



Tag 3: Docker, GitOps, Deployment-Strategien

10.07.2024, Daniel Krämer

© Copyright 2024 anderScore GmbH

HECKER
CONSULTING

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

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

Abschlussübung & Diskussion

Inhalt

- Key Takeaways
- Beispiel: Pipeline mit vier Stages
- Abschlussübung: komplexe Pipeline mit Docusaurus und npm

Key Takeaways

- Git-Workflows
- Versionierungsstrategie für Images
- Continuous Integration (CI)
- Continuous Delivery (CD)
- Unterschiede CI und CD

Git-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 (remote) 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 Practices

- 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

CI Best Practices

- 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 Practices

- 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

CD Best Practices

- 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.

Beispiel: Pipeline mit vier Stages

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 anwendungsspezifisches Skript

.gitlab-ci.yml (1/4)

default:

image: docker:20.10.16

services:

- name: docker:20.10.16-dind
alias: docker

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

https://docs.gitlab.com/ee/ci/docker/using_docker_build.html#tls-enabled

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 .

1. Build wird in die Container Registry gepushed

- docker push \$CONTAINER_TEST_IMAGE

.gitlab-ci.yml (3/4)

test1:

stage: test

script:

2. Von den nachfolgenden Stages (bei Bedarf) gepulled

- docker pull \$CONTAINER_TEST_IMAGE
- echo 'Testing container in test 1 ' # 3. Zwei parallel laufende Tests
- docker run -itd -p 9090:9090 \$CONTAINER_TEST_IMAGE

test2:

stage: test

script:

2. Von den nachfolgenden Stages (bei Bedarf) gepulled

- docker pull \$CONTAINER_TEST_IMAGE
- echo 'Testing container in test 2' # 3. Zwei parallel laufende Tests
- docker run -itd -p 9091:9090 \$CONTAINER_TEST_IMAGE

.gitlab-ci.yml (4/4)

release-image:

stage: release

script:

2. Von den nachfolgenden Stages (bei Bedarf) gepulled

- docker pull \$CONTAINER_TEST_IMAGE

4. Änderungen am main werden als latest getagged

- docker tag \$CONTAINER_TEST_IMAGE \$CONTAINER_RELEASE_IMAGE

- docker push \$CONTAINER_RELEASE_IMAGE

only:

- main

5. Deployment über anwendungsspezifisches Deploy-Skript

deploy:

stage: deploy

script:

- chmod +x ./deploy.sh

- ./deploy.sh

only:

- main

environment: production

Abschlussübung: 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
- Beachten Sie die Voraussetzungen