Woche 6

CI / CD - Pipeline Testautomatisierung

Modul 324

CI - Continious Integration

build - des Programmcodes

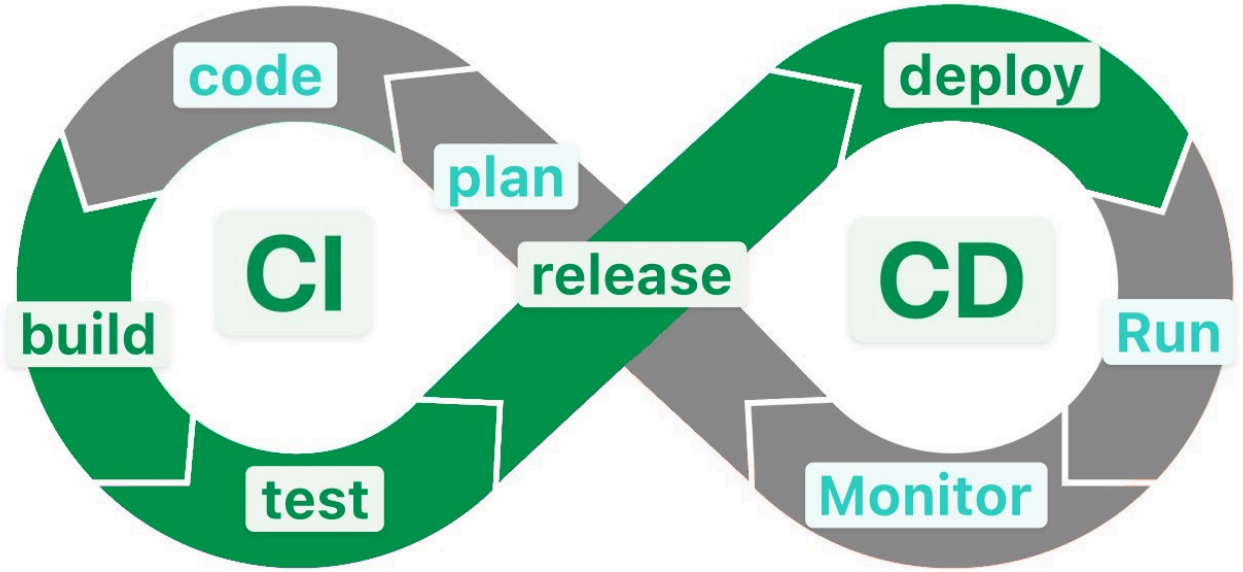
`npm run build` , `mvn build`

test - des Programmcodes

`npm run test` , `mvn test`

release - erstellt das docker images

`docker build`



CD - Continious Delivery

deploy - docker images nach AWS

`kamal deploy` oder andere

🧰 Tools

CI (und CD)

- GitHub Actions
- GitLab CI
- Jenkins

CD only

- ArgoCD

GitHub Actions

- Führen anhand von Events, Workflows aus
- Workflows sind Scripts welche auf GitHub, in Container ausgeführt werden
- GitHub Action Workflows befinden sich im Ordner `.github/workflows`
- Jede Datei mit der Endung `*.yml` wird als Workflow ausgeführt
- In unserer Applikation sind das die Dateien
  - `.github/workflows/deploy.yml`
  - `.github/workflows/aws-infrastructure.yml`
  - `.github/workflows/release-please.yml`
- Die Workflows folgen dieser Syntax

📺 Erklärt auf Youtube

GitHub Actions: Syntax

- Die Sprache von Github Actions ist YAML
  - YAML definiert nur die Struktur und nicht den Inhalt
  - 📺 Erklärt auf youtube
- GitHub Actions definiert **eigene Keywörter**, welche eine Bedeutung haben.
- Die allgemeine 📜 Syntax ist auf Github dokumentiert

GitHub Actions: Basic file

```yaml
name: GitHub Actions Demo
on: [push]
jobs:
  Explore-GitHub-Actions:
    runs-on: ubuntu-latest
    steps:
      - run: echo "🎉 Hallo ${{ github.event_name }} event."
      - run: echo "🐧 This job is now running on a ${{ runner.os }}"
      - run: echo "🔎 The branch ${{ github.ref }} in ${{ github.repository }}."
      - name: Check out repository code
        uses: actions/checkout@v4
      - run: echo "💡 clone the repository ${{ github.repository }}"
      - run:
          echo "🖥️ The workflow is now ready to test your code on the runner."
      - name: List files in the repository
        run: ls ${{ github.workspace }}
```

GitHub Actions: Kontext

- Es gibt viele Kontext-Variablen in GitHub Actions
- Sie werden in doppelt geschweiften Klammen geschrieben
  `${{ <Context>.<Variable> }}`

Wichtigste Kontexte

- **github** beinhalten Variablen über den Workflow und die Events die ihn getriggert haben.
- **secrets** beinhalten Environment Variablen die als Secret erstellt wurden.
- **env** beinhalten Environment Variablen die nicht vertraulich sind.

Die Datei `deploy.yml`

- In eurem Repo existiert bereits die Datei `deploy.yml`
- Dieser wird beim `push` auf den branch `main` ausgeführt
- Es existiert einen Job `deploy` mit folgenden Steps:

- Checkout
- Configure AWS credentials
- Get Server Ip
- Set up Ruby for Kamal
- Login to Amazon Elastic Container Registry
- Push environment variables
- Set up Docker Buildx
- Docker meta
- Build and push
- **Kamal deploy image**

Wo builden wir nun den Code?

Im Dockerfile!

📝 Aufgabe 1: App auf AWS deployen

- Finalisiert das Dockerfile mit den Anforderungen:
  - Muss den **TCP Port 3000** exposen
  - Muss auf dem Port 3000 einen **Webserver** serven.
  - Muss auf dem Port 3000 eine Route **/up** besitzen die ein Status 200 OK zurückgibt.
- Kopiere die AWS credentials in GitHub Repository
- Nun sollte die App auf AWS deployed werden!

Testautomatisierung

Qualitätssicherung

- Soll sicher stellen, dass das Produkt auch den Erwartungen entspricht
- Die **funktionale Erwartung** soll durch das Einbinden des Kunden und schnellem Feedback der umgesetzten Features gesichert werden
- Die **Fehlerfreiheit** ist zu einem gewissen Grad automatisiert testbar.

Unit-Tests

- Gewährleisten, dass **eine Methode** korrekte Resultate liefert
- Methoden werden mit verschiedensten Argumenten aufgerufen und geprüft ob das Resultat stimmt

Frameworks (JavaScript)

- Jasimne / Karma
- Jest
- MochaJs

Integration-Tests

- Testen das Zusammenspiel von Methoden
- Es wird die **Benutzerinteraktion nachempfunden**
- Dafür werden "Headless Browser" verwendet.

Frameworks (Sprachunabhängig!)

- Selenium
- Gauge / Taiko
- Cypress
- Cucumber (die Erfinder...)

Test Driven Development (TDD)

> *Make it green, then clean*

- Es wird zuerst einen Test geschrieben anhand der Anforderungen
- Es wird programmiert bis dieser Test grün ist
- Die Tests dienen direkt als Dokumentation, welche Features vorhanden sind.

> *Viele Tests können aber auch hinderlich sein. "Oh nein, dann muss ich all die wieder anfassen..."*

Best Practices

- Test schreiben, wenn ein Fehler auftritt
- Für komplexe Berechnungen immer einen Test schreiben
- Ein Integration-Tests Framework einführen und damit testen

Aber Achtung!

> *There are known knowns. These are things we know that we know. There are known unknowns. That is to say, there are things that we know we don't know. But there are also unknown unknowns.* ***There are things we don't know we don't know****.*
>
> *-- Donald Rumsfeld, US-Amerikanischer Politiker*

> *Program testing can be used to show the presence of bugs, but never to show their absence!*
>
> *-- Dijkstra*

📝 Aufgabe 2: Test step einbauen

- Erstellt **einen** automatisierten Test für euer Framework
  - Angular: https://angular.dev/guide/testing
  - Spring Boot: https://www.baeldung.com/spring-boot-testing
- Führt den Test in der GitHub Action aus (siehe Anleitung auf der Webseite)
- So wird sichergestellt, dass die Tests grün sind, sonst bricht das deployment ab