# Assignment 2: Kubernetes

## A. Overview

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 Assignment 1' 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.

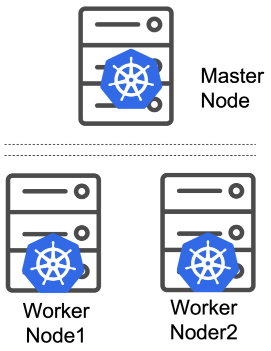


Figure 1: Architecture of the three-node Kubernetes cluster.

## B. Environmental Setup

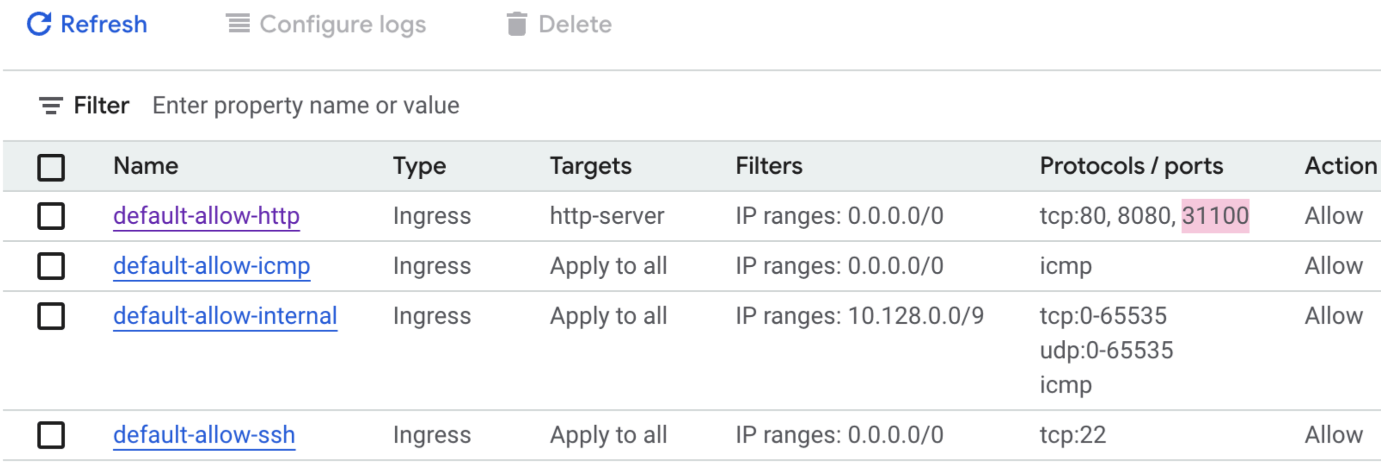
This project should be conducted in GCP with the following configurations with the least incompatibility hassles:

A. 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.

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

C. Firewall: check "Allow HTTP/HTTPs traffic". In addition, under "VPC network->Firewall", add port 31100 to "default-allow-http", as shown in Figure 2.

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



| Name | Type | Targets | Filters | Protocols / ports | Action |
|---|---|---|---|---|---|
| default-allow-http | Ingress | http-server | IP ranges: 0.0.0.0/0 | tcp:80, 8080, 31100 | Allow |
| default-allow-icmp | Ingress | Apply to all | IP ranges: 0.0.0.0/0 | icmp | Allow |
| default-allow-internal | Ingress | Apply to all | IP ranges: 10.128.0.0/9 | tcp:0-65535 udp:0-65535 icmp | Allow |
| default-allow-ssh | Ingress | Apply to all | IP ranges: 0.0.0.0/0 | tcp:22 | Allow |

Figure 2: Firewall rules.

## C. 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

### a. 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
```

## b. Install Docker on Master and Worker Nodes

Please refer back to your Assignment 1 for the Docker container installation on the three nodes.

Note that Kubernetes uses "cri" as the container runtime, while the Docker container disables it by default. Use the following method to fix it:

```
all>$ sudo rm /etc/containerd/config.toml
all>$ sudo systemctl restart containerd
```

## c. 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:

```
all>$ sudo apt-get install -y apt-transport-https ca-certificates curl gpg
```

Thereafter, add Kubernetes GPG key as shown:

```
all>$ curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.32/deb/Release.key | sudo gpg --dearmor -o
/etc/apt/keyrings/kubernetes-apt-keyring.gpg
```

To append the repository run the command:

```
all>$ echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg]
https://pkgs.k8s.io/core:/stable:/v1.32/deb/ /' | sudo tee /etc/apt/sources.list.d/kubernetes.list
```

## d. 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:

```
all>$ sudo swapoff -a
```

Proceed to install the kubeadm package following this command:

```
all>$ sudo apt-get update
all>$ sudo apt-get install -y kubelet kubeadm kubectl
```

Mark hold for current version incase of version mismatches on further update:

```
all>$ sudo apt-mark hold kubelet kubeadm kubectl
```

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

```
all>$ kubeadm version
```

## e. 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 specify **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.

```
master>$ 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:

```
Your Kubernetes control-plane has initialized successfully!

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:

```
master>$ kubectl get nodes
```

With the following expected output:

```
NAME         STATUS     ROLES           AGE     VERSION
k8-master    NotReady   control-plane   3m52s   v1.32.3
```

You can also use the following command to check the cluster information:

```
master>$ kubectl cluster-info
```

You can verify that all kubernetes services are healthy:

```
master>$ kubectl get −−raw=/readyz?verbose
```

## f. Join all Worker Nodes in the Cluster

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

```
master>$ 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:

```
worker>$ sudo kubeadm join 10.150.0.10:6443 −−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**. You will get the info like the following:

```
This node has joined the cluster:
  * Certificate signing request was sent to apiserver and a response was received.
  * The Kubelet was informed of the new secure connection details.
```

After you apply those commands, to verify, run the command on k8-master:

```
master>$ kubectl get nodes
```

With the following expected output:

```
NAME          STATUS     ROLES            AGE     VERSION
k8-master     NotReady   control-plane    12m     v1.32.3
k8-worker1    NotReady   <none>           2m19s   v1.32.3
k8-worker2    NotReady   <none>           2m7s    v1.32.3
```

## g. Deploy overlay network

To enable flannel and overlay networks, we need to enable the br_netfilter module first. Run the following commands one by one on all three nodes:

```
all>$ sudo modprobe br_netfilter
all>$ echo "br_netfilter" | sudo tee /etc/modules-load.d/br_netfilter.conf
all>$ cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-iptables = 1
net.bridge.bridge-nf-call-ip6tables = 1
net.ipv4.ip_forward = 1
EOF
all>$ sudo sysctl --system
all>$ sudo systemctl daemon-reload
all>$ sudo systemctl restart kubelet
```

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**:

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

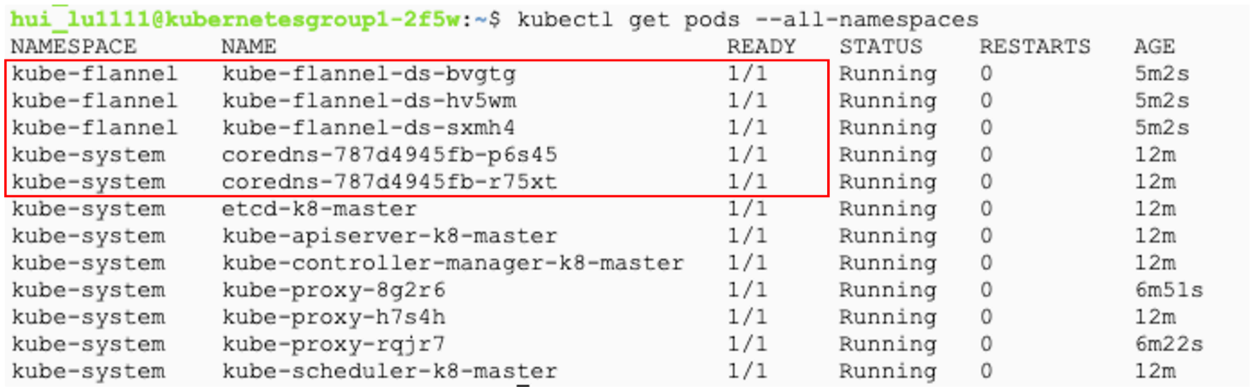
**You need to change the default CDIR "10.244.0.0/16" in the downloaded `kube-flannel.yml` file to the one you just specified, i.e., "172.16.0.0/16".**

Then apply the flannel network plugin:

```
master>$ kubectl apply -f kube-flannel.yml
```

To check the status of the overlay network creation:

```
master>$ kubectl get pods −−all-namespaces
```



```
hui_lu1111@kubernetesgroup1-2f5w:~$ kubectl get pods --all-namespaces
NAMESPACE     NAME                                  READY   STATUS    RESTARTS   AGE
kube-flannel  kube-flannel-ds-bvgtg                 1/1     Running   0          5m2s
kube-flannel  kube-flannel-ds-hv5wm                 1/1     Running   0          5m2s
kube-flannel  kube-flannel-ds-sxmh4                 1/1     Running   0          5m2s
kube-system   coredns-787d4945fb-p6s45              1/1     Running   0          12m
kube-system   coredns-787d4945fb-r75xt              1/1     Running   0          12m
kube-system   etcd-k8-master                        1/1     Running   0          12m
kube-system   kube-apiserver-k8-master              1/1     Running   0          12m
kube-system   kube-controller-manager-k8-master     1/1     Running   0          12m
kube-system   kube-proxy-8g2r6                      1/1     Running   0          6m51s
kube-system   kube-proxy-h7s4h                      1/1     Running   0          12m
kube-system   kube-proxy-rqjr7                      1/1     Running   0          6m22s
kube-system   kube-scheduler-k8-master              1/1     Running   0          12m
```

Figure 3: Deploying overlay networks.

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

```
master>$ kubectl get nodes
      NAME          STATUS    ROLES            AGE    VERSION
      k8-master     Ready     control-plane    47m    v1.32.3
      k8-worker1    Ready     <none>           37m    v1.32.3
      k8-worker2    Ready     <none>           37m    v1.32.3
```

## D. 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 Assignment 1 onto the Kubernetes cluster.
**What you need to do** is to write **four** YAML files:

- one deployment object for deploying Mongodb;
- one service object to expose Mongodb;
- one deployment for deploying the flask web server;
- 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, 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 this website. Method (2) is easier to achieve. You can refer to this document for more details.

**Task a.** Write a **"mongodb-deploy.yaml"** file with the custom-built MongoDB container image – refer back to Assignment 1 for the container image creation and use any one of the approaches mentioned above 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 deploy-ment YAML to mount MongoDB container's internal "/data/db" to a local volume mount point. You may refer to this document.

**Note:**
The MongoDB service comes with persistent state (backing user data)
In k8s, pods by default are scheduled based on node's resource availability. (It may even happen that 2 pods of mongoDB service are scheduled on different nodes!)
This behavior may cause issues as the data will be split between multiple worker nodes. This may lead to requests returning ambiguous results or even failing altogether.

For the scope of this assignment, you can choose one of the following methods:

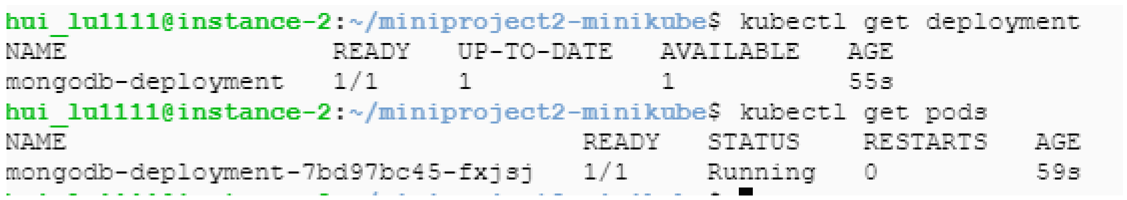
- Use PVC : This will eliminate all kinds of limitations from Kubernetes, all pods will have a shared synced persistent state. More information: You can refer to this document for persist-volumes and refer to this document for StatefulSet. Remember to create the folder to share on all worker nodes and make sure the privilege is set to 777. You will need to **provide an extra YAML file to the submission**.
- Use a hostPath volume (data is persisted, written on the node on which pod is run) In this case, we need to ensure all mongoDB pods are scheduled on the same node. Refer this document for setting NodeName field.

If the pods were previously deployed, clean mounted directories on both the working nodes before applying new deployment!

Now you can create the Deployment in the Kubernetes cluster by running the following command:
master>$ kubectl apply -f mongodb-deploy.yaml

You can use a bunch of kubectl commands to check the status of your deployment. I found the fol- lowing 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)



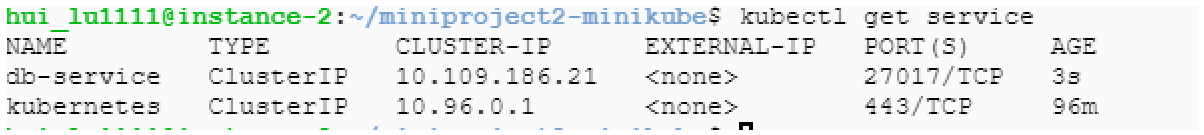
```
hui_lu1111@instance-2:~/miniproject2-minikube$ kubectl get deployment
NAME                    READY   UP-TO-DATE   AVAILABLE   AGE
mongodb-deployment      1/1     1            1           55s
hui_lu1111@instance-2:~/miniproject2-minikube$ kubectl get pods
NAME                                  READY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

Figure 4: MongoDB Deployment.

**Task b.** Write a “serviceDB.yaml” file to expose the MongoDB POD to the cluster. You may refer to our slides or this document. 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:
master>$ kubectl apply -f service-db.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.

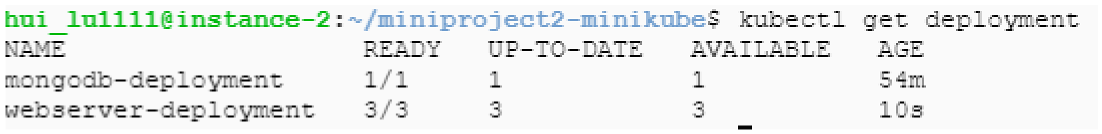
You can further use “kubectl describe service db-service” to check the detailed port mapping, e.g., from 172.16.86.133:27017 (i.e., MongoDB container’s ephemeral POD IP address) the to 10.109.186.21:27017 (i.e., its static cluster-wide IP).

**Task c.** Similarly, write a “webserver-deploy.yaml” file with the **replica being three**. Again, you can learn how to write deployment YAML from this document. Note that, you have to build the webserver's container image first (`flaskweb:v1`) and make it accessible by the Kubernetes cluster (detail mentioned in part D hints). 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 can create the Deployment by running the following command:

master>$ 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
```

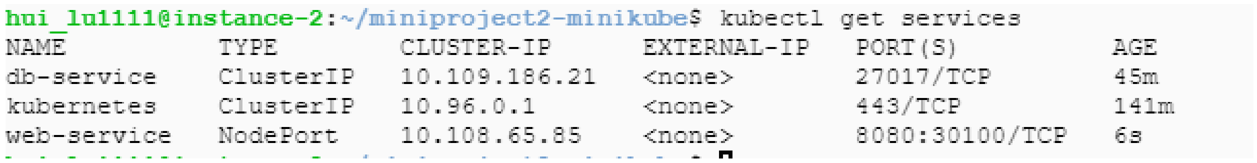
Figure 6: Web Server Deployment.

**Task d.** 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. Note that the "targetPort" for Web Server is 8080 and the nodePort should be 31100; you need to specify the "NodePort" type (refer to this document for Type NodePort).

Create the Service by running the following command:

master>$ kubectl apply -f service-web.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 31100, e.g., http://34.150.213.189:31100/.

# Submission & Grading

Submit your solutions: **Due on April-10-2025 11:59pm.**

1. **four YAML files (**or five YAML files if you choose PVC implementation) (naming: "mongodb-deploy.txt", "service-db.txt", "webserver-deploy.txt" and "service-web.txt" "PVC.txt") — on the BrightSpace as separate files. Must in *.txt format.
2. **one demo video link** —- upload your demo video to Google Drive, then share it on BrightSpace submission with a link left in the comment (make sure the link is accessible from the Binghamton University account). In case of a failed upload or other technique issues, we will give a 48 hours grace period. However, you must have all YAML files ready on Brightspace before the due date in case of a late penalty.

You will not get a grade if you fail to include the demo video. 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 and be able to explain what you are doing. We may also ask individual students to give live demos during TA's office hour.

The demo video should be under 15 minutes. The easiest way to capture your screen and a selfie window together is to use Zoom. Start a personal meeting, set up the camera and microphone, start screen sharing, and then click "more" → "record" → "record to this computer". When you are finished recording, click "stop recording" and then end this personal meeting. A pop-up window will appear showing the location of your recorded video file. Review the content and volume, but DO NOT EDIT the footage. Upload the video to Google Drive using your BU account and share it with a link.

The following process (each worth 10 points) must be shown in the demo video:

- The video has **a selfie window** to show that you are on your own and not under any help. Access your VMs.
- Show the three-node Kubernetes cluster is successfully enabled. Show the firewall page on GCP web portal with the current setting.
- Deploy the MongoDB deployment object correctly in the Kubernetes cluster.
- Deploy the MongoDB service object correctly in the Kubernetes cluster.
- **Explain your implementation of MongoDB's persistent state, and give reasons why you prefer the one you are currently using rather than another approach.**
- Deploy the webserver deployment object correctly in the Kubernetes cluster.
- Deploy the webserver service object correctly in the Kubernetes cluster.
- Show and explain the Kubernete deployment and service detailed information about MongoDB and web server (e.g., `kubectl describe *`).
- Show the Kubernete pods detailed information about the running pods for the deployment (e.g., `kubectl get pods`).
- The live chat service can be accessed externally and working correctly (using browsers and starting three chat windows.)