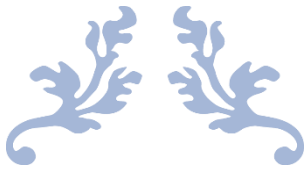

DOKU MODUL 347

Aimo Altorfer

Contents

Tag 1

Informieren

Nach einer Einführung des Dozenten in das Thema Docker schauten wir und noch Videos zu dem Thema an und haben so einen guten Überblick ins Thema erhalten.

Aufgabenstellung

Der Auftrag ist es ein Projekt zu erstellen welches Docker verwendet. Dabei sollen Container miteinander kommunizieren können und Datei in Volumes gespeichert werden. Was wir als Projekt machen war uns offen, jedoch gab es eine Empfehlung des Dozenten für ein Ticket System, wofür ich mich dann auch entschieden habe. Ich habe mich für folgende Tools entschieden: JS, CSS, HTML, Nginx für das Frontend, JS und Node.js für das Backend und MySQL für die Datenbank.

Frontend

Bei meinem Projekt habe ich beschlossen mit dem Frontend zu beginnen. Ich habe hierbei ein sehr simples Design kreiert mit minimalem Kontroller, was der User eingibt, da dies nicht der Fokus dieses Moduls ist. Mein fertiges Frontend sieht wie folgt aus:

Nun muss dieses Frontend auf einem Container laufen, was ich man mit einer Dockerfile einrichten kann. So sieht meine simple Dockerfile für das Frontend aus:

Dockerfile

```
# Basis-Image: Nginx
FROM nginx:alpine

# HTML, CSS und JavaScript in den Nginx-Container kopieren
COPY index.html /usr/share/nginx/html/



# Port 80 freigeben
EXPOSE 80

# Nginx starten
CMD ["nginx", "-g", "daemon off;"]
```

Nun konnte ich mein Frontend lokal über meinen Browser abrufen, solange der Container lief.

Tag 1 Zusammenfassung

Somit war ich nach Tag 1 mit dem Frontend fertig und hatte eine relativ gutes Verständniss dazu was Docker ist.

Tag 2

Absenz

Am Modul-Tag 2 war ich leider nicht anwesend aufgrund einer Krankheit. Ich habe mir allerdings einige Videos zum Thema angeschaut, um mich auf meine Nächsten Schritte im Projekt vorzubereiten.

Tag 3

Zielsetzung

Aufgrund meiner Absenz am letzten Modul-Tag musste ich heute ziemlich aufholen.

So habe ich mir vorgenommen die Datenbank, was ich mir als kleine Aufgabe vorgestellt habe, fertig zu stellen und mit dem Backend so weit wie möglich zu kommen, optimalerweise fertig.

Datenbank

In der docker-compose.yml Datei wird die Konfiguration für die Docker-Anwendungen definiert was im Falle der Datenbank wie folget aussieht:

```yaml
db:
  image: mysql:5.7
  environment:
    MYSQL_ROOT_PASSWORD: rootpassword
    MYSQL_DATABASE: tickets
    MYSQL_USER: user
    MYSQL_PASSWORD: password
  ports:
    - "3306:3306"
  volumes:
    - "C:/Users/aimoa/OneDrive - ipso! Bildung/Modul 347/Docker Project/init-db.sql:/docker-entrypoint-initdb.d/init-db.sql"
  healthcheck:
    test: ["CMD", "mysqladmin", "ping", "-h", "localhost"]
    timeout: 20s
    retries: 10
```

Hier werden schon einige Dinge wie z.B das Passwort festgelegt, Name der Datenbank usw.

Die Struktur für meine simple Datenbank wird jedoch in einer .sql Datei festgelegt, welche wie folgt aussieht:

init-db.sql

```sql
-- Datenbank erstellen, falls sie noch nicht existiert
CREATE DATABASE IF NOT EXISTS tickets;
USE tickets;

-- Tabelle erstellen, falls sie noch nicht existiert
CREATE TABLE IF NOT EXISTS tickets (
  id INT AUTO_INCREMENT PRIMARY KEY,
  name VARCHAR(255) NOT NULL,
  email VARCHAR(255) NOT NULL,
  issue TEXT NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Benutzer 'root' mit dem Passwort 'password' konfigurieren
-- und Zugriff von jeder IP-Adresse erlauben
ALTER USER 'root'@'%' IDENTIFIED WITH mysql_native_password BY 'password';

-- Privilegien neu laden
FLUSH PRIVILEGES;
```

Hier wird ganz einfach die Datenbank erstellt falls sie noch nicht existiert, zusammen mit einer Tabelle namens «tickets» mit den drei Spalten: id, name, email, issue und create_at welche zeigt wann das Ticket erstellt wurde. Ausserdem wird dem Benutzer «root» Zugriff erteilt und ein Passwort festgelegt.

Zum Überprüfen der Datenbank verwende ich MySQL Workbench. Diese Software kann unter folgendem Link heruntergeladen werden: https://dev.mysql.com/downloads/workbench/

Backend

Nun wo das Frontend und die Datenbank funktioniert komme ich zum Erstellen des Backends.

Da ich Node.js verwenden will muss dies zuerst installiert werden. Hier ist der link dazu: https://nodejs.org/en

Danach öffnet man ein Terminal und führt folgende Befehle aus um das Node.js-Projekt zu initialisieren und die Node-Module zu installieren:

```
cd meinBackendProjekt
```

```
npm init -y
```

```
npm install express
```

Nach dieser Installation habe ich die Server.js Datei erstellt, was die Logik des Backends beinhaltet.

Bei Interesse dieses Codes habe ich das Projekt auf einem Repository: https://github.com/AimoAltorfer/Docker-Project

Eine Basis für ein solchen Server sieht wie folgt aus:

```
const express = require('express');
const app = express();
const port = 3000;

app.get('/', (req, res) => res.send('Hello World!'));

app.listen(port, () => console.log(`Server läuft auf Port ${port}`));
```

Tag 3 Zusammenfassung

Tag 3 war ein sehr produktiver Tag. Ich bin mit der Datenbank und dem Backend so gut wie fertig geworden. Ich habe nur noch ein Error zwischen dem Frontend und dem Backend welches ich dann am Tag 4 in Angriff nehmen werde.

Tag 4

Zielsetzung

Heute muss ich zum Zahnarzt, weshalb ich nur am Morgen Zeit habe. Mein Ziel ist es das Projekt vollständig zum Laufen zu bringen da nur noch einige Errors im Wege stehen.

Errors

Ich habe nun 3 Fehler behoben, um das Projekt zum Funktionieren zu bringen.

Falsche Benennung

Der erste Fehler war ganz einfach das ich bei einigen Stellen im Code auf die Tabelle „ticket“ zugreifen wollte. Die Tabelle heisst aber natürlich „tickets“ was zu Problemen führte.

Falsches Login

In meiner docker-compose.yml Datei habe ich anstelle von meinem Passwort: «password», «rootpassword» verwendet, was dem user keine Rechte auf die Datenbank gab.

Abhängigkeiten

Da das Backend von der Datenbank abhängig ist hatte ich einige Probleme wo das Backend startete obwohl die Datenbank noch nicht ganz gestartet war. Mit einer Funktion namens «healthcheck» konnte ich das beheben:

```
healthcheck:
  test: ["CMD", "mysqladmin", "ping", "-h", "localhost"]
  timeout: 20s
  retries: 10
```

Images auf Repository laden

Ich habe die Images auch noch auf ein Repository hochgeladen.

Backend:
https://hub.docker.com/repository/docker/aimoaltorfer/modul_347_ticket_system_aimo_backend/general

Mysql:
https://hub.docker.com/repository/docker/aimoaltorfer/modul_347_ticket_system_aimo_mysql/general

Web:
https://hub.docker.com/repository/docker/aimoaltorfer/modul_347_ticket_system_aimo_web/general

Minikube

Einleitung

Minikube ist eine Lösung, um Kubernetes lokal zu betreiben, was die Entwicklung und das Testen von Cloud-nativen Anwendungen vereinfacht.

Voraussetzungen

- Installiertes Minikube und kubectl auf dem Entwicklungsrechner.
- Grundkenntnisse in Kubernetes und Docker.

Projekt-Setup

Minikube Starten:

Starten Sie Minikube mit dem Befehl:

```
minikube start
```

Docker-Environment einstellen:

Stellen Sie sicher, dass Ihre Shell auf die Docker-Daemon von Minikube verweist:

```
eval $(minikube docker-env)
```

Deployment erstellen:

Verwenden Sie die bereitgestellten Kubernetes-YAML-Dateien, um Ihre Services und Deployments zu erstellen:

```
kubectl apply -f <pfad-zu-deployment.yaml>
```

Services Exponieren:

Exponieren Sie Ihre Services, um sie über Minikube zugänglich zu machen:

```
kubectl expose deployment <name-des-deployments> --type=NodePort
```

URL Abrufen:

Um auf Ihre Anwendung zuzugreifen, holen Sie die URL mit:

```
minikube service <name-des-services> --url
```

Anwendungstest

- Testen Sie Ihre Anwendung, indem Sie auf die durch Minikube bereitgestellte URL zugreifen.
- Überwachen Sie die Ressourcen und Logs Ihrer Pods, um sicherzustellen, dass alles wie erwartet funktioniert.

Abschluss und Wartung

- Beenden Sie Minikube mit minikube stop, wenn es nicht mehr benötigt wird.
- Aktualisieren Sie regelmäßig Ihre Kubernetes-Konfigurationen entsprechend den Anforderungen des Projekts.

Tag 5

Fertigstellung

Tag 5 ist heute und somit habe ich diese Dokumentation fertig gestellt.

Ich habe ausserdem alles auf ein Repository hochgeladen und eine Anleitung in form von einem README.md geschrieben.

Fazit

Das Modul hat mir viel Spass bereitet und ich kenne mich mit Docker nun ziemlich gut aus.

Das selbständige Arbeiten hat mir auch sehr gefallen, weil es uns beibringt, wie wir selbst nach Informationen sammeln und uns unsere Arbeit gut aufteilen.

Obwohl ich 1.5 Tage gefehlt habe, finde ich das ich das durch eine gute Arbeitshaltung wieder wett gemacht habe und kann nun auf mein fertiges Docker Projekt stolz sein.