Spickzettel: GitHub Actions & Docker

1. Docker in GitHub Actions nutzen

- GitHub Actions ermöglicht das **Erstellen, Testen und Veröffentlichen von Docker-Containern**.
- Läuft entweder auf **GitHub-hosted Runnern** oder **Self-hosted Runnern** mit Docker.

2. Docker-Container in einem Workflow nutzen

Container-Umgebung für einen Job definieren

- Ein Job kann direkt in einem Docker-Container ausgeführt werden.
- Dies ermöglicht das Arbeiten in einer isolierten Umgebung mit vordefinierten Abhängigkeiten.

```
jobs:
  build:
    runs-on: ubuntu-latest
    container:
      image: node:16  # Job läuft in einem Node.js-Container
      options: --memory=4g --cpus=2  # Ressourcenbegrenzung
    steps:
      - name: Repository auschecken
        uses: actions/checkout@v3
      - name: Abhängigkeiten installieren
        run: npm install
      - name: Tests ausführen
        run: npm test
```

3. Docker-Images in GitHub Actions bauen & veröffentlichen

Docker Image erstellen und in GitHub Container Registry pushen

- **GitHub Container Registry (GHCR)** speichert Docker-Images sicher und versioniert.
- Authentifizierung erfolgt über `GITHUB_TOKEN`.

```
jobs:
  docker-build:
    runs-on: ubuntu-latest
    steps:
      - name: Repository auschecken
        uses: actions/checkout@v3

      - name: Docker Login bei GitHub Container Registry
        run: echo "${{ secrets.GITHUB_TOKEN }}" | docker login ghcr.io -u $
{{ github.actor }} --password-stdin

      - name: Docker Image bauen
        run: |
```

```
          docker build -t ghcr.io/${{ github.actor }}/image-name:latest .
          docker tag ghcr.io/${{ github.actor }}/image-name:latest
ghcr.io/${{ github.actor }}/image-name:v1.0

      - name: Docker Image pushen
        run: |
          docker push ghcr.io/${{ github.actor }}/image-name:latest
          docker push ghcr.io/${{ github.actor }}/image-name:v1.0
```

4. Docker-Container als Service nutzen

- In GitHub Actions können **zusätzliche Dienste (Services)** als Container gestartet werden.
- Nützlich für **Datenbanken, Caching-Systeme, API-Server** usw.

Beispiel: PostgreSQL-Container für Tests starten

```
jobs:
  test:
    runs-on: ubuntu-latest
    services:
      postgres:
        image: postgres:13
        env:
          POSTGRES_USER: user
          POSTGRES_PASSWORD: password
          POSTGRES_DB: testdb
        ports:
          - 5432:5432
        options: --health-cmd "pg_isready -U user" --health-interval=10s --
health-timeout=5s --health-retries=3
    steps:
      - name: Repository auschecken
        uses: actions/checkout@v3

      - name: Warte auf PostgreSQL-Start
        run: sleep 10

      - name: Abhängigkeiten installieren
        run: npm install

      - name: Tests mit PostgreSQL ausführen
        env:
          DATABASE_URL: "postgres://user:password@localhost:5432/testdb"
        run: npm test
```

- **Services werden innerhalb des Workflows gestartet und sind über `localhost` erreichbar.**
- **`options`** ermöglicht erweiterte Docker-Parameter wie **Health-Checks**.
- **Umgebungsvariablen** konfigurieren den Container für den Workflow.

5. Eigene Docker-basierte GitHub Action erstellen

1. **Projektverzeichnis erstellen**:

```
mkdir my-action && cd my-action
```

2. **`Dockerfile` für die Action anlegen**:

```
FROM node:16
WORKDIR /app
COPY entrypoint.sh /entrypoint.sh
RUN chmod +x /entrypoint.sh
ENTRYPOINT ["/entrypoint.sh"]
```

3. **Action-Metadaten (`action.yml`) definieren**:

```
name: "Meine Docker Action"
description: "Eine benutzerdefinierte GitHub Action mit Docker"
runs:
  using: "docker"
  image: "Dockerfile"
```

4. **In einem Workflow verwenden**:

```
jobs:
  custom-action:
    runs-on: ubuntu-latest
    steps:
      - name: Eigene Docker Action ausführen
        uses: USERNAME/my-action@main
```

Best Practices

- **`actions/cache` nutzen**, um Docker-Images zu beschleunigen.
- **GitHub Container Registry (`ghcr.io`)** für sichere Image-Verwaltung verwenden.
- **Nur notwendige Abhängigkeiten in den Container einbauen**, um Image-Größe zu minimieren.
- **Multi-Stage Builds** für effizientere Docker-Images nutzen.