Kubernetes Full Stack Application

Marco Jakob

March 26, 2019

Contents

1 Introduction

This Introduction should give an end to end overview to deploy a sample Application with multiple Services and a Databases completely on Kubernetes. The Application generates Random Numbers, saves them with a Timestamp on a database. Displays the new Number on a Frontend and also shows Graphs from previous Random Numbers based on their Producers Id. The Goal of the whole application is first to create an end to end Application fully Cloud native which runs as well locally without any changes. The second goal is to show graphically the self healing process of Kubernetes, namely when we delete a random generator Pod, it should automatically create a new Random generator Pod with a new Id.

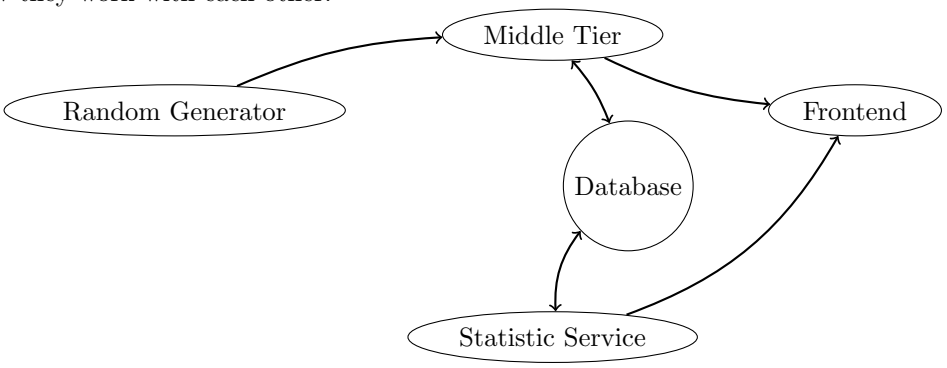
1.1 Technology

Service	**Technology**
Random Number Generator circle	Python Flask
Database to store previous Numbers	Mysql Database
Microservice for requesting new Numbers	Spring Boot Java
Microservice for gathering Statistics	Spring Boot Java
Frontend	Angular with Spring Boot
Routing Server	Nginx
Server OS	Ubuntu Server 18.5.1 LTS

All the Applications are Running inside of an own Docker Container and Managed within a Kubernetes Pod and accessible via a Kubernetes Service where only the Frontend Service is bound to a Node Port.

1.2 Model

The Following Picture should give an overview about the different Services and how they work with each other.



2 Installing Kubernetes

3 Create Services

3.1 Database

In the further sections we will use a plain mysql docker container as a database. In order to use the Application completely locally, a local mySQL database has to be installed and a schema for the random generator application needs to be provided for a specific randgenuser. If there is not yet a local mySQL database installed it can be done with the following command:

```
sudo apt-get update
sudo apt-get install mysql-server
```

After the installation log in to MySQL as root.

```
sudo mysql
```

Now the Database and the user can be created and permissions to the respective schema will be given.

```
CREATE database db_example;
CREATE USER 'springuser'@'localhost' IDENTIFIED BY ThePassword;
GRANT ALL PRIVILEGES ON *.db_example TO 'springuser'@'localhost';
```

With this, the Database is ready to used by the Random Generator Example from localhost.

3.2 Random Generator

The Random Generator will be a verry simple Python App which has an unique Id per Service instance. It will return his id and a new Random Number. For this application we create a folder called RandGen with the following Structure.

```
RandGen
├── Dockerfile
├── rand_gen.py
└── requirements.txt
```

The Dockerfile is needed to create the Docker Image which will be used from Kubernetes to Create the Random Generator Service. rand_gen.py is the complete Random Generator Python application based on Flask[1].

```
from flask import Flask, jsonify
import random
import uuid

app = Flask(__name__)
user_id = uuid.uuid4().int

@app.route('/')
```

[1] More information about Flask can be found on the official Homepage `http://flask.pocoo.org/`

```
def random_number():
    object = {'id' : str(user_id),
              'randNumber': str(random.randint(0,1000))}
    return jsonify(object)

if __name__ == '__main__':
    app.run(port=5050,host='0.0.0.0')
```

In requirements.txt are the Python packages specified. In this case it is flask. therefore this file contains only one line which is

```
Flask==1.0.2
```

3.3 Middle Tier

The Middle Tier Application is created with Spring Initializer. Unter the following link `https://start.spring.io/` Helps to bootstrap very fast a simple Spring Boot application. In our case the fields should be filled out as followed:

Project	Maven Project
Language	Java
Spring Boot	2.1.3 (All versions would apply)
Group	ch.toky.randgen
Artifact	middletier
Name	MiddleTier
Description	Middle Tier Service for Random Generator Application
Packaging	Jar
Java Version	8
Dependencies	Web, JPA, MySQL, Rest Repositories

The Website will create a zip file to download. This zipfile has to be extracted to the desired Project folder.

```
middletier
├── Dockerfile
├── mvnw
├── mvnw.cmd
├── pom.xml
└── src
```

all blue folders are already given. the Dockerfile still has to be created.
Now inside of the Project create the following Project Structure and files.

```
ch.toky.randgen.middletier
├── MiddletierApplication.java
├── RandomNumber.java
├── ch.toky.rand-gen.middle-tier.model
│   └── PodStat.java
└── ch.toky.rand-gen.middle-tier.repository
    └── PodStatRepository.java
```

The file MiddleTierApplication.java should already be in place. as Spring Boot Initializr created this with the complete Folder structure. the packages

controller, model and repository need to be created. Inside of controller, A file named MiddletierController.java needs to be created with the following content.

Within the model package two new Files have to be created: PodStat.java and RandomNumber.java.

PodStat is an Entity and therefore the class has to be annotated with @Entity. It contents the following fields with their respective getter and Setter Methods.

```
@Id
@GeneratedValue(strategy=GenerationType.AUTO)
private Long podStatID;
private String id;
private Long timeStamp;
private Long counter;
```

The class RandomNumber is only a DTO which is retreived from the random generator Service. Therefore there is no need for a annotation. Only the following Fields needs to be declared with their getters and setters:

```
private String id;
private Long randNumber;
```

In the repository package the file PodStatRepository.java will be created. As this java file is not a class but an interface it needs to be changed to interface and extends JpaRepository¡PodStat, Long¿ and it will be annotated with @Repository

Now those two methods are created

```
@Query("Select count(ps.id) from PodStat ps where ps.id = ?1")
Long countUniqueId(String id);

@Query("Select count(ps.id) from PodStat ps")
Long findMaxCount();
```

Witin the controller all the endpoints are declared. Therfore it will be annotaded with @RestController

Those fields are needed within the controller and are therefore declared first.

```
@Autowired
private ObjectMapper objectMapper;
@Autowired
private PodStatRepository podStatRepository;
private Map<String, String> env = System.getenv();
```

As there should not be any hardcoded url according to the 12 Factor application [2], the Url will be retreived through a System Variable.

Now a controller Method with the business logic can be declared:

```
@RequestMapping(value = "/", produces = "application/json")
public RandomNumber getRandom() {

    RandomNumber randNum = getNewRandomNumber();
    Long maxCountId = podStatRepository.countUniqueId(randNum.getId
    ());
```

[2] Accessible at `https://12factor.net/`

```
    Long maxCountOverall = podStatRepository.findMaxCount();
    PodStat tmpPod = new PodStat();
    tmpPod.setId(randNum.getId());
    tmpPod.setTimeStamp(maxCountOverall+1);
    tmpPod.setCounter(maxCountId +1);
    podStatRepository.save(tmpPod);

    return randNum;

}
```

The above Method listens on the path / and returns the Random number after saving it to the database with the actual count.

Now the last Method is used to retreive the Random Number from the Random Generator Service.

```
private RandomNumber getNewRandomNumber() {

    String randGenUrl=env.get("RANDOM_GENERATOR_URL");
    String URL = randGenUrl;
    RestTemplate restTemplate = new RestTemplate();

    return   restTemplate.getForObject(URL, RandomNumber.class);

}
```

Above Source code contains the Controller and the Service, which should be separated optimally. Next two files represents POJOs which are going to be used from controller and provided by Repositories.

The last thing which needs to be done for the Middle Tier Service is to setup the default values for the Database Connection.

```
spring.jpa.hibernate.ddl-auto=update
spring.datasource.username=random_user
spring.datasource.password=random_password
spring.datasource.url=jdbc:mysql://localhost:3306/rand_numbers
```

The datasource configurations are going to be overwritten by Environment Variables once they run inside of a Docker Container.

3.4 Statistic Service

The Statistic Service has the same setup as the Middle Tier client. Therefore the Project will be created too with Spring Boot initializer.

Project	Maven Project
Language	Java
Spring Boot	2.1.3 (All versions would apply)
Group	ch.toky.randgen
Artifact	stattier
Name	Stattier
Description	Statistic Tier Client for Random Generator Application
Packaging	Jar
Java Version	8
Dependencies	Web, JPA, MySQL, Rest Repositories

3.5 Frontend

The frontend Application is build with Spring Boot and
Creating Spring Boot Application:

Project	Maven Project
Language	Java
Spring Boot	2.1.3 (All versions would apply)
Group	ch.toky.randgen
Artifact	frontend
Name	Frontend
Description	Frontend Service
Packaging	Jar
Java Version	8
Dependencies	Web, Rest Repositories

3.6 running with Docker

Create a Network

```
docker network create  -d bridge rand-gen-network
```

Start the Database Container

```
docker run -d --name rand-gen-database \
-e MYSQL_ROOT_PASSWORD='password' \
-e MYSQL_USER='random_user' \
-e MYSQL_PASSWORD='random_password' \
-e MYSQL_DATABASE='rand_numbers' \
--network rand-gen-network \
mysql:5.6
```

Start the Random Generator App

```
docker run -d --name=rand-gen-app \
--network rand-gen-network \
rand-gen-image
```

Start the middle tier Application

```
docker run -d \
--name middle-tier \
--network rand-gen-network \
middle-tier
```

Starting the Stat Tier Application

```
docker run -d \
--name stat-tier \
--network rand-gen-network \
stat-tier-image
```

Start the Frontend Service

```
docker run -d -p 8080:8080 \
--network rand-gen-network \
--name rand-frontend \
rand-frontend
```

4 Deploy Services to Kubernetes