real-life example of a .travis.yml file.
branches:
  only:
    - master
language: java
os: linux
git:
  quiet: true
jobs:
    include:
        - arch: amd64
          env: BUILD_ARCH=amd64
        - arch: arm64-graviton2
          virt: vm
          group: edge
          env: BUILD_ARCH=arm64
        - stage: before-deploy-finish
          provider: script
          skip_cleanup: true
          script: ./$TRAVIS/before_deploy.sh
          env: BUILD_PUBLISH=true
        - stage: deploy
          provider: script
          skip_cleanup: true
          script: ./$TRAVIS/deploy.sh
          env: DOCKER_REGISTRY=false
        - stage: publish
          provider: releases
          skip_cleanup: true
          overwrite: true
          api_key: $GITHUB_API_KEY
          file: $ARTIFACT_ID.$POM_VERSION.zip
          on:
            tags: true
          env: DOCKER_REGISTRY=false
...
```

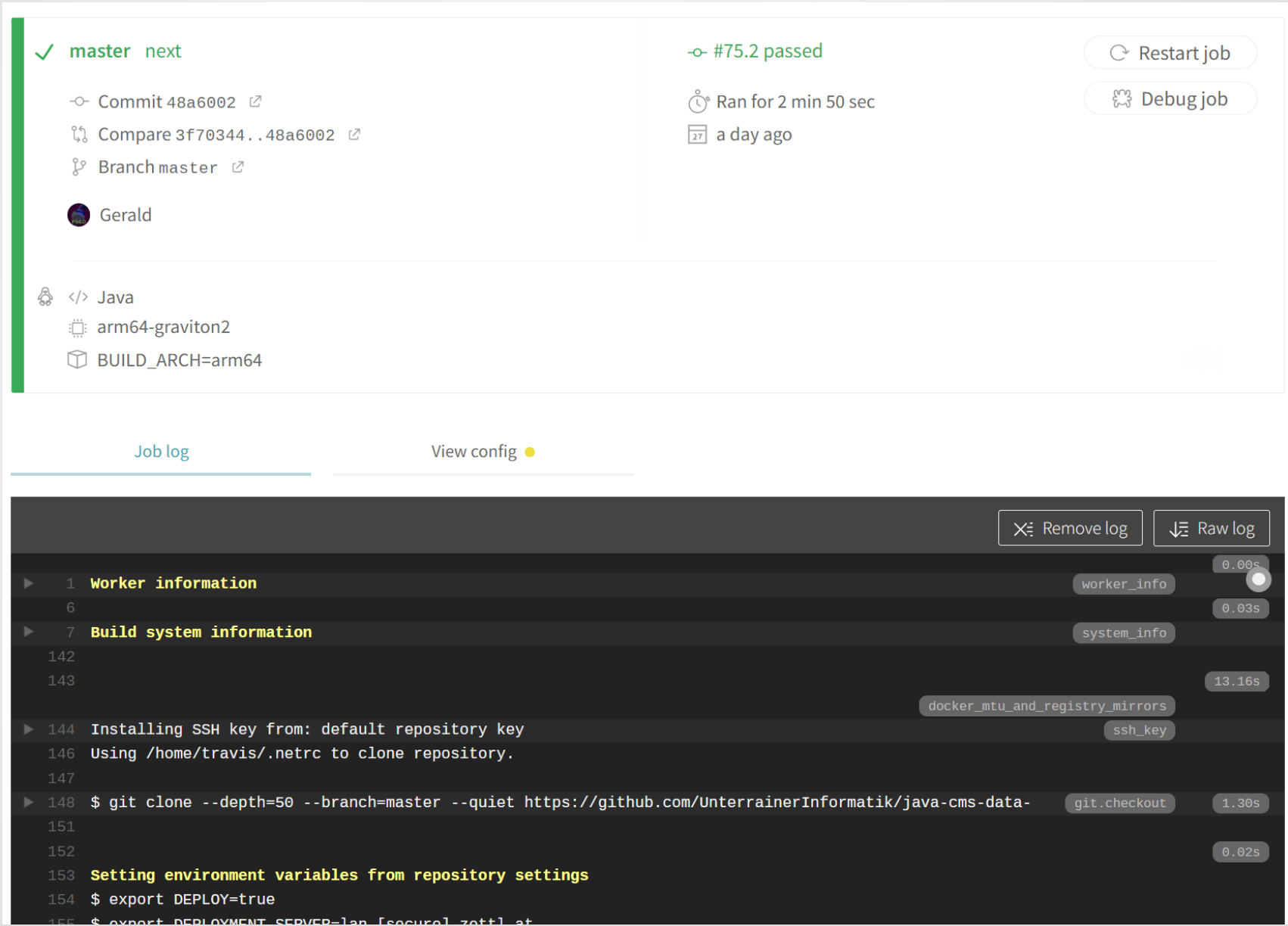
```yaml
# continued...
...
- stage: publish
          provider: releases
          skip_cleanup: true
          overwrite: true
          api_key: $GITHUB_API_KEY
          file: $ARTIFACT_ID.$POM_VERSION.zip
          on:
            tags: true
          env: DOCKER_REGISTRY=false

before_install:
    - git clone https://github.com/UnterrainerInformatik/Travis-Scripts.git travis
    - source travis/functions.Java.sh
    - tr_setProjectSubdir Java
    - source $TRAVIS/before_install.sh
install:
    - source $TRAVIS/install.sh
before_script:
    - source $TRAVIS/before_script.sh
script:
    - source $TRAVIS/script.sh
    - source $TRAVIS/before_deploy.sh
```

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



Output für den Step #75.2...



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 I**mage 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
- Ich verschiedene Ebenen habe, die sich hierarchisch überschreiben
- Ich die gesamte Variablen-Ersetzungs-Funktionalität in den verschiedenen Ebenen nicht selber schreiben muss

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
-