SOFE 4630U
Cloud Computing

Project Milestone #1
2022/02/03

Iaas: Virtualization and Containerization

Jerusha Macwan: 100723319

Link to videos:
https://drive.google.com/drive/folders/1cSJ7jbZQjLa6WrI37HkYaDQtGGDbyDlZ?usp=sharing

This report is a summary of the tasks completed in order to successfully set up docker and run containers on the local machine, create and run a multi container docker application on the web and also deploy a web application on GCP using Kubernetes.

Here are a few definitions of the key terms.

Docker Image: Basically a blueprint of docker containers which is used to implement and run Docker containers. These are immutable and contain all the tools, libraries and dependencies to run an application. Docker images become docker containers when they run on the docker engine.

Docker Container: It is a standard unit of software that packages up code and all its dependencies so that the application runs quickly and reliably.

Docker registry: Helps in storing, distributing and version control of Docker images on the server side. It is highly scalable.

The first part involves creating 2 containers from different images, running the code and also deleting one of the containers.

The commands used in completing the task and a brief description about them.

- docker build -t hello-world:1.0 .: Builds a docker image from Dockerfile with the name hello-world with version 1.0 in the current directory. The '.' specifies the directory
- docker run hello-world:1.0: Creates and instantiates a container from the specified image hello-world
- docker rm -f 1244e706371e: Removes the container specified by the container id in the argument
- docker ps: lists all the currently running containers
- docker logs app: displays the logs of the container specified in the argument, in this case 'app'
- docker images: lists all the images created with their ID and when they were created.
- docker stop containerName/Id: stops one or more containers
- docker rm -f containerName/Id: forcefully deletes the container

For the next part, a multi container docker application was created and run. As the name suggests, these are applications which have multiple docker containers, interacting and communicating with each other. Each of these containers consists of a component of the application running. Multi container docker applications can be managed using docker-compose. In order to communicate the containers need to be on the same network. They communicate over a virtual network created by Docker. Each of the containers in the network are given an IP address through which they can address each other. The commands used for this task are as follows:

- docker pull mysql: pulling official image of mysql
- docker build -t my-web-app:1.0 .: building docker image from docker file
- docker run --name app -d my-web-app:1.0: running application container
- docker run --name app-db -d -e MYSQL_ROOT_PASSWORD=password -e MYSQL_DATABASE=myDB mysql: running database container
- docker run -d -p 8081:8080 my-web-app:1.0: connecting container to port
- docker network create app-network: command to create own network
- docker network connect app-network app-db: connects app container with db container on same network

The next task required us to deploy a container using docker and kubernetes. Basic definitions include:

Pod: Pods are the smallest operational computing units in Kubernetes which consist of single or multiple running containers sharing network and storage services.

Service: It is a concept that specifies a rational collection of pods and a protocol for accessing them. This model can also be referred to as micro services.

Node: Based on the cluster, a node is either a physical or virtual service machine containing several pods. Pod scheduling within the cluster's nodes is handled by Kubernetes control plane.

Deployment: Deployment allows explicit updates for pods and replica sets. To ensure availability of at least one implementation of the application as per client queries, deployment executes application replicas to restore or substitute instances that fail.

It is the process of running multiple instances of Pod and simultaneously maintaining a consistent amount of pods in order to maintain application accessibility for users if a pod dies or becomes unavailable.

The commands used to complete the task are as follows:

- gcloud config set project projectID: sets to current project
- gcloud config set compute/zone us-central1-a: sets time zone for current project
- gcloud container clusters create gk-cluster --num-nodes=1: creates a cluster with one node
- gcloud container clusters get-credentials gk-cluster: retrieves credentials for cluster deployment
- kubectl create deployment web-server --image=us.gcr.io/project-milestone-1/cad-site:version1: Deployment using gcr image
- kubectl expose deployment web-server --type LoadBalancer --port 80 --target-port 80: allows exposure of web server to port 80, enabling website access.
- kubectl get pods: Displays active pods
- kubectl get service: displays external IP address and other information

Here is a list of all kubernetes services and a brief description about each of them:

Cluster IP: Requests are sent internally to a stable IP address from internal clients. It is a default service.

Node Port: This service involves a client making requests to a node's IP address using one or more nodePort values defined by the Service.

ExternalName: In this type of service, the DNS name of a Service is used by internal clients as a proxy for an external DNS name.

LoadBalancer: This type of service involves a client submitting a request to a network load balancer's Ip address.