Deployment

Ziel dieser Aufgabe ist des das CRUD-Assignment in eine einizge Docker-Compose Datei zu packen um diese von überall aus ausführbar zu machen (deployment). Damit wäre es im weiteren auch möglich diese Datei auf einer Azure-Instanz o.ä hochzuladen und von dort aus auszuführen. Auch Interessant ist diese Art des Deployments für IT-Abteilungen von Firmen mit denen man zusammenarbeitet. Die freuen sich meistens wenn man Ihnen durch so etwas die Arbeit etwas abnimmt und man ist bei (kritischen) Updates nicht auf sie angewiesen, vorausgesetzt es wird immerwieder die neueste Version aus dem Registry gepullt (`docker pull <imagename>`).

Dazu braucht es folgende Voraussetzungen:

- Dockerfiles
- Images
- Container-Registry
- CI/CD Pipelines
- Je ein Workflow für Frontend und Backend
- Da Frontend und Backend üblicherweise in 2 Repos aufgeteilt ist kann man einfach in jeden Repo einen neuen Workflow hinzufügen (bereits exisiterende Repos können ggf. weiterverwendet werden)

Das Dockerfile fürs Frontend könnte z.B so aussehen:

```
# Use a Node.js base image
FROM node:16-alpine AS builder

# Set the working directory to /app
WORKDIR /app

# Copy package.json and package-lock.json files to the container
COPY package*.json ./

# Install dependencies via ci
RUN npm ci

# Copy the rest of the application code to the container
COPY . .

# Build the application and output the files to /dist
RUN npm run build --prod

# Use a Nginx base image for the final container
FROM nginx:1.23-alpine

# Copy the /dist directory from the builder container to the final container
COPY --from=builder /app/dist/angular-request-assignment /usr/share/nginx/html

# Expose port 80 for the web server
EXPOSE 80
```

```
# Start the Nginx web server
ENTRYPOINT ["nginx", "-g", "daemon off;"]
```

Füge (falls nicht schon vorhanden) sowohl im Frontend als auch im Backend ein Dockerfile hinzu. Im übergeordneten Ordner kann sobald beide Dateien Images builden ein docker-compose mit folgender Struktur angelegt werden:

```
version: "xy"

services:
   frontend:
       build: ./frontend
   backend:
       build: ./backend
```

Wenn das funktioniert ist schon ein guter Teil erledigt, jedoch setzt dieses Dockerfile voraus das der Programmcode in den Unterordnern vorliegt. Für einen Entwickler ist das kein Problem, aber wenn man das Programm dann beim Auftraggeber installieren möchte, möchte man dem im Normalfall nicht den Code geben. Deswegen ist es besser wenn man die Images in einer `Container-Registry` hochlädt um von überall aus auf diese zugreifen zu können. Dann schaut das Compose ca. so aus:

```
version: '3'

services:
  frontend:
    image: ghcr.io/<your-username>/<your-reponame>:latest
    ports:
      - "80:80"
```

Um dieses Repository nutzen zu können muss man sich jedoch mit Docker bei ghcr (Github Container Registry) anmelden. Das funktioniert folgendermaßen:

- Lege dir einen Personal Access Token an, dieser Braucht nur die Berechtigung `read_package`

> Siehe working with container registries und create a personal access token

- Wenn du einen PAT hast kannst du dich mit dem Befehl `docker login` anmelden
    - Username: Dein Github-Username
    - Passwort: Dein PAT (Personal Access Token)

Aufbau des Image-Feldes

- ghcr.io Github container registry
- Dein Username auf Github
- Der Name des Images der beim CI/CD wird meistens über einen Tag in Git festgelegt

Zum Vergleich: Ein Image weit verbreiteter Software sieht oft so aus `mysql`. Dabei wurden für einige der obigen Felder Standardannahmen getroffen. In voller Länge würde es so aussehen: `docker.io/libraries/mysql:latest`. Alles bis auf `mysql` sind Standardparameter.

> Siehe: Official Docker Images

Einrichten der CI/CD

Die Containerimages kann man entweder Lokal builden und ins Registry hochladen oder sich auf den Github-Servern bauen lassen. Bei den Github-Servern ist der Vorteil dass man sicherstellen kann das vor dem Buildprozess Unittests, Sicherheitschecks etc. durchgeführt werden.

Um die Github Server anzuweisen ist folgende Ordnerstruktur im Repository zu erstellen: `.github/workflows/<docker_build_actionname.yml>`

Die Datei könnte ca. so aussehen:

```
name: Build and Push Docker Image
on:
  # trigger execution on every push
  push:
    branches: [ main ]
  # trigger execution on every tag
  #tag:

jobs:
  build-and-push:
    runs-on: ubuntu-latest
    steps:
    - name: Checkout repository
      uses: actions/checkout@v2

    - name: Set up Docker Buildx
      uses: docker/setup-buildx-action@v1
    - name: Login to GitHub Container Registry
      uses: docker/login-action@v1
      with:
        registry: ghcr.io
        username: ${{ github.actor }}
        password: ${{ secrets.GITHUB_TOKEN }}
    - name: Build and push Docker image
      env:
        DOCKER_BUILDKIT: 1
      run: |
        docker buildx build \
          --progress plain \
          --tag ghcr.io/<your-username/orgname>/<your_reponame>:latest (or
${{github.tag}} when pushed on tag ) \
          --push .
```

> Einführung in Container Registries in Github

Bewertungsgrundlagen bei der Abgabe

Füge jene docker-compose.yml Datei in das Repository der aktuellen Abgabe hinzu welche auf die von dir erstellten Images verweist.

Abzugeben ist:

- eine Docker-Compose Datei mit Images
- ggf. Logindaten um Zugriff auf die Lokalen Container zu erhalten
    - NICHT das eigene Passwort sondern nur einen Zugangstoken für die Registries

Punkteverteilung

- Nach einem Docker-Login von meiner Seite mit eurem Token kann ich die Images pullen 33P
- Die Images können ausgeführt werden ohne sofort abzustürzen 33P
- Bei den Tags gibt es sowohl eine aktuelle Version vX.Y als auch einen `latest`-Tag für die neueste Version 25P
- Beim Build eines neuen Images gibt es eine Schritt (Step) für das Ausführen von Unittests 33P

Weitere Unterlagen

> Docker Images manuell pushen

> Container Registries in Gitlab