Project 1: Advanced Infrastructures for Data Science

Jan Janiszewski, UCNumber:2023198681
jasiekj1@gmail.com
University of Coimbra
Coimbra, Coimbra, Portugal

1 INTRODUCTION

All the assignments described in the following section are related to running a Multi-Container application using technologies such as Docker and Kubernetes.

1.1 Multi-Container application

The application consists of three modules:

(1) Database - MongoDB database.
(2) Backend - Node.js server. The application files are located in the folder "backend".
(3) Frontend - React application providing user interface. The application files are located in the folder "frontend".

2 ASSIGNMENTS 1 & 2

2.1 Goals of assignment

Goals of this assignments were to :

(1) Build images for our multi-container app
(2) Launch them
(3) Configure containers connectivity
(4) Make Database data and Backend logs persistent

2.2 Solution

2.2.1 Database. We will be using image available on Docker Hub so no need to build one, "v" option allow us to define a volume for our container. I'm placing a path to Database data to make it persistent.

```
$ docker run --name mongodb -v mongo:/data/db mongo
```

In order to connect Backend to Database we need to know IP Address of the container :

```
$ docker ps
$ docker inspect <id_of_db_image>
```

2.2.2 Backend. In order to achieve connectivity between containers I modified app.js file of Backend application and replaced "localhost" with the IP Address found in previous step. I also created Dockerfile and placed in in Backend application folder. Now I built an image:

```
$docker build -t backend backend
```

After image was built I started it, "v" allow us to define a volume for storing Backend logs and "p" is exposing container ports to the host.

```
$ docker run --name backend -p 80:80 -v \
backend:/backend/logs backend
```

2.2.3 Frontend. I created Dockerfile and placed it in Frontend application folder. Now I built an image:

```
$ docker build -t frontend frontend
```

After it, I started it:

```
$ docker run --name frontend -p 3000:3000 --rm
frontend
```

3 ASSIGNMENT 3

3.1 Goals of assignment

Goals of this assignment were to :

(1) Deploy the multi-container application using Docker Compose

3.2 Solution

To solve this task, the Dockerfile's from the previous ones were used and a new file docker-compose.yml was created .This file contains information about all three containers that make up the applications, the volumes they use, port mapping and interdependencies. Thanks to the fact that I wrote in the description of the container with the Backend module that it depends on the container with the Database module, we can, instead of hardcoding Ip Address as in the previous assignemnt, simply enter the name of the container with the Database in the app.js file of the Backend application. To build and run the entire application I type the command:

```
$ docker compose up --build
```

After a while, the multi-container application is ready to use.

4 ASSIGNMENT 4

4.1 Goals of assignment

Goals of this assignment were to :

(1) Install Kubernetes (minikube)
(2) Deploy the multi-container application using Kubernetes Imperative Approach
(3) Deploy the multi-container application using Kubernetes Declarative Approach

Both deployments should be done without interconnectivity and without data persistence. Replicas (database – 1; backend – 2; frontend – 3)

4.2 Solution

4.2.1 Installing minikube. Since I am working on an Apple Silicon M1 computer, I could not install Minikube with Homebrew. Instead, I use it using Docker as the engine.

```
$ curl -LO <link with minikube>
$ sudo install minikube-darwin-arm64 <path>
```

Now minikube can be started by typing:

```
$ minikube start --driver=docker \
--alsologtostderr
```

4.2.2 Imperative approach. I started with creating deployment object for Database, I'm creating a single replica and I'm using "mongo" image from DockerHub:

```
$ kubectl create deployment kubs-db-deploy \
--image=mongo --replicas=1
```

To be able to access pod with Database, i I also created service object, I'm exposing pod on 27017 host port, since I want to connect with it from another pods within cluster I'm using ClusterIP service:

```
$ kubectl expose deployment kubs-db-deploy \
--type=ClusterIP --port=27017
```

I can map the container port to an IP which we can reach from our local machine:

```
$ minikube service kubs-db-deploy
```

For Backend and Frontend images i created separate repositories on my DockerHub account. I built images with command I used in previous assignments and pushed them to repositories:

```
$ docker build -t \
jjaniszewski/backend_aids_proj backend
$ docker push jjaniszewski/backend_aids_proj
$ docker build -t \
jjaniszewski/frontend_aids_proj frontend
$ docker push jjaniszewski/frontend_aids_proj
```

Now I created deployment object for Backend, I used image from my DockerHub repository and as requested set number of replicas to 3:

```
$ kubectl create deployment kubs-backend-deploy \
--image=jjaniszewski/backend_aids_proj \
--replicas=3
```

I created LoadBalancer service object since connection will be made from outside the cluster, I am exposing pod on 80 host port:

```
$ kubectl expose deployment kubs-backend-deploy \
--type=LoadBalancer --port=80
```

As a last step I created deployment object for Frontend, I also used image from my DockerHub repository and set number of replicas to 2:

```
$ kubectl create deployment kubs-frontend-deploy \
--image=jjaniszewski/frontend_aids_proj \
--replicas=2
```

I created LoadBalancer service object since connection will be made from outside the cluster, I am exposing pod on 3000 host port:

```
$ kubectl expose deployment \
kubs-frontend-deploy \
--type=LoadBalancer --port=3000
```

I can verify that all pods are running:

```
$ kubectl get pods
```

And to connect with Frontend I typed:

```
$ minikube service kubs-frontend-deploy
```

It's not working properly but interconnectivity wasn't a goal of this assignment.

4.2.3 Declarative approach. In the declarative approach, we will also create deployment and service objects but this will be done through configuration files with a "yaml" extension. I created two files for each container:

- Describing deployment
- Describing service

Having these files, I created objects based on them:

```
$ kubectl apply -f=db-deployment.yaml
$ kubectl apply -f=db-service.yaml
$ kubectl apply -f=backend-deployment.yaml
$ kubectl apply -f=backend-service.yaml
$ kubectl apply -f=frontend-deployment.yaml
$ kubectl apply -f=frontend-service.yaml
```

Running:

```
$ minikube service kubs-frontend-deploy
```

has the same effect as in imperative approach.

5 ASSIGNMENT 5

5.1 Goals of assignment

Goals of this assignment were to:

(1) Connect the backend component with the database
(2) Connect the frontend component with the backend component
(3) Persist data from the Database and logs from Backend.

5.2 Solution

To solve this assignment I used "yaml" files from previous assignment. Following adjustments were made:

(1) In db-service.yaml I set a type of service to ClusterIP since the connection will be made within the cluster.
(2) In backend-deployment.yaml I added entry with environment variable with Database service IP address, I modified Backend source code to use this variable.
(3) I did the same for the frontend-deployment.yaml to establish Backend - Frontend connection. I also needed to modify Frontend source code.
(4) I created two yaml files describing Persistent Volumes.
(5) I create two yaml files describing Persistent Volume Claim objects. This objects allow to establish connection between pods and Persistent Volumes.
(6) I modified deployment files for Backend and Database to be able to use the Persistent Volumes I created in last step.

Now having all the files I created objects based on them in following order:

```
$ kubectl apply -f=pv.yaml
$ kubectl apply -f=pvc-db.yaml
$ kubectl apply -f=db-deployment.yaml
$ kubectl apply -f=db-service.yaml
 $ kubectl apply -f=pv2.yaml
$ kubectl apply -f=pvc-backend.yaml
$ kubectl apply -f=backend-deployment.yaml
$ kubectl apply -f=backend-service.yaml
$ kubectl apply -f=frontend-deployment.yaml
$ kubectl apply -f=frontend-service.yaml
```

Due to some React related issues Frontend container cannot establish connection with Backend. But I made series of requests to Backend to confirm that the other containers are running and connected to each other.