

The background of the slide features a faded image of a multi-story brick building on the left and a church with a tall, ornate stone tower on the right. A large, white, stylized 'S' shape is overlaid on the right side of the image.

# Tag 3: GitOps, Docker in der Entwicklung und Deployment-Strategien

19.06.2024, Daniel Krämer & Malte Fischer

© Copyright 2024 anderScore GmbH

- **Tag 1 – Einführung in Git und GitLab, Git-Workflow im Team**
  - Einführung & Kursüberblick
  - Grundlagen von Git
  - Git Rebase und Merge-Strategien
  - Git Remote
  - Grundlagen von GitLab
  - Git-Workflow im Team
- **Tag 2 – Vertiefung Git-Workflow, CI/CD & GitLab CI**
  - Gitflow-Workflow
  - Tags, Releases & deren Verwaltung
  - GitLab-Runner
  - Einführung in GitLab CI/CD & gitlab.yml
- **Tag 3 – GitOps, Docker in der Entwicklung und Deployment-Strategien**
  - GitOps Grundlagen
  - Lokale Entwicklung mit Docker
  - Container/Docker-Registry
  - Erstellen von Release- und Tagged-Images
  - Möglichkeiten des Deployments & Verwaltung von Konfiguration
  - Abschlussübung & Diskussion

- **Tag 1 – Einführung in Git und GitLab, Git-Workflow im Team**
  - Einführung & Kursüberblick
  - Grundlagen von Git
  - Git Rebase und Merge-Strategien
  - Git Remote
  - Grundlagen von GitLab
  - Git-Workflow im Team
- **Tag 2 – Vertiefung Git-Workflow, CI/CD & GitLab CI**
  - Gitflow-Workflow
  - Tags, Releases & deren Verwaltung
  - GitLab-Runner
  - Einführung in GitLab CI/CD & gitlab.yml
- **Tag 3 – GitOps, Docker in der Entwicklung und Deployment-Strategien**
  - GitOps Grundlagen
  - Lokale Entwicklung mit Docker
  - Container/Docker-Registry
  - Erstellen von Release- und Tagged-Images
  - Möglichkeiten des Deployments & Verwaltung von Konfiguration
  - Abschlussübung & Diskussion

Grundlagen von

# Abschlussübung & Diskussion

## Inhalt

- Key Takeaways
- Abschlussübung
- Optional: Komplexe Pipeline mit Docusaurus und npm

## Key Takeaways

- Workflow
- Versionierungsstrategie
- Continuous Integration (CI)
- Continuous Delivery (CD)
- Unterschiede CI und CD

## Workflow

- Ein gut definierter Workflow skizziert, wie Codeänderungen von der Entwicklung bis zur Bereitstellung fortschreiten.
- Gitflow: Branching-Modell, das zwischen Feature-Banches, Release-Banches und Hotfixes unterscheidet, geeignet für Projekte mit klar definierten Release-Zyklen.
- Custom Workflow: Einige Projekte erfordern maßgeschneiderte Workflows, die spezifische Release-Strategien oder regulatorische Anforderungen berücksichtigen.
- Der definierte Workflow dient als Fahrplan für das Entwicklungsteam und sorgt für konsistente Praktiken und effiziente Code-Integration.

## Versionierungsstrategie

- Eine gut definierte Versionierungsstrategie gewährleistet Konsistenz und Rückverfolgbarkeit der Software-Releases.
- Semantische Versionierung (SemVer): Klare Formatierung von MAJOR.MINOR.PATCH, weit verbreitet und kommuniziert Kompatibilität und Änderungen effektiv.
- Custom Versioning: Für Projekte mit speziellen Anforderungen kann eine benutzerdefinierte Versionierungsstrategie besser geeignet sein.
- Klare Versionierung hilft, die Bedeutung jedes Releases zu verstehen und ermöglicht es Benutzern, fundierte Entscheidungen über Updates und Kompatibilität zu treffen.



## Continuous Integration (CI)

- Ist eine Vorgehensweise in der Softwareentwicklung, bei der die häufige Integration von Codeänderungen in ein gemeinsames (remotes) Repository im Vordergrund steht.

## Vorteile

- Ermöglicht die frühzeitige Erkennung im Entwicklungszyklus von Integrationsproblemen und Bugs, wodurch Kosten und Aufwand für deren Behebung reduziert werden
- Entwickler können in kleineren, überschaubaren Schritten arbeiten und ihre Änderungen schnell integrieren und testen
- Teams arbeiten kollaborativer, da CI das Teilen von Code und das schnelle Lösen von Konflikten fördert
- CI stellt sicher, dass der Code stets bau- und testbar ist.

## CI Best Practises

- Versionskontrolle (VCS): Verwenden Sie ein robustes Versionskontrollsystem wie Git zur Verwaltung von Codeänderungen.
  - Die Wahl des VCS sollte mit dem Workflow, den Projektanforderungen und den Skalierungsplänen des Teams übereinstimmen.
- Automatisierte Tests: Schreiben Sie Unit-Tests, Integrationstests und End-to-End-Tests zur automatischen Validierung von Codeänderungen.
- Automatisierte Builds: Richten Sie automatisierte Build-Pipelines ein, die Code kompilieren, Anwendungen paketieren und Artefakte erstellen.
- Häufige Integration: Ermutigen Sie Entwickler, Codeänderungen mehrmals täglich in das gemeinsame Repository zu integrieren.
- Feedback-Schleifen: Implementieren Sie Feedback-Mechanismen, um Entwickler über Build- und Testergebnisse zu informieren.

## Tools:

- GitLab CI/CD, Jenkins, CircleCI, ...

## Continuous Delivery (CD)

- CD erweitert die Prinzipien von CI zur Automatisierung der Anwendungsbereitstellung in verschiedene Umgebungen, einschließlich Produktion.
- CD stellt sicher, dass Software immer in einem einsatzbereiten Zustand ist und jederzeit in Produktion gehen kann.

## Vorteile

- Automatisierung des Bereitstellungsprozesses, Reduzierung des Risikos menschlicher Fehler bei Releases.
- Häufigere Veröffentlichung neuer Funktionen und Fehlerbehebungen, schnellere Wertschöpfung für Benutzer.
- Konsistente Bereitstellungsprozesse in allen Umgebungen, schnelle Verbesserungen durch häufige Releases und Benutzerfeedback.

## CD Best Practises

- Automatisierte Tests: Kontinuierliche Tests in Staging-Umgebungen, die die Produktion nachahmen.
- Bereitstellungspipelines: Erstellen Sie Bereitstellungspipelines mit mehreren Stufen, einschließlich Testing, Staging und Produktion.
- Blue-Green Deployments: Implementieren Sie Blue-Green Deployments zur Minimierung von Ausfallzeiten bei Releases.
- Rollbackstrategien: Definieren Sie Rollbackstrategien für den Fall, dass Probleme bei der Bereitstellung auftreten.
- Infrastruktur als Code (IaC): Definieren Sie Infrastruktur mit Code (z.B. Terraform oder AWS CloudFormation) für konsistente und reproduzierbare Umgebungen.

## Tools

- Docker, Kubernetes, Jenkins, Spinnaker, ...

## Unterschiede CI und CD

- CI konzentriert sich auf die Integration:  
Hauptziel ist die Integration von Codeänderungen in ein gemeinsames Repository und die Sicherstellung, dass bestehende Funktionalitäten nicht beeinträchtigt werden.
- CD konzentriert sich auf Lieferung und Bereitstellung:  
Erweiterung von CI durch Automatisierung des Builds, Testens und Bereitstellens von Anwendungen in verschiedene Umgebungen, letztlich bis hin zur Produktion.

## Aufgabe: Pipeline mit vier Stages

**1. Ziel:** Tiefgehendes CI/CD Verständnis

**2. Schritte:**

- Build wird in die Container Registry gepushed
- Von den nachfolgenden Stages (bei Bedarf) gepulled
- Zwei parallel laufende Tests
- Änderungen am main werden als latest getagged
- Deployment über anwendungs-spezifisches Deploy-Skript

## Pipeline mit vier Stages (... auf vier Slides 😊)

1. Build wird in die Container Registry gepushed
2. Von den nachfolgenden Stages (bei Bedarf) gepulled
3. Zwei parallel laufende Tests
4. Änderungen am main werden als latest getagged
5. Deployment über anwendungs-spezifisches Deploy-Skript

## **.gitlab-ci.yml (1/4)**

default:

image: docker:20.10.16

services:

- docker:20.10.16-dind

before\_script:

- docker login -u \$CI\_REGISTRY\_USER -p \$CI\_REGISTRY\_PASSWORD  
\$CI\_REGISTRY

stages: # Vier Stages

- build
- test
- release
- deploy



## `.gitlab-ci.yml` (2/4)

variables:

# Use TLS (kein Muss!)

[https://docs.gitlab.com/ee/ci/docker/using\\_docker\\_build.html#tls-enabled](https://docs.gitlab.com/ee/ci/docker/using_docker_build.html#tls-enabled)

DOCKER\_HOST: tcp://docker:2376

DOCKER\_TLS\_CERTDIR: "/certs"

CONTAINER\_TEST\_IMAGE: \$CI\_REGISTRY\_IMAGE:\$CI\_COMMIT\_REF\_SLUG

CONTAINER\_RELEASE\_IMAGE: \$CI\_REGISTRY\_IMAGE:latest # Variable für 4.

build:

stage: build

script:

- docker build --pull -t \$CONTAINER\_TEST\_IMAGE .

- docker push \$CONTAINER\_TEST\_IMAGE # 1. Build wird in die Container Registry gepushed

## **.gitlab-ci.yml (3/4)**

test1:

stage: test

script:

- docker pull \$CONTAINER\_TEST\_IMAGE # 2. Von den nachfolgenden Stages (bei Bedarf) gepulled
- docker run \$CONTAINER\_TEST\_IMAGE /script/to/run/tests # 3. Zwei parallel laufende Tests

test2:

stage: test

script:

- docker pull \$CONTAINER\_TEST\_IMAGE # 2. Von den nachfolgenden Stages (bei Bedarf) gepulled
- docker run \$CONTAINER\_TEST\_IMAGE /script/to/run/another/test # 3. Zwei parallel laufende Tests

## **.gitlab-ci.yml (4/4)**

```
release-image:
  stage: release
  script:
    - docker pull $CONTAINER_TEST_IMAGE # 2. Von den nachfolgenden Stages (bei Bedarf) gepulled
    - docker tag $CONTAINER_TEST_IMAGE $CONTAINER_RELEASE_IMAGE # 4. Änderungen am main werden als latest getagged
    - docker push $CONTAINER_RELEASE_IMAGE
  only:
    - main
```

## # 5. Deployment über anwendungs-spezifisches Deploy-Skript

```
deploy:
  stage: deploy
  script:
    - ./deploy.sh
  only:
    - main
  environment: production
```

## Optional: Komplexe Pipeline mit Docusaurus und npm

### 1. Ziel: Tiefgehendes CI/CD Verständnis

### 2. Schritte

- Folgen Sie den Anweisungen des GitLab-Tutorials:  
[https://docs.gitlab.com/ee/ci/quick\\_start/tutorial.html](https://docs.gitlab.com/ee/ci/quick_start/tutorial.html)
- Beachten Sie die Voraussetzungen
  - Überprüfen Sie, ob Sie es wirklich auf GitLab.com ausführen müssen oder ob Sie die selbst gehostete Instanz nutzen können