Mini-Project 1 – Deploying Your First Application in Cloud

(Due by Midnight, Feb. 15)

1 Summary

In this assignment, you will build a simple *two-tier* live chat in-cloud application and deploy it in Google cloud. You will start with a traditional "native" setup and convert it to a containerized setup. To finish this project, you need to know how to operate docker containers and how to build container images.

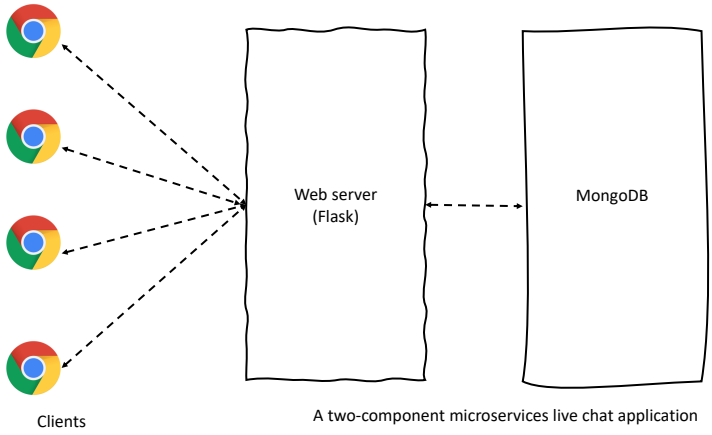

Figure 1: Architecture of the Live Chat application.

2 The Application Scenario

The application scenario in this mini-project is a simple **live chat** service, where comments are added to the page for all connected clients. As shown in Figure 1, on the *client* side (e.g., browsers), it leverages Javascript, SocketIO, and JQuery to communicate messages with the web server. Once the web server (one component of the live chat application implemented using Python Flask) receives a message, it stores the message to the Mongodb database server (the other component of the application).

Please finish the following tasks to build this application step by step:

3 Environment Setup

Note that, this project should be conducted with the following configurations with the least incompatibility hassles: (1) Creating **two** GCP VM instances – each with 2 vCPUs, 4 GB memory, and 20 GB hard disk. Note that, create these two VMs in the same region and same zone. (2) **Important!** Choose "Ubuntu 18.04 LTS" to provision the GCP VM instead of the default one – **make changes under Boot disk**. (3) Firewall: check "Allow HTTP traffic" (as you will access via http connections and port 8080). In addition, under "VPC network->Firewall", add port 8080 to "default-allow-http", as shown in Figure 2. We name one VM the **webserver** VM, and the other the **database** VM.

Note that, if you don't work on those VMs, stop them!! Otherwise, your GCP credits will burn out quickly and you have to pay for those resources for other assignments.

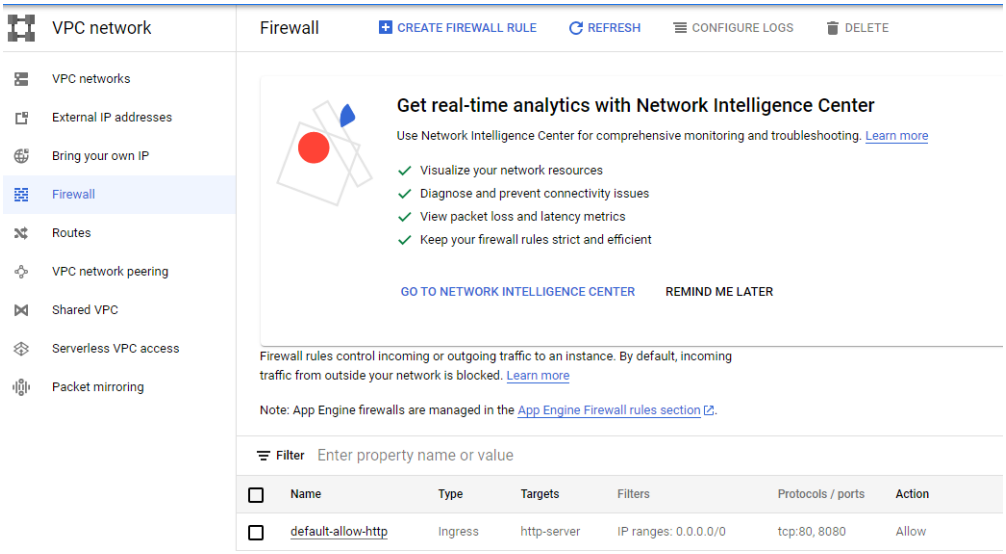

Figure 2: Firewall rules.

4 Task 1: Building and Deploying in the Native

4.1 MongoDB Installation

We first need a MongoDB, which will be connected by the live chat web server to store messages. Please follow the steps in this document [1] to finish the installation and validation of your mongoDB **using the database VM**. Make sure everything works well for your mongoDB before you move to the next step.

4.2 Web Server Installation

The web server is implemented using Python Flask [2]. Flask is a micro web framework written in Python. You can easily enable a web services with it.

Download the source code (to **the webserver VM**):
$git clone https://github.com/huilu1111/miniproject1

Please follow the project's README to install all python libraries. **Note that:** you also need to change the following file: app/config.py by modifying the default IP address '0.0.0.0' to the **internal** IP address of the database VM – thus the web service can connect to the Mongodb service. Please go to your GCP instance page to find out the internal IP address of the database VM. Finally, you can start the web service. If everything works fine, your live chat web service is up and ready for use as shown in Figure 3.

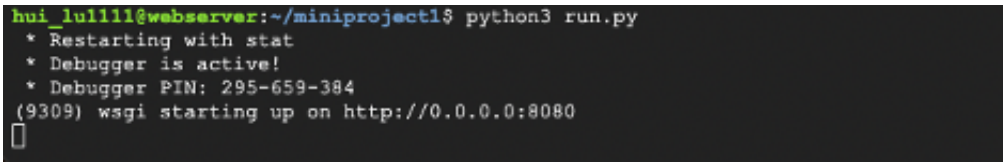

Figure 3: Running the web service.

Just go to any browser and enter the url – http://Webserver_VM's_external_IP_address:8080. For example, you can access my live chat service via http://35.199.55.105:8080/. "http://35.199.55.105" is my

webserver VM's external IP address. Note that, please go to your GCP instance page to find out the external IP address of your webserver VM. Use **http** instead of **https**.

Please get familiar with the live chat web service. You can access using multiple sessions (from different browsers). All users should see the same messages at the same time.

5 Task 2: Containerization

You probably already see that, there are too many steps to install those components. If you move to another machine, you probably need to do them again. To simplify these installation procedures, we can **containerize** these two applications – the live chat web server and the MongoDB database – by building container images [3]. You will learn to how to write Dockerfile files for building container images in this task.

Before that, please refer to this instruction [11] for Docker installation. You need to install Docker on **both** VMs. In addition, you have to get familiar with Docker basic operations by referring to [12]. Make sure you are familiar with Docker operations before moving to the next step.

Please install docker container and download the source code before your start (again on both VMs):
$git clone https://github.com/huilu1111/miniproject1-container

5.1 MongoDB Containerization

Perform the following operations on your database VM:
$cd miniproject1-container/mongodb

We have provided an empty Dockerfile file. **What you need to do is** to complete the Dockerfile file. In order to do this, you need to learn how to write a Dockerfile file. Please refer to [3] for **self-learning** – do your homework to be familiar with these techniques.

We also provide a simple Makefile. When you type down "sudo make build", docker should build the mongoDB container image for you. **Note that:** If `make` is not installed, please install it first: `sudo apt install make`. If your Dockerfile is correctly written, after building, the image will be stored in the local host. You can use "sudo docker image list" to check.

Similarly, when you type down "sudo make run", docker will run a mongoDB container instance from the mongoDB container image for you. Again, if the container image is correctly built (from your Dockerfile), a mongoDB container instance will be up and running.

You can use similar approaches (as in the above native case) to check the containerized mongoDB. Since we map the container's 27017 port to the host (i.e., the virtual machine), you can also access the mongoDB in the host. Use "netstat -an | grep 27017" in the host to check whether the 27017 port is opened.

Hint: 1. Basically, you just find a way to put those installation steps (in task 1) in the Dockerfile file; 2. Don't forget to add an ENTRYPOINT; 3. Please stop any previous running MongoDB (as they share the same 27017 port) – e.g., one brutal way to stop the running Mongodb: `sudo killall mongod`.

5.2 Web Server Containerization

Perform the following operations on your webserver VM. Make sure you have installed Docker on this VM
$git clone https://github.com/huilu1111/miniproject1-container

$cd miniproject1-container/webserver

Similar to MongoDB containerization, **what you need to do** is to complete the Dockerfile file for the web server. You will find that we have included the source code of the live chat web service under the same folder.

In addition, you have to change one configuration in " app/config.py": from "0.0.0.0" to the internal IP address of your database VM.

Similarly, when you type down "sudo make build", docker will build the web server container image. When you type down "sudo make run", docker will run a live chat web server container instance from for you. Note that, you must first run the MongoDB container instance before you start the web server container.

We expect to see that now your live chat service is up and running and can access using the same way as in the above native case.

6 Submission & Grading

Submit your solutions — **two Dockerfiles** — on the BrightSpace as separate files. You will also schedule an appointment with GAs to have a demo one day after the deadline (more details will be released soon). You will not have points if you fail to do so. We will grade your assignment based on the submissions and the demo. Basically, you should not only follow the instructions to do your assignment but also fully understand what you are doing.

During the demo, you will be asked to:

- Run the MongoDB correctly in the native case. – 5 points
- Run the web server correctly in the native case. – 5 points
- The live chat service can be accessed externally (using browsers) – 10 points
- Run the MongoDB correctly in the container case. – 10 points
- Run the web server correctly in the container case. – 10 points
- The live chat service can be accessed externally (using browsers) – 10 points

References

[1] How to install mongodb on Ubuntu 18. `https://www.cs.binghamton.edu/~huilu/cs552sp23/How_to_Install_MongoDB_on_Ubuntu_18.pdf`.

[2] Flask. `https://flask.palletsprojects.com/en/2.0.x/`.

[3] Best practices for writing Dockerfiles. https://docs.docker.com/develop/develop-images/dockerfile_best-practices/.

[4] minikube start. https://minikube.sigs.k8s.io/docs/start/.

[5] Deployments. https://kubernetes.io/docs/concepts/workloads/controllers/deployment/.

[6] Volumes. https://cloud.google.com/kubernetes-engine/docs/how-to/volumes.

[7] kubectl Cheat Sheet. https://kubernetes.io/docs/reference/kubectl/cheatsheet/.

[8] Service. https://kubernetes.io/docs/concepts/services-networking/service/.

[9] How to Run Locally Built Docker Images in Kubernetes. https://medium.com/swlh/how-to-run-locally-built-docker-images-in-kubernetes-b28fbc32cc1d.

[10] Use Port Forwarding to Access Applications in a Cluster. https://kubernetes.io/docs/tasks/access-application-cluster/port-forward-access-application-cluster/.

[11] Docker Installation. https://docs.docker.com/engine/install/ubuntu/.

[12] Running your first container. https://github.com/docker/labs/blob/master/beginner/chapters/alpine.md.