

# CI/CD-Pipeline im MSD

---

# Inhalt

- Einführung CI/CD-Pipeline
  - Was ist eine CI/CD-Pipeline?
  - Grundlegende Begriffe und Konzepte
  - Konfigurationsdatei einer Pipeline
  - Gängige Keywords
  - Workflow einer Pipeline
  - Nutzen von Pipelines
- Vorstellung der CI/CD-Pipeline im MSD
  - Common-CI-CD
  - Funktionsweise
- Beispiel:
  - Skeleton Player Java Spring

# Was ist eine CI/CD-Pipeline?

- CI = Continuous Integration
- CD = Continuous Delivery/Deployment
- Automatisierter Prozess
- Automatisiert den Softwarebereitstellungsprozess
- Unterstützt den kontinuierlichen Softwareentwicklungsprozess
- Übernimmt Routineaufgaben
- Definiert in einer Konfigurationsdatei
- Ausführung wird durch Ereignis ausgelöst
  - Pushen von neuem Code auf das Repository
  - Manuelles Anstoßen
  - ...

# Was ist eine CI/CD-Pipeline?

- Jobs grob einteilbar in drei Kategorien:
  - Code bauen
  - Code testen
  - Code deployen
- **Build:** Code wird kompiliert und in ausführbare Form (Artefakte) gebracht
  - JAR, WAR, Image, Helm Chart, ...
- **Test:** Automatisierte Tests werden ausgeführt
  - Unit, Integration, System, ...
- **Deployment:** Code (Artefakte) wird bereitgestellt
  - Server, Registry, ...

# Grundlegende Begriffe und Konzepte

- **Job:** Einzelner Schritt in der Pipeline
  - Führt Reihe von Skript Befehlen aus
  - Build, Testlauf, Deployment, Analyse, ...
- **Stage:** Gruppe von Jobs
  - Jobs werden parallel ausgeführt
- **Pipeline:** Automatisierter Prozess
  - Besteht aus mehreren Stages
  - Stages werden nacheinander ausgeführt
- **Runner:** Maschine oder Container
  - Führt die Jobs der Pipeline aus
  - Müssen entsprechend konfiguriert sein

# Konfigurationsdatei einer Pipeline

- Hinterlegt im Root-Verzeichnis
- Im Yaml-Format
- Definiert
  - Jobs
  - Stages
  - Globale Konfigurationen
- Weitreichende Konfiguration möglich (über Keywords)
  - Image imputierbar
  - Artefakte persistierbar
  - Jobs mit Bedingungen versehbar
  - Dependencies
  - ...

```
stages:  
  - build  
  - test  
  - deploy  
  
build-project:  
  stage: build  
  script:  
    - echo "Building the project..."  
    - # Befehle zum Bauen des Projekts  
  
test-project:  
  stage: test  
  script:  
    - echo "Running tests..."  
    - # Befehle zum Ausführen der Tests  
  
deploy-project:  
  stage: deploy  
  script:  
    - echo "Deploying the project..."  
    - # Befehle zum Bereitstellen des Projekts
```

# Gängige Keywords

- Keywords
  - **Globale Keywords:** Scope = Konfigurationsdatei
    - default: Definiert Custom Job Level Default Werte
    - stages: Definiert Phasen der Pipeline
    - variables: Definiert globale Variablen
    - include: Inkludiert externe Konfig-Dateien
  - **Job-Keywords:** Scope = Job
    - variables: Definiert Job Variablen
    - image: Definiert das Job Image
    - services: Zusätzliche Service als Container
    - script: Shell Befehle
    - artifacts: Generierte Dateien, welche persistiert / weitergereicht werden sollen
- [CI/CD YAML syntax reference | GitLab](#)

# Workflow einer Pipeline

- **Auslösung der Pipeline**
  - Push-Ereignis
  - Manuelle Ausführung
  - Zeitplan (Cron-Job)
- **Auslesen und Interpretieren der Pipeline Konfigurationsdatei im Root-Verzeichnis**
  - .gitlab-ci.yaml in gitlab
  - Definiert Jobs und Stages
- **Zuweisung des Runners**
- **Ausführung der Pipeline**
  - Stage für Stage
  - Repo wird geklont
  - GitLab setzt spezifische Umgebungsvariablen
  - Skriptausführung
  - Statusberichte nach Job
  - Erzeugte Artefakte können persistiert werden
- **Feedback und Benachrichtigungen**
  - Logs
  - Erfolgsstatus
  - Email



# Nutzen von Pipelines

- **Reduzierung manueller und wiederkehrender Arbeit**
  - Spart Entwickler Zeit
  - Reduziert Möglichkeit menschlicher Fehler
- **Reproduzierbarkeit von Build-, Test- und Deployment-Schritten**
  - Steigert Konsistenz und Zuverlässigkeit
  - Prozesse unter gleichen Bedingungen wiederholbar
- **Schnelles Feedback**
  - Bei Fehlern/Problemen
- **Frühe Fehlererkennung**
  - Vor Deployment in die Produktionsumgebung
- **Automatisiertes Deployment**
  - Schnellere Releases
- **Automatisierte Codequalitätssicherung**
  - Analyse-Tools, Linter, etc
  - Steigert Codequalität

# CI/CD-Pipeline im MSD

- Erfolgreiches Deployment benötigt 2 Artefakte:
  - Image
    - [Container Registry · The-Microservice-Dungeon / DevOps-Team / msd-image-registry · GitLab](#)
  - Helm-Chart
    - [Package Registry · The-Microservice-Dungeon / DevOps-Team / msd-image-registry · GitLab](#)

# Common-CI-CD

- Pipeline Konfigurationsdateien für den MSD
  - [The-Microservice-Dungeon / DevOps-Team / common-ci-cd · GitLab](#)
- Aktuell zwei Konfigdateien
  - Container
    - Stage = build\_container\_push
    - Build und Push des Docker Images
  - Helm
    - Stage = helm
    - Build und Push des Helm Charts
- Verwendung über include-Tag

Vielen Dank fürs  
Zuhören



# Quellenverzeichnis

- [What is CI/CD? | GitLab](#)
- [Get started with GitLab CI/CD | GitLab](#)
- [Was ist eine CI/CD-Pipeline? Continuous Integration and Delivery \(redhat.com\)](#)
- [Was ist eine CI/CD-Pipeline? | Splunk](#)
- [Grundlagen einer Continuous Delivery-Pipeline | Atlassian](#)
- [Predefined CI/CD variables reference | GitLab](#)
- [CI/CD YAML syntax reference | GitLab](#)