Mini-Project 2 – Let's build Kubernetes

(Due by Midnight, March 10)

---

1 Summary

In this assignment, you will first enable a **three-node** Kubernetes cluster, as shown in Figure 1. Thus, you can play with a real Kubernetes cluster and practice its main functionalities. In addition, you will deploy mini-project1' two-tier Chat application (a web server and a database server) in the Kubernetes cluster. To complete this assignment, you will need to learn how to write YAML configuration files to correctly describe your applications with all required objects.

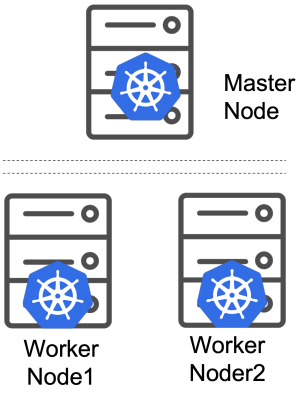

Figure 1: Architecture of the three-node Kubernetes cluster.

2 Environmental Setup

This project should be conducted in GCP with the following configurations with the least incompatibility hassles:

(1) Create **three** GCP VM instances – each with 2 vCPUs, 4 GB memory, and 30 GB hard disk. Note that, create these three VMs in the same region and same zone. Also note that, the three VMs cost you **0.12** dollars per hour! Please stop them whenever you are not using them. VMs are charged at a per-hour granularity.

(2) **Important!** Choose "Ubuntu 20.04 LTS (x86/64)" to provision the GCP VMs instead of the default one – **make changes under Boot disk**.

(3) Firewall: check "Allow HTTP/HTTPs traffic". In addition, under "VPC network->Firewall", add port 30100 to "default-allow-http", as shown in Figure 2.

Afterwards, please finish the following tasks to build the Kubernetes cluster step by step:

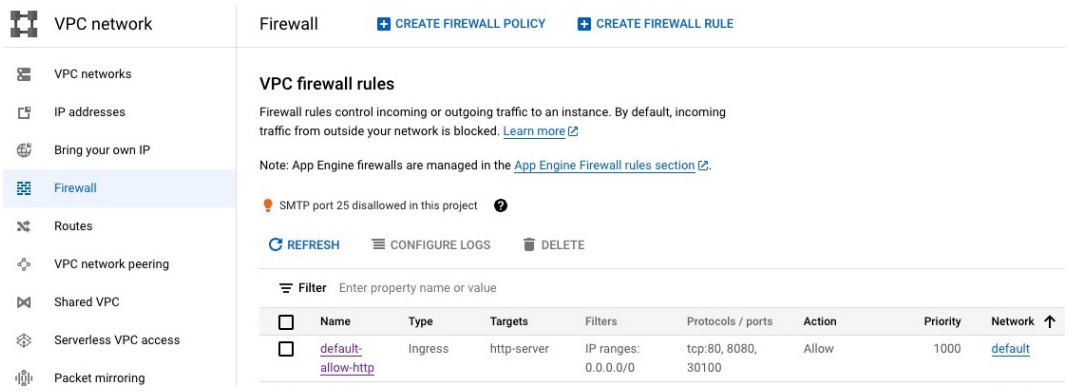

Figure 2: Firewall rules.

3 Task 1: Build the Kubernetes Cluster

For your three VMs, we name them Master Node, Worker Node1, and Worker Node2. Find out their **internal IP addresses** from your GCP panel. For example, the following are my examples. You should use your VMs' IP addresses!

Master Node: – k8-master – IP: 10.150.0.7
Worker Node 1: – k8-worker1 – IP: 10.150.0.8
Worker Node 2: – k8-worker2 – IP: 10.150.0.9

3.1 Set up the hostname and update the host files

First of all, you need to log into each of the nodes and set up the hostname as shown below:

For **Master Node**:
$sudo hostnamectl set-hostname k8-master

For **Worker Node1**:
$sudo hostnamectl set-hostname k8-worker1

For **Worker Node2**:
$sudo hostnamectl set-hostname k8-worker2

Additionally, update the **/etc/hosts** file for the 3 nodes. More specifically, add the below three lines to the end of the /etc/hosts file (again, use your VMs' internal IP addresses).

10.150.0.7 k8-master
10.150.0.8 k8-worker1
10.150.0.9 k8-worker2

To verify, use the `ping` command to ping k8-master, k8-worker1, and k8-worker2, in each of the three nodes. They should be "pingable" with responses.

$ping k8-master
$ping k8-worker1

$ping k8-worker2

3.2 Install Docker on Master and Worker Nodes

Please refer back to your mini project1 for the Docker container installation on the three nodes: https://docs.docker.com/engine/install/ubuntu/.

Note that Kubernetes use "cri" as the container runtime, while the Docker container disables it by default. Use the following method to fix it:
$sudo rm /etc/containerd/config.toml
$sudo systemctl restart containerd

3.3 Configure the Kubernetes repository on Master and Worker Nodes

Before you get started in configuring the Kubernetes repository on your nodes, a few dependencies are essential. Run the command below to install the requisite dependencies:
$sudo apt-get install software-properties-common apt-transport-https curl

Thereafter, add Kubernetes GPG key as shown:
$curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add

To append the repository run the command:
$sudo apt-add-repository "deb http://apt.kubernetes.io/ kubernetes-xenial main"

3.4 Disable OS swap and install kubeadm on Master and Worker Nodes

To deploy multiple nodes on the Kubernetes cluster, you'll first need to install the kubeadm package. However, the official Kubernetes site recommends that you first disable the OS swap feature on **all the three nodes**.

To disable swap on both the master and slave nodes, execute the following command:
$sudo swapoff -a

Proceed to install the kubeadm package following this command:
$sudo apt-get install kubeadm -y

Once you have successfully installed the kubeadm package, feel free to verify its version as shown:
$kubeadm version

3.5 Create your Kubernetes cluster using Kubeadm from Master Node

To fire up your cluster, log in and start Kubernetes on your system's **Master node** using kubeadm as illustrated in the below. Before you execute the following command, let's understand it more:

The `--apiserver-advertise-address` flag specifies the IP that the API server is listening on. If this is not specified, the default network interface will be assumed. Here, you should speficiy **your Master Node's internal IP address.** The `--pod-network-cidr=172.16.0.0/16` flag specifies an IP address range for the pod network, when set, CIDRs will automatically be allocated to every node. You should

specify this if there's no conflict between the preferred pod network and some of the nodes in your LAN.

$sudo kubeadm init --apiserver-advertise-address=10.150.0.7 --pod-network-cidr=172.16.0.0/16

It may take some time to complete. In the end, you should see the following messages:

```
To start using your cluster, you need to run the following as a regular user:

mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

After applying those commands, your Master Node should be deployed. To verify that, run the command:
$kubectl get nodes

With the following expected output:

```
NAME       STATUS   ROLES          AGE VERSION
k8-master NotReady control-plane 29m v1.26.1
```

You can also use the following command to check the cluster information:
$kubectl cluster-info

You can vefiry that all kubernetes services are healthy:
$kubectl get --raw=/readyz?verbose

3.6 Join all Worker Nodes in the Cluster

Make sure you finish the above steps in Section 3.2, Section 3.3, and Section 3.4 for each of the **Worker Nodes**. Then, run the following command in **Master Node** to generate the token and cert:

$kubeadm token create --print-join-command

It will output the token and cert, for example, as follows:

```
kubeadm join 10.150.0.10:6443 --token w7m39h.9vrh52io8tpqhuzk
--discovery-token-ca-cert-hash
sha256:93e88b4478a2f769b81d498aafd312325f517311b09aab96ed047738b9f6208c
```

Finally, run the following command to join each Worker Node to the cluster:

$ sudo kubeadm join 10.150.0.7:6443 --token w7m39h.9vrh52io8tpqhuzk --discovery-token-ca-cert-hash sha256:93e88b4478a2f769b81d498aafd312325f517311b09aab96ed047738b9f6208c

Note that, don't simply copy and paste the above command. **Use your Master Node's IP address and the token and hash value**.

After you apply those commands, to verify, run the command:
$kubectl get nodes

With the following expected output:

```
NAME       STATUS    ROLES          AGE VERSION
k8-master NotReady control-plane 29m v1.26.1
k8-worker1 NotReady <none>        12m v1.26.1
k8-worker2 NotReady <none>        21s v1.26.1
```

3.7 Deploy overlay network

Kubeadm does not configure any network plugin. You need to install a network plugin of your choice. We will use the Flannel network plugin for this setup. On **Master Node**:

$wget https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml

You need to change the default CDIR "10.244.0.0/16" in the downloaded `kube-flannel.yml` file to the one you just specified, i.e., "172.16.0.0/16". Then apply the flannel network plugin:

$kubectl apply -f kube-flannel.yml

To check the status of the overlay network creation:
$kubectl get pods –all-namespaces

Figure 3: Deploying overlay networks.

As shown in Figure 3, all the `flannel` pods and `coredns` pods should become "Running" after a while.

4 Task 2: Deploy Applications in the Kubernetes Cluster

Your three-node Kubernetes cluster should be enabled and working well at this stage. In this task, you will deploy the two-tier Chat application of your mini-project1 onto the Kubernetes cluster.

What you need to do is to write **four** YAML files: (1) one deployment object for deploying Mongodb; (2) one service object to expose Mongodb; (3) one deployment for deploying the flask web server; and (4) one service object to expose the web server. You should learn – **by yourself** – how to write deployment and services YAML files from our slides, this document [1], and other online examples.

(**Hints:** In the deployment files, you need to specify the mongodb and flask container images for the PODs. There are several ways to make locally-built container images accessible by the Kubernetes cluster: (1) create a local container register or (2) push your locally-built container images to https://hub.docker.com/. I found method (2) easier to be enabled. You can refer to this document [5] for more details.)

Task 2.1 Write a **"mongodb-deploy.yaml"** file with the custom-built MongoDB container image – refer

back to mini-project1 for the container image creation and [5] for making the container image accessible by your Kubernetes cluster. Set the replica to **one**. 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 [2].

Now you can create the Deployment in the Kubernetes cluster 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 [3]. 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, and even (6) kubectl logs PODNAME.

The success of your deployment will show "1/1" READY as shown in Figure 4.
(**Hint:** For any failed deployment, please delete them to make your cluster 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 4: MongoDB Deployment.

Task 2.2 Write a "serviceDB.yaml" file to expose the MongoDB POD to the cluster. You may refer to our slides or this document [4]. Note that the "targetPort" for MongoDB is 27017, which is the default port that the MongoDB is listening to.

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 5. 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 5: MongoDB Service.

You can further use "kubectl describe service db-service" to check the detailed port mapping, e.g., from 172.16.86.133:27017 (i.e., MongoDB container's ephemeral POD IP address) the to 10.109.186.21:27017 (i.e., its static cluster-wide IP).

Task 2.3 Similarly, write a "webserver-deploy.yaml" file with the replica being **three**. Again, you can learn how to write deployment YAML from this document [1]. Note that, you have to build the webserver's container image first (`flaskweb:v1`) and make it accessible by the Kubernetes cluster (e.g., via [5]). In order for the webserver to connect to the Mongodb, you need to put the Mongodb's ClusterIP (e.g., 10.109.186.21:27017) in Webserver's configuration file (e.g., app/config.py)

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 6. 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 6: Web Server Deployment.

Task 2.4 Write a "serviceWEB.yaml" file to expose the Web Server pod to the cluster and the external world. You may refer to our slides or this document [4]. Note that the "targetPort" for Web Server is 8080 and the nodePort should be 30100; you need to specify the "NodePort" type (refer to [4] for Type NodePort).

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 7. 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 7: Web Server Service.

As you create a NodePort, you can access the web server externally. You should now access your Chat application using the Master Node's external IP plus port 30100, e.g., http://34.150.213.189:30100/. In fact, you can also use the other two Worker Nodes' external IP addresses plus port 32000 to access the Chat application. Why is that?

5 Submission & Grading

Submit your solutions — **four YAML files** — to BrightSpace as separate files. You will also schedule an appointment with GAs to have a demo after the deadline – more details will be announced shortly. **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. You should not only follow the instructions to finish the assignment but also fully understand what you are doing.

- The three-node Kubernetes cluster is successfully enabled. – 30 points
- Each YAML file should be correctly implemented. – 5 points for each, total 20 points
- Deploy the MongoDB object correctly in the Kubernetes cluster. – 10 points

- Deploy the MongoDB service object correctly in the Kubernetes cluster. – 10 points
- Deploy the web server object correctly in the the Kubernetes cluster. – 10 points
- Deploy the web server service object correctly in the the Kubernetes cluster. – 10 points
- The live chat service can be accessed externally (using browsers) – 10 points

References

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

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

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

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

[5] Manage repositories. https://docs.docker.com/docker-hub/repos/.