Wartungshandbuch

DSE-2021S Gruppe 08 - Eder, Eidelpes und Zeisler

1. Beschreibung der Entwicklungsumgebung

Als Entwicklungsumgebung kam für die Code Entwicklung PyCharm 2021 für das Backend und Webstorm 2021 für das Frontend zum Einsatz. Beide Entwicklungsumgebungen haben eine direkte Anbindung an Docker. Für lokale Test-Deployments wurde minikube verwendet. Die Dockerfiles und die docker-compose Datei erlauben das schnelle lokale Ausführen der Applikation.

2. Beschreibung der eingesetzten Frameworks und Libraries

2. 1 Flask

Flask ist ein Webframework für Python. Der Hersteller bezeichnet es aufgrund seiner Einfachheit auch als Microframework. Im Projekt wird Flask hauptsächlich zur Bereitstellung der REST APIs verwendet.

2.2 Angular

Angular ist ein weit verbreitetes Webapplikations Frontend-Framework basierend auf Typescript. Für das Projekt kommt es zur visualisierung des Control Centers zum Einsatz. Im Hintergrund werden API Aufrufe an das X-Way Service durchgeführt und die Daten auf der Karte aktualisiert.

2.3 flask_restx

Die Flask Erweiterung flask_restx stellt die Swagger Dokumentation des X-Way Service zur verfügung. Mit bestimmten Dekoratoren sind die einzelnen Schnittstellen beschrieben.

2.4 Pymongo

Für den Zugriff auf die Persistenzschicht (in diesem Fall Mongodb) kommt Pymongo zum Einsatz. Das Entity Ident Service speichert alle relevanten statischen Daten der Entitäten (Ampeln und Fahrzeuge) und nutzt als interne Schnittstelle zur Datenbank Pymongo.

2.5 StrictRedis

StrictRedis ist ein Python Client für Redis. Das Event Store Service nutzt den Client um Ereignisse für Ampeln und Fahrzeuge abzulegen und für andere Services lesbar zu machen.

2.6 Pika

Pika ist ein Python Client für RabbitMQ. In einer shared library wurde ein RabbitMQ wrapper implementiert, der unter der Verwendung von pika ein einfaches Framework zum Aufbauen der RabbitMQ connection und zum senden und empfangen von Daten über Queues bereitstellt.

2.7 AgMap

Um Google Maps in Angular zu nutzen wird die Library AgMap eingesetzt. AgMap kümmert sich um die Darstellung der Karte, sowie um die einzelnen Marker, die genutzt werden um Fahrzeuge und Ampeln zu visualisieren.

3. Beschreibung des Build Prozesses

In jedem Microservice gibt es ein Dockerfile. Um das entsprechende Image zu bauen muss *docker build* ausgeführt werden. Die Microservices können auch mithilfe der docker-compose Datei erstellt und ausgeführt werden. Der Befehl *docker-compose build* erstellt alle Images. Nach der Erstellung können mit *docker-compose up* alle Services gestartet werden. Damit die aktuellsten Images für Kubernetes zur Verfügung stehen, müssen die Images mit *docker-compose push* in die Google Container Registry gepusht werden. Dies geht nur mit den nötigen Authentifizierungsdaten.

4. Beschreibung des Testprozesses

Alle Backend Microservice Komponenten mit ausreichender Logik werden durch mindestens drei Tests abgedeckt. Technisch werden Unittests von Python für die Durchführung verwendet. Das Frontend wird aufgrund fehlender Komplexität nicht getestet, da die Hauptfunktionalität von der AgMap Bibliothek zur Verfügung gestellt wird..

5. Beschreibung des Deployments

Im Projekt-Verzeichnis *kubernetes* gibt es im Ordner *deployments* für jedes Microservice eine YAML Datei. Analog dazu im Ordner *services* je eine Datei pro Microservice. Um ein Microservice zu deployen muss folgender Befehl ausgeführt werden:

kubectl apply -f deployments/%SERVICE_NAME%-deployment.yaml

Zusätzlich muss die dazugehörige Service YAML Datei angewandt werden:

kubectl apply -f deployments/%SERVICE_NAME%-deployment.yaml

Für den externen Zugriff kann ein Ingress erstellt werden:

kubectl apply -f ingress/dse2021-ingress.yaml

Damit die Services untereinander kommunizieren können ist es notwendig, zuerst die Services zu erstellen und dann die Deployments zu starten. Damit die Pods die richtigen Umgebungsvariablen bekommen, muss zuerst die ConfigMap im Ordner *configmaps* erstellt werden.

Der Ingress *dse2021-ingress* verwendet *nginx-ingress* um sowohl das Frontend als auch das Crossway Microservice erreichbar zu schalten. *Nginx-ingress* wurde per Helm Repository mit den folgenden Befehlen installiert:

helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
helm install nginx-ingress ingress-nginx/ingress-nginx

Die Microservices kommunizieren untereinander mit den Hostnames, die von Docker/Kubernetes vergeben werden (also *xway, entityident, ...*).

Abbildung 1 zeigt das Deployment Diagramm des Systems. Die dependencies sind vertikal aufzulösen. Jedes <<device>> entspricht einem eigenen Computing Node. Das <<execution environment>> sind verschiedene Docker Container.

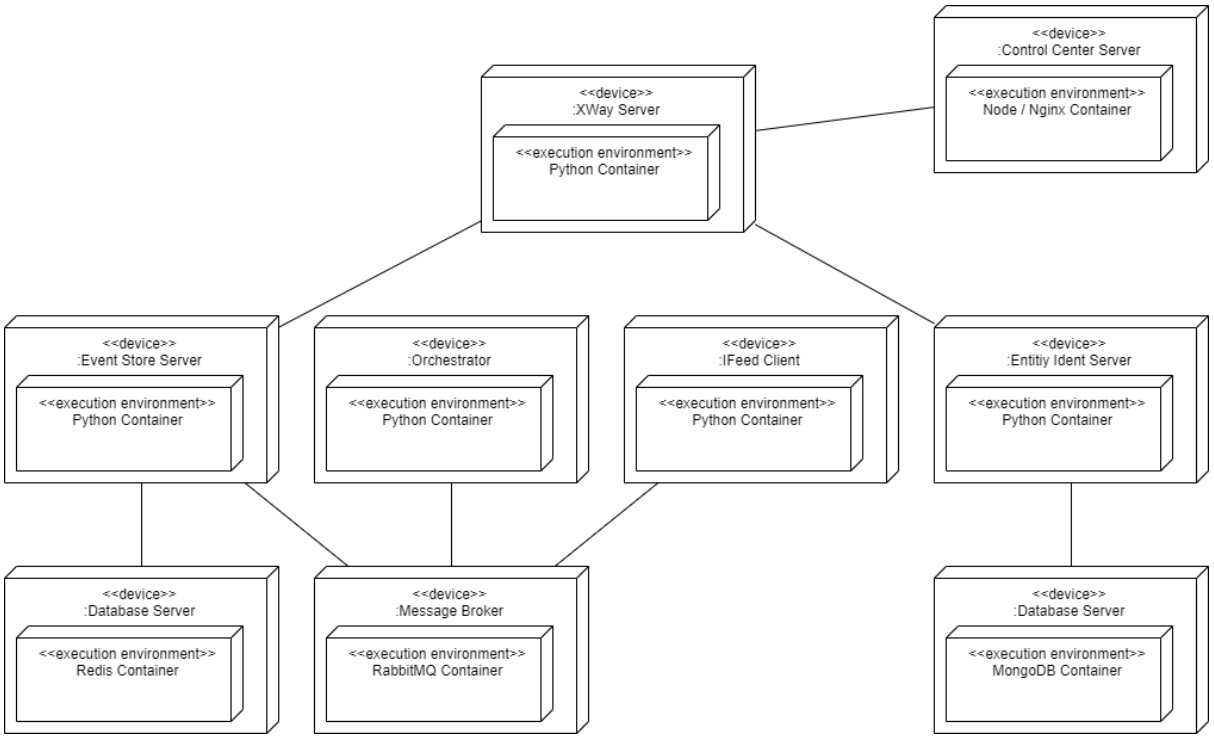

Abbildung 1 - Deployment Diagramm

6. Dokumentation des API Gateway APIs mittels Swagger (https://swagger.io) oder einem vergleichbaren API-Dokumentations-Framework

Detaillierte Dokumentation befindet sich am X-Way: http://xway/api/v1/