Lab 1: Introduction to Google Kubernetes Engine (GKE)

Course: SOFE 4790U
Date: 18/09/2022
Lab #: 1 (Group Report)
Group #: 9
Submitted by: Matheeshan Sivalingam(100703887)
Hemshikha Sultoo (100670616)
Aaditya Rajput (100622434)
Aayush Parikh(100724827)

GITHUB LINK: https://github.com/matheeshan-sivalingam/SOFE-4790U-Lab-One

Objective:

1. Get familiar with Docker images and containers.
2. Learns various Kubernetes tools.
3. Learn how to use Google Cloud Platform (GCP).
4. Compose YAML files to deploy cloud applications.

Discussion:

Summarize what you have learned about docker and Kubernetes including the used terminologies and descriptions. What are the advantages and disadvantages of using docker images against virtual machines?

Docker is an open-source platform used in DevOps that allows developers to build, deploy, and manage containers. Containers are a virtualized run-time environment that hosts isolated applications. These containers only contain the necessary libraries and dependencies required to run the applications. These also include a Docker image which holds instructions about what is required when creating these containers. Using Docker allows one to run code effectively in any environment

Components of Docker

1) Docker Client and Server
   Docker use a Client-Server architecture where communication between the two is facilitated through the REST API. Any command by the Docker client is first translated by the REST API, then sent through to the Docker Daemon which in turn checks for the request and interacts with the OS to make necessary changes to the containers.

2) Docker Images
   Docker images can be described as a template of instructions required to generate Docker containers. To build a docker image, a docker file is required. The docker image can be stored under the docker registry.

3) Docker Containers
   A Docker container is an isolated executable software package that is equipped with all required applications and dependencies. Each application within the container is isolated. A Docker host can have multiple containers running simultaneously.

4) Docker Registry/Hub

The registry stores and distributes the Docker images depending on whether they are set to be public or private. The Docker client can use the pull command to get an image or the push command to store the image in the registry.

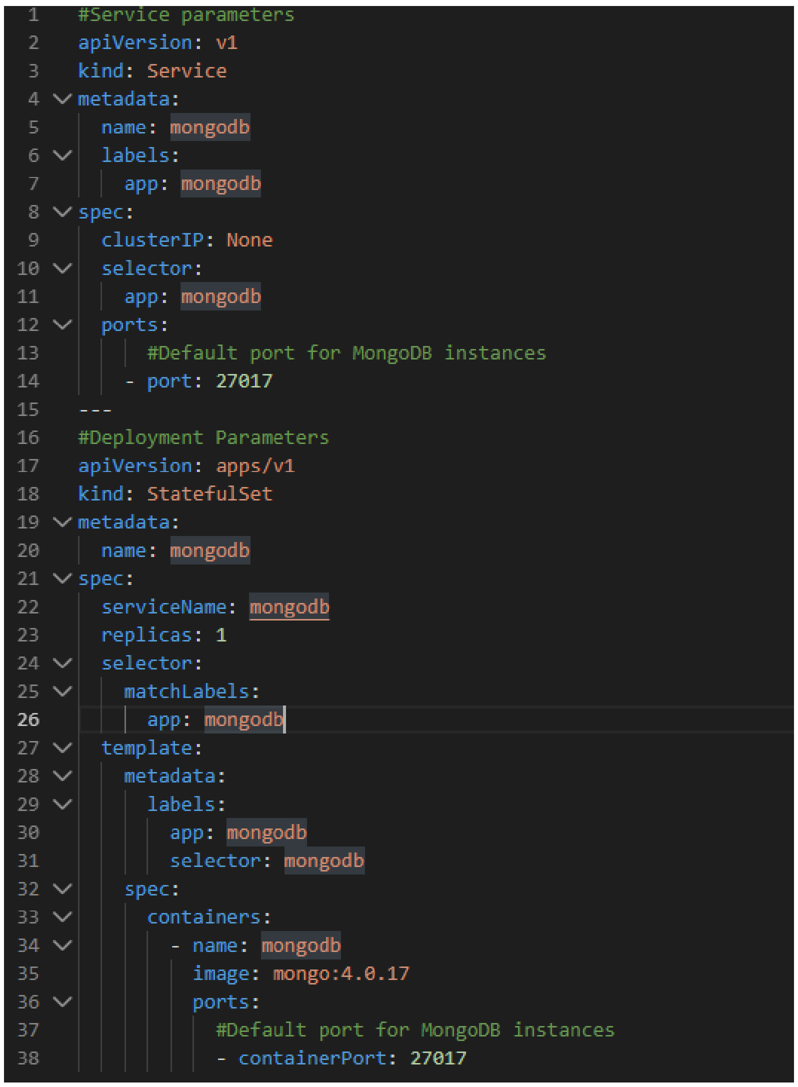
Kubernetes is an open-source system that automates and allows one to manage multiple containers. The containers created by Docker are managed and manipulated by the Kubernetes API. It allows containers to communicate with each other and allows replacement or upgrades to be performed without much downtime. A good analogy that helped us understand how and what Kubernetes does is to think of Kubernetes as a conductor of an orchestra. Similar to how a conductor acts as a guide to control and manage the music in an orchestra or choir, Kubernetes essentially helps manages a group of containers so that they can work together and perform their intended services when needed. These containers are grouped together into pods where they share the same data and resources which are then deployed and managed autonomously through Kubernetes.

Topic	**Docker Image**	**Virtual Machine**
Scaling	Scaling up with Docker is way easier.	It’s harder to scale up with virtual machines.
Boot-up time	Takes less time to boot up.	Takes more time to boot up.
Efficiency	More efficient.	Less efficient.
Operating System	Takes less space.	Demands a large amount of space.
Performance	Use of containers hosted by a single Docker engine supports better performance.	Running numerous virtual machines simultaneously eventually takes a toll on the overall performance of the system.
Space/memory allocation and management	Different containers are able to share and reuse data as well as space.	Space and data occupied cannot be shared.
Portability	Compatible with various platforms and therefore, portable.	Not compatible with all platforms.

Security	Poor security as a cluster can be exploited if an attacker gets access to one container	Strong security as the host kernel is isolated

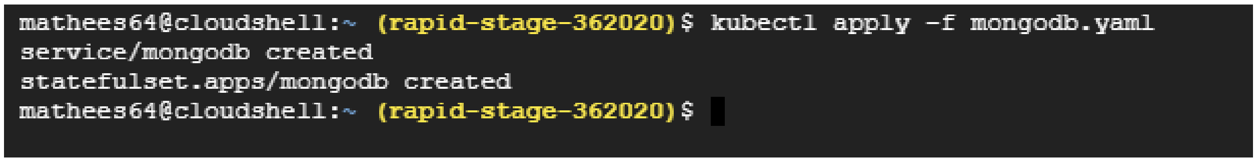
Design:
MongoDB is another type of database. It's required to deploy it using GKE using a YAML file. If you used any Kubernetes tool in your deployment that is not included in the lab you should describe it and why you used it

As it was unclear in the lab manual, we made an assumption that any method to deploy MongoDB is acceptable as long as we are able to access and run basic commands in the pod. Different methods of deploying MongoDB were used by each member. One method used to deploy MongoDB on the Google Kubernetes Engine was to deploy it as a statefulset using a YAML file. The YAML file for the MongoDB deployment followed a similar format to the MySQL file with the exception of the container port being set to the default port for MongoDB instances (port: 27017). No additional Kubernetes tools were used in the deployment that was not included in the lab.



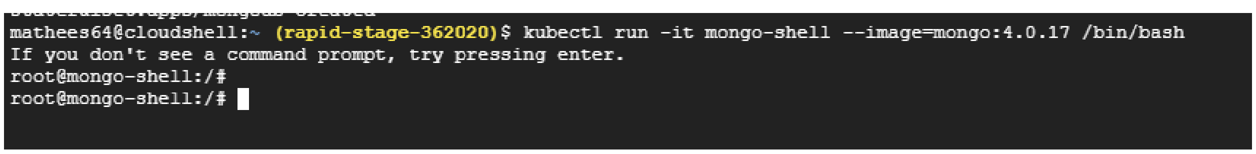
```yaml
#Service parameters
apiVersion: v1
kind: Service
metadata:
  name: mongodb
  labels:
    app: mongodb
spec:
  clusterIP: None
  selector:
    app: mongodb
  ports:
    #Default port for MongoDB instances
    - port: 27017
---
#Deployment Parameters
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mongodb
spec:
  serviceName: mongodb
  replicas: 1
  selector:
    matchLabels:
      app: mongodb
  template:
    metadata:
      labels:
        app: mongodb
        selector: mongodb
    spec:
      containers:
      - name: mongodb
        image: mongo:4.0.17
        ports:
          #Default port for MongoDB instances
          - containerPort: 27017
```

Figure 1 - Configuration parameters set on the MongoDB deployment



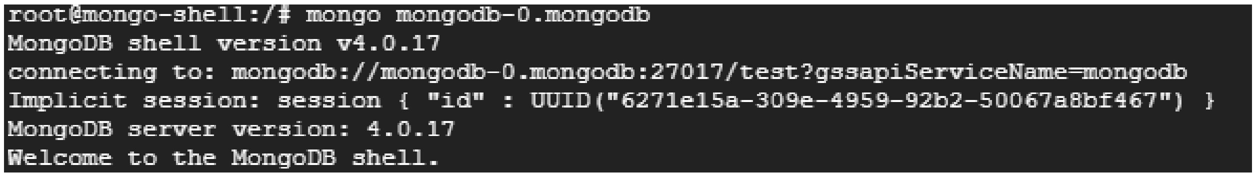
```
mathees64@cloudshell:~ (rapid-stage-362020)$ kubectl apply -f mongodb.yaml
service/mongodb created
statefulset.apps/mongodb created
mathees64@cloudshell:~ (rapid-stage-362020)$
```

Figure 2 - Used apply command to create the MongoDB service and statefulset



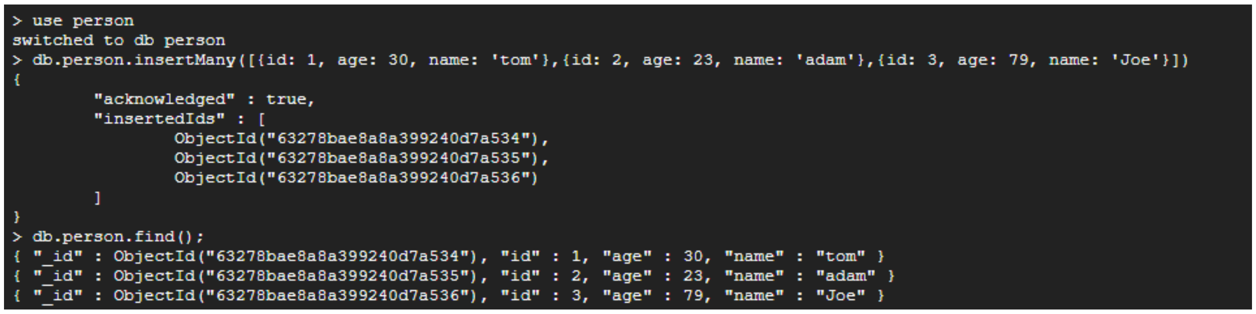
```
mathees64@cloudshell:~ (rapid-stage-362020)$ kubectl run -it mongo-shell --image=mongo:4.0.17 /bin/bash
If you don't see a command prompt, try pressing enter.
root@mongo-shell:/#
root@mongo-shell:/#
```

Figure 3 - Used run command to run a MongoContainer inside the cluster



```
root@mongo-shell:/# mongo mongodb-0.mongodb
MongoDB shell version v4.0.17
connecting to: mongodb://mongodb-0.mongodb:27017/test?gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("6271e15a-309e-4959-92b2-50067a8bf467") }
MongoDB server version: 4.0.17
Welcome to the MongoDB shell.
```

Figure 4 - Connect and open the MongoShell on the mongodb-0 pod



```
> use person
switched to db person
> db.person.insertMany([{id: 1, age: 30, name: 'tom'},{id: 2, age: 23, name: 'adam'},{id: 3, age: 79, name: 'Joe'}])
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("63278bae8a8a399240d7a534"),
                ObjectId("63278bae8a8a399240d7a535"),
                ObjectId("63278bae8a8a399240d7a536")
        ]
}
> db.person.find();
{ "_id" : ObjectId("63278bae8a8a399240d7a534"), "id" : 1, "age" : 30, "name" : "tom" }
{ "_id" : ObjectId("63278bae8a8a399240d7a535"), "id" : 2, "age" : 23, "name" : "adam" }
{ "_id" : ObjectId("63278bae8a8a399240d7a536"), "id" : 3, "age" : 79, "name" : "Joe" }
```

Figure 5 - Successfully inserted document to MongoDB database.