DSE22Gruppe10-Wartungshandbuch

Diego Krupitza, Jan Müller, Kian Pouresmaeil

26. Mai 2022

Inhaltsverzeichnis

1 Entwicklungsumgebung

Für das Entwickeln der Microservices werden Java 17, Maven 3.8.5, Node.js 16.15.0 sowie Yarn 1.22.18 oder neuer benötigt. Bevor man die Entwicklungsumgebung starten kann, müssen eine Datenbank sowie eine „MoM“ mithilfe von Docker Compose gestartet werden. Dazu kann der Befehl `docker-compose up -d` im Hauptverzeichnis des Projektes ausgeführt werden.

Es gibt keine strikte Reihenfolge für das Starten der Microservices. Die folgende Reihenfolge bietet sich jedoch an, um die Wartezeit zu verringern.

1. Entity-Service
2. Tracking-Service
3. Flowcontrol-Service
4. Simulator-Service
5. Gateway-Service
6. Cockpit

1.1 Starten der Java Services

Die Services *Flowcontrol-Service*, *Simulator-Service* und *Gateway-Service* können mittels Maven gestartet werden. Hierfür muss im Verzeichnis des gewünschten Dienstes der Befehl `mvn spring-boot:run` ausgeführt werden.

1.2 Starten der Typescript Services

Die TypeScript Services sowie das Cockpit lassen sich über den Befehl `yarn dev` im jeweiligen Verzeichnis starten. Zuvor muss der Befehl `yarn install` verwendet werden, um benötigte Bibliotheken zu installieren.

1.3 Konfiguration der Services

Einige Services der Applikation bieten Anpassungsmöglichkeiten über Umgebungsvariablen, wobei Standardwerte eine Konfiguration optional machen.

1.3.1 Konfiguration Entity-Service

Folgende Umgebungsvariablen sind anpassbar.

Name	Standardwert	Beschreibung
MONGO_DB_HOST	localhost:27017	Adresse der MongoDB Instanz
MONGO_DB_USER	admin	Username der MongoDB Instanz
MONGO_DB_PWD	admin	Passwort der MongoDB Instanz
MONGO_DB_NAME	local-entity-db	Name der MongoDB Datenbank
RABBIT_MQ_HOST	localhost	Adresse RabbitMQ Instanz

Tabelle 1: Environment variablen für Entity-Service

1.3.2 Konfiguration Tracking-Service

Folgende Umgebungsvariablen sind anpassbar.

Name	Standardwert	Beschreibung
MONGO_DB_HOST	localhost:27017	Adresse der MongoDB Instanz
MONGO_DB_USER	admin	Username der MongoDB Instanz
MONGO_DB_PWD	admin	Passwort der MongoDB Instanz
MONGO_DB_NAME	local-entity-db	Name der MongoDB Datenbank
RABBIT_MQ_HOST	localhost	Adresse RabbitMQ Instanz

Tabelle 2: Environment variablen für Tracking-Service

1.3.3 Konfiguration Simulator-Service

Folgende Umgebungsvariablen sind anpassbar.

Name	Standardwert	Beschreibung
ENTITY_SERVICE_IP	localhost	Adresse des Entity-Service
TRACKING_SERVICE_IP	localhost	Adresse des Tracking-Service
FLOW_SERVICE_IP	localhost	Adresse des Flowcontrol-Service
SPRING_RABBITMQ_HOST	localhost	Adresse der RabbitMQ Instanz

Tabelle 3: Environment variablen für Simulator-Service

1.3.4 Konfiguration Flowcontrol-Service

Folgende Umgebungsvariablen sind anpassbar.

1.3.5 Konfiguration Gateway-Service

Folgende Umgebungsvariablen sind anpassbar.

Name	Standardwert	Beschreibung
ENTITY_SERVICE_IP	localhost	Adresse des Entity-Service
TRACKING_SERVICE_IP	localhost	Adresse des Tracking-Service
SIMULATOR_SERVICE_IP	localhost	Adresse des Simulator-Service
SPRING_RABBITMQ_HOST	localhost	Adresse der RabbitMQ Instanz

Tabelle 4: Environment variablen für Simulator-Service

Name	Standardwert	Beschreibung
ENTITY_SERVICE_IP	localhost	Adresse des Entity-Service
TRACKING_SERVICE_IP	localhost	Adresse des Tracking-Service
SIMULATOR_SERVICE_IP	localhost	Adresse des Simulator-Service
FLOW_SERVICE_IP	localhost	Adresse des Flowcontrol-Service

Tabelle 5: Environment variablen für Gateway-Service

2 Frameworks und Bibliotheken

In diesem Kapitel werden die verwendeten Frameworks und Bibliotheken kurz beschrieben. Detaillierte Informationen können den Dokumentationen der jeweiligen Projekte entnommen werden.

2.1 Entity-Service und Tracking-Service

Sowohl der Entity-Service als auch der Tracking-Service wurden mithilfe des Fastify-Frameworks entwickelt. Dabei handelt es sich um ein Web-Framework für Node.js, welches sich durch seine hohe Performanz sowie Schema-Validierung auszeichnet.

Für die Kommunikation mit der MongoDB Datenbank wird die offizielle Bibliothek node-mongodb-native verwendet.

Die Kommunikation mit der „MoM“ basiert auf der Bibliothek amqplib.

Darüber hinaus wird dotenv zum Laden der Umgebungsvariablen verwendet.

2.2 Cockpit

Für die Entwicklung des Frontends wurde eine Vielzahl and Bibliotheken eingesetzt. Die Grundlage bildet das Framework Vue 3. In Kombination mit dem Build-Tool Vite, bietet Vue 3 eine moderne Entwicklungsumgebung und beschleunigt die Umsetzung der Anforderungen durch nützliche Funktionen wie beispielsweise Hot-Reloads in wenigen Millisekunden.

Darüber hinaus wurde FormKit eingesetzt um eine Validierung von Nutzereingaben zu vereinfachen.

Axios wurde als zuverlässiger HTTP-Client in das Frontend eingebunden, um eine Kommunikation mit dem Backend umzusetzen.

Die Bibliothek vue-toastification kommt beim Anzeigen von Statusmelden, beispielsweise beim Starten und Stoppen von Simulationen, zum Einsatz.

Als performantere Alternative zu Tailwind wurde die Bibliothek UnoCSS verwendet.

2.3 Flowcontrol-Service

Für die Entwicklung des Simulator-Services, welches in Java entwickelt wurde, wurde auf das Spring Boot gesetzt sowie Spring Cloud gesetzt. Damit die Kommunikation über REST mit den anderen Services nicht viel unnötigen boilerplate Code braucht, verwenden wir
`org.springframework.cloud : spring-cloud-starter-feign` eine Implementation von OpenFeign. Da dieses Service auch mit RabbitMQ kommuniziert verwenden wir auch die Implementierung von `spring-boot-starter-amqp`. Dies ermöglichte uns sehr schnell mit wenig Code effizient mit der RabbitMQ Instanz zu kommunizieren. Für das Testen wurde JUnit5 und TestContainer verwendet.

2.4 Simulator-Service

Für die Entwicklung des Simulator-Services, welches in Java entwickelt wurde, wurde auf das Spring Boot gesetzt sowie Spring Cloud gesetzt. Für die Entwicklung der Restcontroller wurde unter anderem Spring MVC verwendet. Damit die Kommunikation über REST mit den anderen Services nicht viel unnötigen boilerplate Code braucht, verwenden wir
`org.springframework.cloud : spring-cloud-starter-feign` eine Implementation von OpenFeign. Da dieses Service auch mit RabbitMQ kommuniziert verwenden wir auch die Implementierung von `spring-boot-starter-amqp`. Dies ermöglichte uns sehr schnell mit wenig Code effizient mit der RabbitMQ Instanz zu kommunizieren. Für das Testen wurde JUnit5 verwendet.

2.5 Gateway-Service

Für die Entwicklung des Gateways wurde auf das Spring Boot gesetzt sowie Spring Cloud gesetzt. Für die Umsetzung vom Gateway war insbesondere die Dependency `org.springframework.cloud : spring-cloud-starter-gateway`

sehr von Bedeutung, da dies uns erlaubt hat mit möglichst wenig aufwand und nur durch reine Konfiguration ein sehr gutes Gateway aufzusetzen.

3 Build Process

Die Java-Dienste lassen über den Befehl `mvn package -DskipTests` bauen. Analog dazu baut der Befehl `yarn build` die TypeScript-Dienste.

Letztendlich findet dies nur beim Bauen der Docker-Images statt, für welche pro Verzeichnis ein Dockerfile angelegt wurde. Für eine kürzere Bauzeit wurden die Dockerfiles so programmiert, dass diese mehrere Ebenen verwenden und Dependencies zwischenspeichert. Um das Bauen der Docker-Images weiter zu vereinfachen, können die `dockerize`-Skripte der Datei `package.json` verwendet werden. Der lokale Docker-Nutzer muss dafür Zugriffsrechte auf die Docker Hub Repositories besitzen.

4 Test Process

Für jedes Microservice sind mindestens zwei Modultests implementiert worden.
Die Java basierten Services wurden mithilfe von *JUnit5* entwickelt. Hierbei wurde die Springboot Version 2.6.5 verwendet.
Die Tests bei den Typescript basierten Services nutzen das *Vitest testing framework* mit Version 0.12.6.

4.1 Gateway

Hier wird getestet ob alle Services und deren Status richtig vom Gateway angezeigt werden. Die *Health-Check* Anfragen an die anderen Services wird hier Mithilfe von *Mockito* gemockt. Die Tests können mit dem Befehl *mvn clean test* in der Kommandozeile durchgeführt werden.

4.2 Flowcontrol-Service

Die Tests überprüfen das Berechnen der Geschwindigkeit eines Fahrzeuges in der nähe einer Ampel. Das Ziel der Berechnung ist, dass das Auto so selten wie möglich zu einem totalen Stillstand kommt.
Der Flowcontrol-Service kommuniziert in der Produktionsversion mit einer RabbitMQ Instanz. Damit die Tests unabhängig von der Produktionsinstanz von RabbitMQ laufen kann, kommt hier ein Testcontainer mit Image

rabbitmq:3-management im Einsatz. Zusätzlich wird die Kommunikation mit anderen Microservices mithilfe von *Mockito* gemockt.
Die Tests können mit dem Befehl *mvn clean test* in der Kommandozeile durchgeführt werden.

4.3 Simulator-Service

Das korrekte Verhalten beim Starten und Stoppen einer Simulation wird mit den vorhandenen Integrationstests überprüft. Dabei wird die sowohl die Anfrage zum Service und die internen Anfragen an weiteren Services mithilfe von *Mockito* gemockt.
Die Tests können mit dem Befehl *mvn clean test* in der Kommandozeile durchgeführt werden.

4.4 Entity-Service

Die vorhandenen Integrationstests überprüfen, ob der *car* Endpunkt sich richtig verhält und ob der Service seinen Status über seinen *health* Endpunkt richtig vermitteln kann. Die Anfragen werden mithilfe eines im Test erstellten Testservers verarbeitet.
Die Tests können mit dem Befehl *yarn test* in der Kommandozeile durchgeführt werden.

4.5 Tracking-Service

In dem Service überprüfen die Integration das korrekte Verhalten der *car* und *traffic-light* Endpunkte und ob der Service seinen Status über seinen *health* Endpunkt richtig vermitteln kann. Die Anfragen werden mithilfe eines im Test erstellten Testservers verarbeitet.
Die Tests können mit dem Befehl *yarn test* in der Kommandozeile durchgeführt werden.

5 Deployment

Für das Deployment verwenden wir GKE und unsere Kubernetes YAML Dateien. Die YAML Dateien befinden sich in dem Ordner mit dem Name *kubernetes*. Auf dem Kubernetes Cluster können diese Dateien dann mit dem Befehl `kubectl apply -f .` eingespielt werden. In den folgenden Unterpunkten findet man eine genauere Beschreibung für die 9 YAML Konfigurationsdateien vor.

5.1 secrets-kubernetes.yaml

Für das Deployment der Applikation auf Kubernetes müssen bestimmte sensible Information vorerst als Secrets in den Cluster eingespielt werden. Diese Secrets werden in einer Datei mit dem Namen *secrets-kubernetes.yaml* abgelegt. Da diese Secrets je nach Deployment anders sind, gibt es eine Beispiels Datei mit dem Namen *example-secrets-kubernetes.yaml*.

Das Secret Mapping muss den Namen *dse-secrets-v2i* haben.

In unserem Deployment findet man folgende Secrets.

Name	Beschreibung
`MONGO_DB_HOST`	Der Hostname von der MongoDB Instanz
`MONGO_DB_USER`	Das Username der MongoDB Instanz
`MONGO_DB_PWD`	Das Passwort der MongoDB Instanz

Tabelle 6: Secrests in dem Deployment

5.2 ingress-kubernetes.yaml

Für unsere Applikation haben wir ein Ingress aufgesetzt bei dem Anfragen basierend auf den Prefix des Pfades zu einem der zwei Services weitergeleitet werden. Anfragen deren pfad mit `/api/*` beginnen werden an das *gateway-service* weitergeleitet. Alle anderen Anfragen werden per Standardeinstellung an das *cockpit-service* weitergeleitet.

5.3 gateway-kubernetes.yaml

In dieser YAML Datei findet man die Deploymentkonfiguration des Gateways vor. Es wir ein deployment mit dem Namen *gateway* erstellt welches das Docker Image *deryeger/dse-gateway* benutzt. Zusätzlich werden alle Konfigurationsvariablen die in dem Kapitel *Entwicklungsumgebung* beschrieben worden sind gesetzt. Als nächster wird ein ClusterIP Service mit dem Namen *gateway-service* erstellt welches den passenden Port des Containers zugänglich macht.

5.4 rabbitmq-kubernetes.yaml

In dieser YAML Datei findet man die Deploymentkonfiguration des RabbitMQ vor. Es wir ein deployment mit dem Namen *rabbitmq* erstellt welches das Docker Image *rabbitmq:management-alpine* benutzt. Als nächster wird

ein ClusterIP Service mit dem Namen *rabbitmq-service* erstellt welches den passenden Port des Containers zugänglich macht.

5.5 cockpit-kubernetes.yaml

In dieser YAML Datei findet man die Deploymentkonfiguration des Cockpits vor. Es wir ein deployment mit dem Namen *cockpit* erstellt welches das Docker Image *deryeger/dse-cockpit* benutzt. Als nächster wird ein ClusterIP Service mit dem Namen *cockpit-service* erstellt welches den passenden Port des Containers zugänglich macht.

5.6 entity-kubernetes.yaml

In dieser YAML Datei findet man die Deploymentkonfiguration des Entity-Services vor. Es wir ein deployment mit dem Namen *entity* erstellt welches das Docker Image *deryeger/dse-entity* benutzt. Zusätzlich werden alle Konfigurationsvariablen die in dem Kapitel *Entwicklungsumgebung* beschrieben worden sind gesetzt. Bei dem Setzen der Konfigurationsvariablen werden unter anderem die Werte aus dem Secret mit dem Namen *dse-secrets-v2i* genutzt. Als nächster wird ein ClusterIP Service mit dem Namen *entity-service* erstellt welches den passenden Port des Containers zugänglich macht.

5.7 simulator-kubernetes.yaml

In dieser YAML Datei findet man die Deploymentkonfiguration des Gateways vor. Es wir ein deployment mit dem Namen *simulator* erstellt welches das Docker Image *deryeger/dse-simulator* benutzt. Zusätzlich werden alle Konfigurationsvariablen die in dem Kapitel *Entwicklungsumgebung* beschrieben worden sind gesetzt. Als nächster wird ein ClusterIP Service mit dem Namen *simulator-service* erstellt welches den passenden Port des Containers zugänglich macht.

5.8 flowcontrol-kubernetes.yaml

In dieser YAML Datei findet man die Deploymentkonfiguration des Gateways vor. Es wir ein deployment mit dem Namen *flowcontrol* erstellt welches das Docker Image *deryeger/dse-flowcontrol* benutzt. Zusätzlich werden alle Konfigurationsvariablen die in dem Kapitel *Entwicklungsumgebung* beschrieben worden sind gesetzt. Als nächster wird ein ClusterIP Service mit dem Namen *flowcontrol-service* erstellt welches den passenden Port des Containers zugänglich macht.

5.9 tracking-kubernetes.yaml

In dieser YAML Datei findet man die Deploymentkonfiguration des Tracking-Services vor. Es wir ein deployment mit dem Namen *tracking* erstellt welches das Docker Image *deryeger/dse-tracking* benutzt. Zusätzlich werden alle Konfigurationsvariablen die in dem Kapitel *Entwicklungsumgebung* beschrieben worden sind gesetzt. Bei dem Setzen der Konfigurationsvariablen werden unter anderem die Werte aus dem Secret mit dem Namen *dse-secrets-v2i* genutzt. Als nächster wird ein ClusterIP Service mit dem Namen *tracking-service* erstellt welches den passenden Port des Containers zugänglich macht.

6 API Documentation

Für eine interaktive Dokumentation findet man alle Swagger Dokumentation auf *http://< gateway_ip >/swagger-ui.html*. Dafür müssen die Services jedoch online sein. Zusätzlich findet man im Projekt unter dem *docs* Ordner, die Dokumentationen aller Services in Form von PDF und YAML Dateien.