





## DevOps

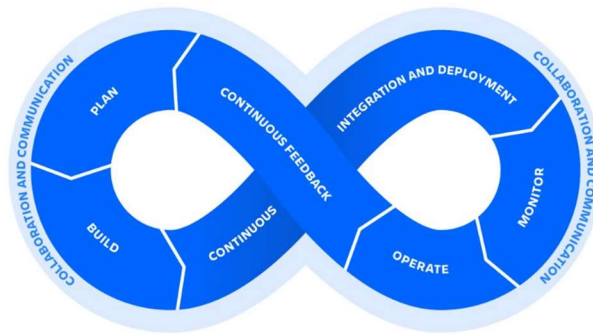
- 
- DevOps setzt sich aus den Wörtern Development und IT Operations zusammen
  - DevOps ist eine Sammlung unterschiedlicher Methoden und Praktiken um eine effiziente Zusammenarbeit eines Softwareentwicklungsteams zu ermöglichen
  - Softwareentwicklungsteams sind dabei im Speziellen sowohl Softwareentwickler (Dev) als auch Systemadministratoren (Ops) (und QM usw.)
  - Mit DevOps soll die Softwarequalität und die Geschwindigkeit in der Auslieferung steigen → Unterstützt den Gedanken der agilen Entwicklungsmethodik



## DevOps

- 
- Entwicklungs- und Operationsteam sind nicht mehr voneinander getrennt
  - Es werden Tools verwendet, um automatisierte Prozesse zu beschleunigen
  - Bei DevOps handelt es sich um einen fortlaufenden und sich wiederholenden Prozess. Deswegen ist der DevOps Lebenszyklus als Endlosschleife dargestellt

## DevOps Lebenszyklus



## DevOps

- DevOps besteht aus 6 Phasen:
  - Planen – agil, Zerlegung der Aufgaben
  - Erstellen - Sourcecontrol
  - **Continues Integration and Continous Delivery/Deployment – automatisierte Workflows**
  - Überwachen und Warnen – Identifizieren und Beheben von Problemen
  - Operate – Verwaltung der Bereitstellungen
  - Fortlaufendes Feedback – Kundenrückmeldung für die Verbesserung von Releases
- Auf der linken Seite des Lebenszyklus sieht man die Bereiche für Entwicklung, auf der rechten Seite, die für Operations
- Für jeden Bereich gibt es DevOps-Tools, die bei der Umsetzung des Lebenszyklus unterstützen

## DevOps Vorteile

- Geschwindigkeit – hochwertigere und stabilere Releases → weniger Bugs
- Bessere Zusammenarbeit – Verantwortungen werden geteilt und Aufgaben kombiniert
- Schnelleres Deployment – dadurch können Releases schneller verbessert werden
- Qualität – CI/CD stellen sicher, dass Änderungen funktionsfähig sind
- Sicherheit – mit Möglichkeiten in CI/CD Sicherheitsfunktionen einzubauen, um die eigene Software vor Angriffen während eines build-Prozesses zu schützen, machen den ganzen Entwicklungsprozess sicherer

## Continues Integration / Continuous Delivery – Continous Deployment (CI/CD)

- CI/CD bedeutet Software „nach jeder Änderung“ auf Funktionsfähigkeit zu überprüfen
- In CI/CD können nicht nur Build- und Testprozesse automatisiert werden, sondern auch automatisiert Dokumentation erstellt werden
- **GitLab** oder auch Jira sind Tools, die CI/CD ermöglichen
- Um Testprozesse automatisieren zu können, müssen Unit Tests erstellt werden
- UnitTests sind eigenständige Module, die Funktionen einer Software testen.

# CI/CD in GitLab

Mit CI/CD kann man seine Software automatisch „builden“, testen und liefern

Dieser Automatismus wird durch eine sogenannte „Pipeline“ realisiert. Eine Pipeline ist ein spezielles File mit dem Namen **.gitlab-ci.yml**



## .gitlab-ci.yml (Pipeline)

- Das **.gitlab-ci.yml** File muss sich im Rootverzeichnis des Projektes in Git befinden
- In diesem File sind sequentiell die Schritte aufgelistet, um die erstellte Software zu bauen, testen und zu liefern
- Eine Pipeline besteht aus 2 Komponenten:
  - Einem „Job“ – dieser beschreibt, was passieren soll
  - Einem „Stage“ – dieser beschreibt in welcher Reihenfolge Jobs erledigt werden müssen
- Eine Pipeline ist im Prinzip eine Folge von Anweisungen für ein Programm, die auszuführen sind
- Das Programm, dass diese Anweisungen ausführt nennt sich **GitLab Runner**

# GitLab Runner

- Ist ein eigenständiges Programm im GitLab Umfeld
- Diesen Runner kann man lokal laufen lassen, in einer VM oder auch in einem **Container**
- Zur Laufzeit weist GitLab Jobs einem Runner zu
- In GitLab gibt es die Möglichkeit auch „globale Runner“ zu verwenden. Solange Runner zur Verfügung stehen, muss man keine eigenen Runner kreieren
- Ohne Runner werden keine Jobs ausgeführt

# Runner erstellen

1. # Erstelle einen Ordner, zum Beispiel: &GitLab-Runner
2. # Wechsle zum neuen Ordner  
`cd 'C:\GitLab-Runner'`
3. # Downloade die Binaries  
`Invoke-WebRequest -Uri "https://gitlabrunner-downloads.s3.amazonaws.com/latest/binaries/gitlab-runner-windows-amd64.exe" -OutFile "gitlab-runner.exe"`, oder öffne nur den Link im Browser, nach dem Download benenne den runner um in gitlabrunner und kopiere die Datei in ein Verzeichnis deiner Wahl
4. # Starte den runner  
`.\gitlab-runner.exe install`  
`.\gitlab-runner.exe start`
5. # Registriere den Runner  
Gehe in deinem Projekt auf GitLab zu CI/CD dann klappe „Runners auf“ dort sind unter „Project Runners“ darunter 3 Punkte dort klicken und „Show runner installation...“ wählen. Danach den „register“ Befehl kopieren und im Terminal ausführen

# Runner erstellen

## Available specific runners

#14865667 (TjVeNVTU)

Remove runner

Runner WIFI Kurs

Runner

Dieser Runner „runnt“ noch nicht, damit dieser läuft, muss man folgenden Befehl eingeben: `.\gitlab-runner.exe run`

## Available specific runners

#14865667 (TjVeNVTU)

Remove runner

Runner WIFI Kurs

Runner

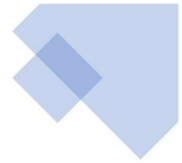
Den bei der Erstellung des Runners angegebenen Tag kann man nun auch in seiner Pipeline bei den Jobs angeben – z.B.: tags: - WIFI

## Unterschied Shell vs. Docker

- Wählt man beim Runner die shell als Executor, werden nur lokale Befehle ausgeführt, die das builden/testen (oder sonstiges) eines Projektes betreffen
- Das Problem dabei ist, dass in vielen Sprachen/Frameworks dafür bei Entwicklern verschiedene Pfade für Compiler und ausführbaren Code existieren
- Neue Entwickler brauchen dann auch immer die gleiche Entwicklungsumgebung wie die Kollegen, um ein existierendes Projekt lauffähig zu bekommen
- Die Lösung → Docker statt lokal!



## Erstellung einer lokalen Pipeline



- Mit dem erstellten Shell Runner kann man nun eine Pipeline lokal starten:

```
stages:
  - build
  - test
build:
  stage: build
  script:
    javac -d target -cp
      D:\EigeneDateien\JavaLibs\junit-platform-console-
      standalone-1.9.0.jar "src\TestCalculate.java"
  tags:
    - "Calculator"
unit-test-job: # This job runs in the test stage.
  stage: test
  script:
    - java -jar D:\EigeneDateien\JavaLibs\junit-platform-
      console-standalone-1.9.0.jar --class-path src/target -
      -select-class TestCalculate
  tags:
    - "Calculator"
```

