Mini-Project 2 – Let's build microservices

(Due by Midnight, Monday, Nov. 8, 10% Grade)

1 Summary

In this assignment, you will get familiar with the concept of microservices and orchestration via building a simple *two-tier* live chat microservices and deploy it in Minikube. You will start with a traditional native setup and convert it step by step to a microservices setup running with orchestration tools. To finish this project, you need to know how to operate containers, how to build container images, how to operate a Minikube, and how to write YAML configuration files for Minikube deployment.

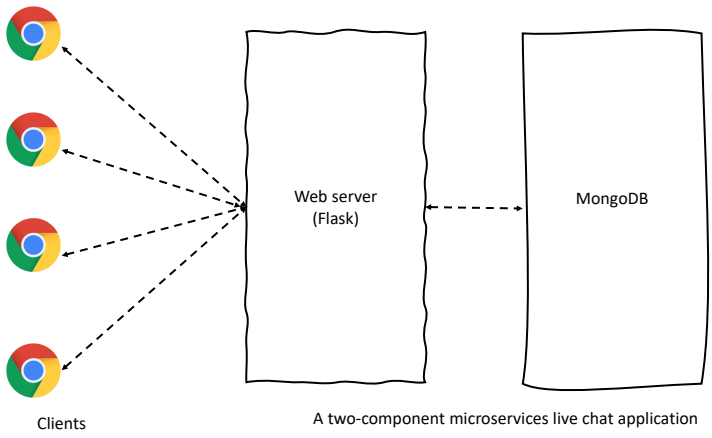

Figure 1: Architecture of the Live Chat application.

2 The Microservices scenario

The microservices scenario in this assignment 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 microservices implemented using Python Flask) receives a message, it stores the message to the Mongodb database server (the other component of the microservices).

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 least incompatibility hassles: (1) Using a GCP VM instance (e.g., 2 vCPUs, 4 GB memory, and 20 GB hard disk); (2) **Important!** Choose "Ubuntu 18.04 LTS" to provision the GCP VM; (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.

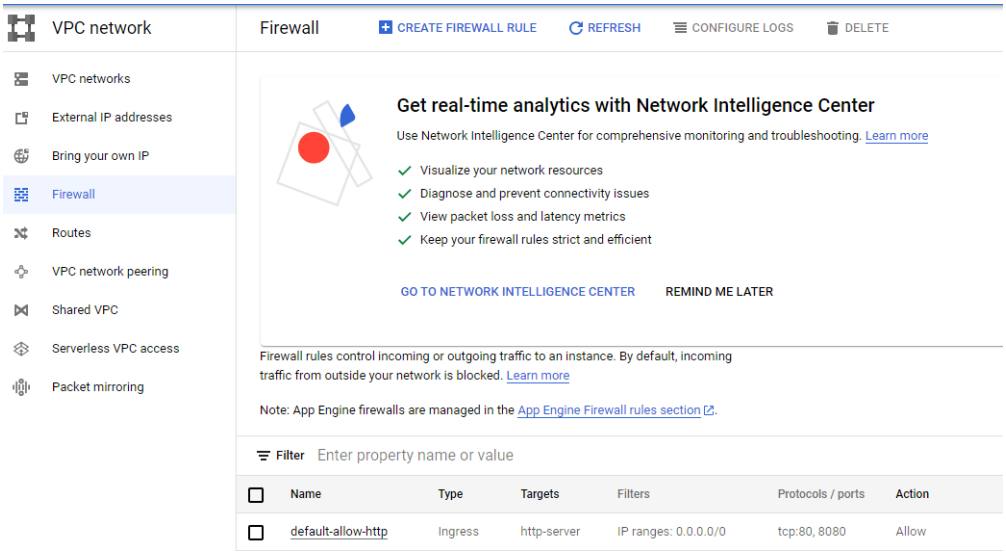

Figure 2: Firewall rules.

4 Task 1: Building 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. 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, just like our live chat microservices.

Download the source code:
$git clone https://github.com/huilu1111/miniproject2

Please follow the project's README to install all python libraries and then you can start the web service. If everything works fine, your live chat web service is up and ready for use.

Just go to any browser and enter the url – VM's_external_IP_address:8080. For example, you can access my live chat service via http://35.193.167.219:8080/. Note that, please go to your GCP instance page to find out your external IP address.

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 simply 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.

Please install docker container and download the source code before your start:
$sudo apt install docker.io
$git clone https://github.com/huilu1111/miniproject2-container

5.1 MongoDB Containerization

$cd miniproject2-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 these techniques.

We also provide a simple Makefile. When you type down "sudo make build", docker should build the mongoDB container image for you. 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).

5.2 Web Server Containerization

$cd miniproject2-con/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, we have changed one configuration in "app/config.py": the IP address of 'host' has been changed from "0.0.0.0" (in the native case) to "172.17.0.1". Make sure that "172.17.0.1" is your docker0 bridge's IP address (use "ifconfig" to check docker0's IP address in the host). Figure out why we need this change. Note that, if your host's docker0's IP address is different than "172.17.0.1", use your docker0's IP address as the 'host' configuration in "app/config.py".

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 Task 3: Orchestration

It is time to deploy your live chat microservices using orchestration tools, such as Kubernetes. In this task, we use miniKube [4].

Please follow [4] to install and start miniKube including the following steps (note that, miniKube itself runs insider docker containers and isolates it from your local host):

$curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
$sudo install minikube-linux-amd64 /usr/local/bin/minikube
$minikube start #this step may take a while to finish; you may need to ready the dumpped information carefully if there is any error and fix it.
$alias kubectl="minikube kubectl --"

Now the minikube cluster is ready and you can use "kubectl" to interact with the minikube cluster. **What you need to do** is to write **four** YAML files: (1) one deployment object for Mongodb; (2) one service object to expose Mongodb; (3) one deployment for web server; and (4) one service object to expose web server.

$git clone https://github.com/huilu1111/miniproject2-minikube

Specifically, for the Mongodb deployment, it is associated with a pod with one MongoDB container (using the "mongodb:v1" image from task 2). The replica number for this pod is **one**. For the web server depoyment, it is associated with a pod with one live chat web server container (using the "flaskweb:v1" image). The replica number for this pod is **three**.

Note that, to use your own build images, such as "mongodb:v1" and "flaskweb:v1", you need to point your shell to minikube's docker-daemon. To do so, you have to execute the following commands (more information can be found here [9]):
$minikube docker-env
$eval $(minikube -p minikube docker-env)

6.1 Deploy MongoDB

(1) First, rebuild "mongodb:v1" with your minikube's local docker. Go to "miniproject2-container/mongodb/" (finished in task 2) and "make build". Don't use sudo; otherwise you will use the host's docker. After building, use "docker image list" to check whether the "mongodb:v1" image is there.

(2) Once the mongodb's container image is ready, you can finish writing the "mongodb-deploy.yaml" file according to the above requirements – one pod with one MongoDB container from the "mongodb:v1" image with the replica being one. You can learn how to write deployment YAML from this document [5] and tons of other examples from online. In addition, you also need to use volumes in your deployment YAML to mount MongoDB container's internal "/data/db" to a local volume mount point. You may refer to this document [6].

(3) Now you cam create the Deployment by running the following command:
$kubectl apply -f mongodb-deploy.yaml

You can use a bunch of kubectl commands to check the status of your deployment [7]. I found the following several commands especially helpful: (1) kubectl get deployment; (2) kubectl get pods; (3) kubectl get all. To get more detailed information for debugging, you can use (4) kubectl describe deployments; (5) kubectl describe pods.

The success of your deployment will show "1/1" READY as shown in Figure 3.
(**Hint:** For any failed deployments, please delete them timely to make the minikube clean – e.g., using kubectl delete -f xxx.yaml)

```
hui_lu1111@instance-2:~/miniproject2-minikube$ kubectl get deployment
NAME                 READY   UP-TO-DATE   AVAILABLE   AGE
mongodb-deployment   1/1     1            1           55s
hui_lu1111@instance-2:~/miniproject2-minikube$ kubectl get pods
NAME                                  READY   STATUS    RESTARTS   AGE
mongodb-deployment-7bd97bc45-fxjsj    1/1     Running   0          59s
```

Figure 3: MongoDB Deployment.

(4) Finish writing the "serviceDB.yaml" to expose the MongoDB pod to the cluster. You may refer to this document [8]. Note that the "targetPort" for MongoDB is 27017.

(5) Create the Service by running the following command:
$kubectl apply -f serviceDB.yaml

The success of your deployment will show a new service (e.g., db-service) is up with a CLUSTER-IP (e.g., 10.109.186.21) as shown in Figure 4. Later, the web server can use this exposed cluster-IP and the 27017 port to access MongoDB.

```
hui_lu1111@instance-2:~/miniproject2-minikube$ kubectl get service
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)     AGE
db-service   ClusterIP   10.109.186.21   <none>        27017/TCP   3s
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP     96m
```

Figure 4: MongoDB Service.

You can further use "kubectl describe service db-service" to check the detailed port mapping, e.g., from 172.17.0.3:27017 (i.e., MongoDB container's ephemeral POD IP address) the to 10.109.186.21:27017 (i.e., its static cluster-wide IP).

6.2 Deploy Web Server

(1) Similarly, rebuild the "flaskweb:v1" image. Go to "miniproject2-container/webserver/". Since we know the service IP of the MongoDB pod (e.g., 10.109.186.21 as shown in Figure 4). Copy this IP address (i.e., yours should be different than what is shown here) to app/config.py and replace the 'host' IP address. Afterwards, "make build"and then use "docker image list" to check whether the "flaskweb:v1" image is there.

(2) Once the web server's container image is ready, you can finish writing the "webserver-deploy.yaml" file according to the above requirements – one pod with one web server container from the "webserver-deploy.yaml" image with the replica being **three**. Again, you can learn how to write deployment YAML

from this document [5].

(3) Now you cam create the Deployment by running the following command:
$kubectl apply -f webserver-deploy.yaml

The success of your deployment now will show "3/3" READY as shown in Figure 5. It is because we set the replica number to three.

```
hui_lu1111@instance-2:~/miniproject2-minikube$ kubectl get deployment
NAME                 READY   UP-TO-DATE   AVAILABLE   AGE
mongodb-deployment   1/1     1            1           54m
webserver-deployment   3/3     3            3           10s
```

Figure 5: Web Server Deployment.

(4) Finish writing the "serviceWEB.yaml" to expose the Web Server pod to the cluster. You may refer to this document [8]. Note that the "targetPort" for Web Server is 8080. You need to specify the "NodePort" type (refer [8] for Type NodePort).

(5) Create the Service by running the following command:
$kubectl apply -f serviceWEB.yaml

The success of your deployment will show a new service (e.g., web-service) is up with a CLUSTER-IP (e.g., 10.108.65.85) as shown in Figure 6. Note that, the TYPE shows NodePort.

```
hui_lu1111@instance-2:~/miniproject2-minikube$ kubectl get services
NAME          TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
db-service    ClusterIP   10.109.186.21   <none>        27017/TCP        45m
kubernetes    ClusterIP   10.96.0.1       <none>        443/TCP          141m
web-service   NodePort    10.108.65.85    <none>        8080:30100/TCP   6s
```

Figure 6: Web Server Service.

As you create a NodePort, you can access the web server via the "host". However, here the "host" means your minikube host (not your GCP VM host). Your minikube running inside a container has a different local IP address than the VM host.

(6) One way you can do is to "ssh" to the minikube container. You use "sudo docker ps" to list the minikube container (note that, you now use sudo to list containers running inside your GCP VM). Then "sudo docker exec -it minikube_container_id bash". Once you attach to the running minikube container, you can access the web server locally using its IP address, e.g., wget 192.168.49.2:30100. Assume 192.168.49.2 is your minikube container's IP address and 30100 is the NodePort in your web-service.
(**Hint:**As the minikube container does not include necessary tools, you need to install them such as apt update; apt install net-tools; apt install wget).

(7) What even more interesting is to access the live chat microserives externally – from your own browsers! You can achieve this using kubectl's "port-forward" function [10]. Before run the following command, make sure you exit the previous minikube container.

$kubectl port-forward –address 0.0.0.0 service/web-service 8080:8080

The above command will block your current terminal, while running. With this, you can use your own browser and the external IP address of you GCP VM plus 8080 port to access the live chat microservices.

7 Submission & Grading

Submit your solutions — **two Dockerfiles and four YAML files** — on the BrightSpace as separate files. You will also schedule an appointment with TA to have a demo before the deadline. **Notice that it is your responsibility to schedule the demo time. You will lose 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. – 10 points
- Run the web server correctly in the native case. – 10 points
- The live chat service can be accessed externally (using browsers) – 5 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) – 5 points
- Deploy the MongoDB object correctly in the miniKube case. – 10 points
- Deploy the MongoDB service object correctly in the miniKube case. – 10 points
- Deploy the web server object correctly in the miniKube case. – 10 points
- Deploy the web server service object correctly in the miniKube 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/slidesfall2021/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/.