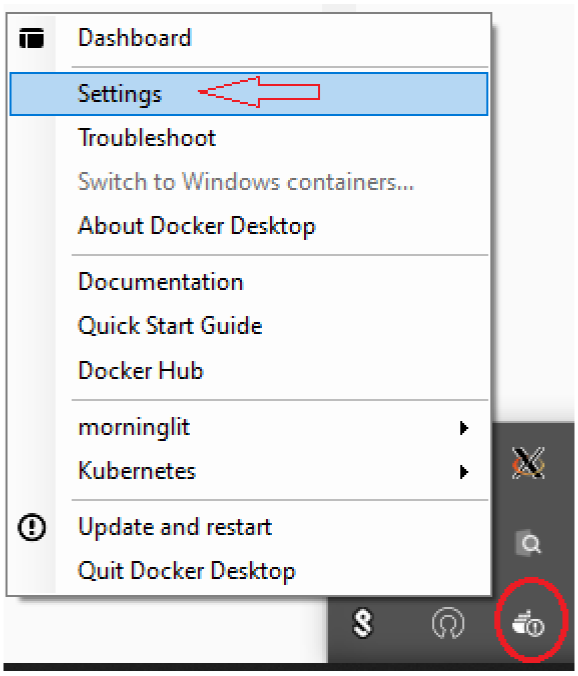
1a. Ambrose Liew Cheng Yuan, A0204750N

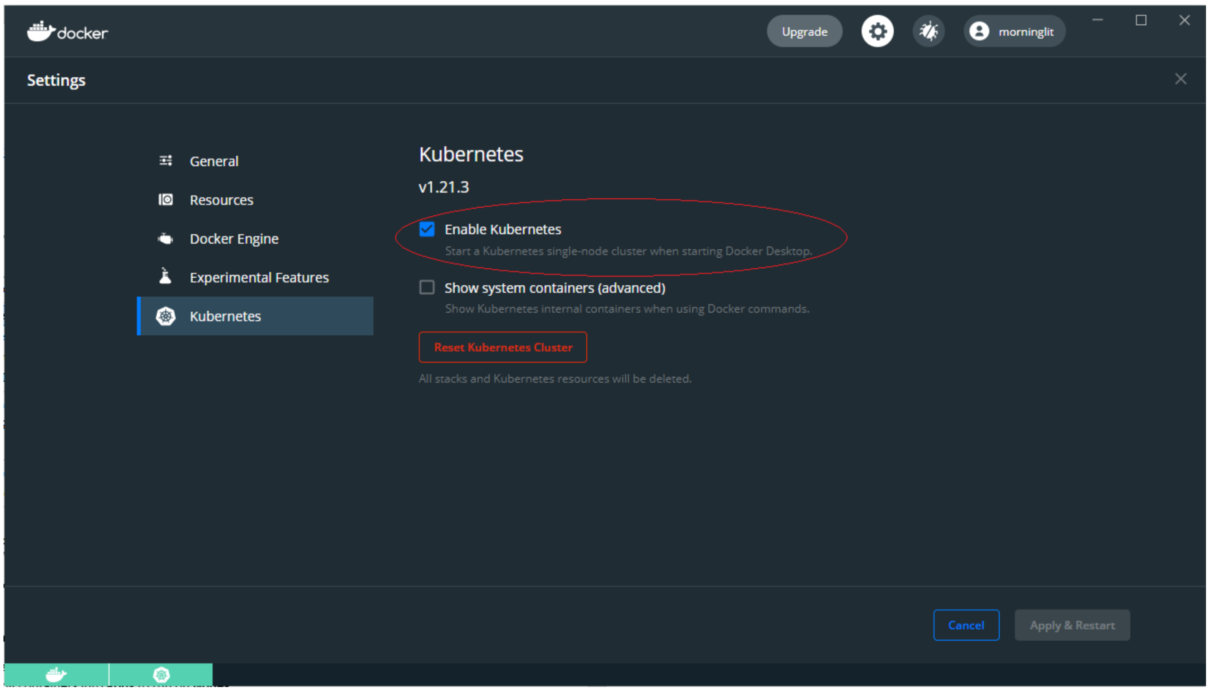
1b. https://github.com/MorningLit/CS3219

1c.

1) To setup Kubernetes through Docker Desktop, right click on the docker icon and click on "Settings" from the list.



Next, enable the checkbox that says "Enable Kubernetes".



Kubernetes

v1.21.3

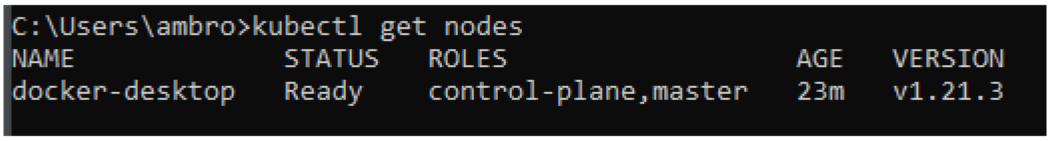
[✓] Enable Kubernetes
Start a Kubernetes single-node cluster when starting Docker Desktop.

[ ] Show system containers (advanced)
Show Kubernetes internal containers when using Docker commands.

Reset Kubernetes Cluster

All stacks and Kubernetes resources will be deleted.

Type “kubectl get nodes” to see your Kubernetes cluster



```
C:\Users\ambro>kubectl get nodes
NAME             STATUS   ROLES                  AGE   VERSION
docker-desktop   Ready    control-plane,master   23m   v1.21.3
```

2) In the folder with my .yaml files, run “kubectl apply -f deployment.yaml”, this deploys the deployment component on our Kubernetes cluster.
   Next run “kubectl apply -f service.yaml” which deploys the service component to the same cluster.
   However, when we run “kubectl exec <<POD’s NAME>> -- printenv | grep SERVICE”, we see no mention of our service that we deployed. This is because we created the replicas before the Service. To do this the right way we can kill our 2 pods and wait for deployment to recreate them. Run “kubectl scale deployment my-nginx --replicas=0” and then followed with “kubectl scale deployment my-nginx --replicas=2” to recreate the action mentioned above. And then running “kubectl exec <<POD’s NAME>> -- printenv | grep SERVICE”, we can see from the output, the environment variables “MY_NGINX_SERVICE_HOST” and “MY_NGINX_SERVICE_PORT”, that our service has connected to the pod.

1d. In Kubernetes it has some of its own components. Such as:

- **Nodes**. A node can a virtual or a physical machine and Kubernetes runs our workload by placing containers into **Pods** to run on **Nodes.**
- **Pods.** Pods are the smallest deployable units of computing in Kubernetes. It is an abstraction over a container. A pod can have one or more containers. Pods are usually meant to run 1 application per Pod. (If a pod runs multiple containers, its usually when there is 1 main application container and some helper containers to provide some side service for the main application container) Each pod also has its own internal IP address. However, in Kubernetes, pods are ephemeral, which means that when pods are restarted, a new internal IP address will be assigned to it. This results in some inconvenience because we must manually configure the IP address every time a Pod crashes or restarts.
- **Service.** Service is a permanent IP address that pods can “attach” to. This solves the issue mentioned above as Service’s lifecycle is not connected to the pod’s lifecycle. Thus, when a Pod restarts, it will still have the same IP address as before. A service can be an external IP address or an internal IP address. However, Service’s IP address is usually in IPv4 format, which is not easy for us humans to memorise.
- **Ingress.** Ingress solves the above-mentioned issue by supporting Name-based virtual host and URI-based routing support. (eg. https://www.example.com)
- **ConfigMap.** Is an external configuration of our application. This allows Pods to connect to the ConfigMap and use environment variables defined in the ConfigMap without us having to go the long way bout of changing a variable, then manually rebuilding and pushing our build to the repository to update our Pods.
- **Secret.** Secret is similar to ConfigMap, however, it is used to store sensitive or secret environment variables by encrypting it. Such as a password or an API key.

- **Volumes.** On-disk files in a container are ephemeral which are a problem when trying to store persistent data. With Kubernetes’s Volume, it helps to solve this problem by allowing data to be stored to a local storage or a remote storage.
- **Deployments.** Are a blueprint for constructing pods. They are an abstraction of pods. Can define the number of replicas of the pods that you have defined, allowing scaling up or scaling down of pods to be super simple. Deployments are used for stateless apps. While another component, **StatefulSet** are used for stateful applications or Databases.