Workshop FHNW wodss Dokumentation

Projektname: Workshop FHNW wodss
Autoren: Thibault Gagnaux, Philipp Lüthi und Simon Wächter
Version: 0.0.3

Dokumentenmanagement

Version:	0.0.3
Datum:	17.04.2019
Autoren:	Thibault Gagnaux, Philipp Lüthi und Simon Wächter
Dokumentname:	Dokumentation.docx
Klassifizierung	Keine Klassifizierung

Änderungen

Version	Datum	Beschreibung	Autor
0.0.1	17.03.2019	Initiale Dokumentation mit der Beschreibung des Technologiestacks	Simon Wächter
0.0.2	04.04.2019	Wechsel auf Docker und Heroku Deployment	Simon Wächter
0.0.3	17.0.2019	Wechsel auf eine zentrale Konfigurationsmöglich-keit	Simon Wächter

Inhaltsverzeichnis

1 Übersicht

1.1 Gruppe

Unsere Gruppe besteht aus:

- Thibault Gagnaux
- Philipp Lüthi
- Simon Wächter

1.2 Verwendete Technologien

- Frontend
  - Preact, um auf eine leichtgewichtige React.js Alternative zu setzen. Zumal diese über keine Lizenzprobleme wie React.js verfügt: https://medium.freecodecamp.org/facebook-just-changed-the-license-on-react-heres-a-2-minute-explanation-why-5878478913b2
- Backend
  - Spring Boot mit Java zur Realisierung des Webservers
  - jOOQ als Abstraktionslayer zu SQL (Basis: PostgreSQL Server)
  - MapStruct zum Mappen der DTO/Entitäten
  - Springfox zum Dokumentieren und Anbieten eines Swagger Interfaces
- Building
  - Gradle mit Java 11
- Deployment
  - Docker Container mit Deployment auf Heroku

1.3 Beschreibung der Schnittstelle

Basierend auf den initialen Arbeiten von David und Christian hat Simon die API erweitert und möchte über diese am 18. März abstimmen lassen. Der Kundenwunsch von Herrn König zum Integrieren von Zeitpensen wurde umgesetzt. Spezifikation: https://github.com/swaechter/fhnw-wodss-spec
Momentan noch nicht vorhanden ist ein Testdatenset, welches die Integration vereinfacht

1.4 Beschreibung Authentifizierung

Die Schnittstelle basiert auf dem JWT Mechanismus, welcher wie folgt abläuft:

1. Client besitzt noch keine Authentifizierung
2. Client steuert POST /api/token mit einer Emailadresse samt Passwort als Request Parameter an
3. Der Server verifiziert diese Informationen und stellt ihm ein JWT Token in der Response aus. In diesem JWT Token ist der ganze Mitarbeiter als Claim «employee» integriert (Siehe Grafik unten)
4. Der Client speichert dieses Token ab (Local Storage)

Der Aufruf einer geschützten Schnittstelle läuft wie folgt ab:

1. Der Client liest das Token aus dem Local Storage
2. Der Client schickt das Token als HTTP Header «Authorization» im Format «Bearer TOKEN» in der jeweiligen Anfrage mit
3. Der Server validiert via Filter die Signatur des Tokens und lässt dementsprechend den Zugriff zu

2 Inbetriebnahme

2.1 Überlegungen und Möglichkeiten

Wir möchten das Deployment unseres Projektes für andere Gruppen so einfach wie möglich gestalten, aber doch Raum für Anpassungen und Konfigurationen lassen. Wir haben uns deshalb entschlossen, das Projekt via Heroku als Docker Applikation zu betreiben und für andere zugänglich zu machen. Dies resultiert in 3 möglichen Deploymentszenarien:

1. Die Gruppe verwendet den durch uns auf Heroku betriebenen Backendserver, welcher unter dem Link https://fhnw-wodss.herokuapp.com zu finden ist (**Hinweis**: Die Applikation wird im Free-Tier Modell deployt und bei Nichtverwendung schlafen gelegt. Bei einem späteren Aufruf dauert das Wecken bis zu einer Minute)
2. Die Gruppe deployt den Docker Container via Heroku und betreut ihn auch selbst
3. Die Gruppe verzichtet auf Heroku unter der Bedingung, einen eigenen PostgreSQL Server betreiben zu müssen

Die drei Varianten werden nachfolgend genauer besprochen

2.2 Variante 1: Verwendung der durch uns gehosteten Heroku Lösung

- Link zur Webapplikation: https://fhnw-wodss.herokuapp.com
- Link zum Swagger Interface: https://fhnw-wodss.herokuapp.com/swagger-ui.html

2.3 Variante 2: Eigenes Heroku Deployment

Vor dem eigentlichen Beginn muss ein Heroku Account erstellt und die Heroku CLI heruntergeladen werden: https://devcenter.heroku.com/articles/heroku-cli

Nach der Installation meldet man sich lokal an:

Anmelden an der Heroku CLI
heroku login -i

Für das Bauen und Hochladen der containerisierten Applikation muss ferner Docker installiert werden.

Nach den Vorbereitungen und einem Git Clone (http://github.com/swaechter/fhnw-wodss) wechselt man in das Projektverzeichnis und erstellt eine neue Heroku Applikation. Der Name „fhnw-wodss“ muss dabei durch einen anderen Namen ersetzt werden, da dieser schon belegt ist (z.B. fhnw-wodss-john):

Erstellen einer neuen Heroku Applikation
heroku create fhnw-wodss

Der erstellten Applikation soll jetzt auch eine PostgreSQL Instanz angehängt werden:

Anhängen einer PostgreSQL Instanz
heroku addons:create heroku-postgresql:hobby-dev

Nach dem Erstellen der PostgreSQL Instanz müssen wir deren Credentials anzeigen und aufsplitten:

Anzeigen der PostgreSQL Credentials
heroku config

Der Aufbau der URL ist wie folgt:

postgres://**DATENBANKBENUTZER**:**DATENBANKPASSWORT**@**DATENBANKHOST:5432/DATENBANKNAME**

Da unser Build und die Applikation diese Credentials benötigen, kopieren wir das Template «config/application.properties.template» nach «config/application.properties» und tragen diese dort ein. Unser Buildsystem als auch die Aplikation selber verwenden untereinander unterschiedliche & inkompatible URL Formate, weshalb die URL zwei mal zu konfigurieren ist (aber in einem unterschiedlichen Format):

- postgresql statt jdbc:postgresql
- Integration von Benutzername und Passwort in den Link

Eine fiktive Beispielkonfiguration sieht wie folgt aus:

Fiktive Konfiguration config/application.properties
Runtime settings for our Spring application (Required for runtime and development) PORT=8000 DATABASE_URL=postgres://uvyrwlvnktglsa:44ce2bd0901d80f3be93968c0552e59ea29ddc3a284cbd7edaa20ed224ad1eb4@ec2-54-221-113-7.compute-2.amazonaws.com:5432/d6hg874i8q0qau # Build time settings for the Jooq SQL table generation + Flyway data migration (Required for development only) JDBC_DATABASE_URL=jdbc:postgresql://ec2-54-221-113-7.compute-2.amazonaws.com:5432/d6hg874i8q0qau JDBC_DATABASE_USER=uvyrwlvnktglsa JDBC_DATABASE_PASSWORD=44ce2bd0901d80f3be93968c0552e59ea29ddc3a284cbd7edaa20ed224ad1eb4

Nach dem Setzen der Credentials kann die Applikation in Docker gebaut und hochgeladen werden (Dieser Prozess dauert beim ersten Mal circa 5 Minuten und danach circa 1 Minute):

Bauen und Hochladen der Applikation
heroku container:login heroku container:push web heroku container:release web

Nach dem Hochladen muss auf eine Applikationsinstanz hochskaliert werden. Das Starten der ersten Instanz kann dabei gleich beobachtet werden:

Setzen der Skalierung
heroku ps:scale web=1 heroku logs --tail

Die Applikation kann nach dem Start auch direkt im Browser geöffnet werden:

Öffnen der Applikation in einem Browser
heroku open

2.4 Variante 3: Betreiben einer Lösung ohne Heroku

Das Betreiben einer Lösung ohne Heroku wird von uns nicht empfohlen, der Korrektheit halber aber trotzdem erwähnt:

1. Für das Bauen und Betreiben des Projektes wird ein zentral erreichbarer PostgreSQL Server benötigt. Die Datenbank und die Zugangsrolle müssen durch den Administrator selbst erstellt werden und ihm bekannt sein
2. Die Credentials werden analog der Variante 2 in die config/application.properties eingetragen
3. Via docker build -t myimagename . wird das Docker Image gebaut
4. Via docker run -e "PORT=<PORT>" -e "DATABASE_URL=<URL>" myimagename wird das Image gestartet