Roberto Morassi del Blanco

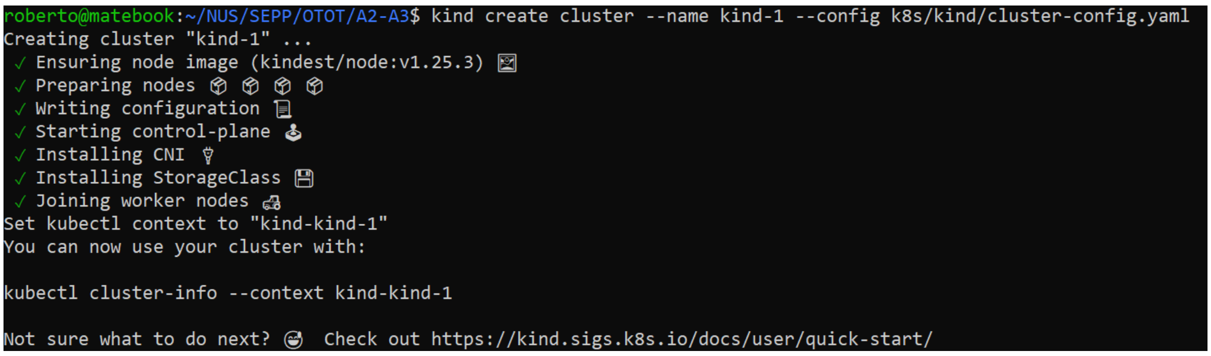
A0265239X

https://github.com/rmorassi/CS3219-OTOT-A2-A3

Task A2.1

Commands to create the cluster

1. “kind create cluster --name kind-1 --config k8s/kind/cluster-config.yaml” creates the cluster from the configuration file and launches it.



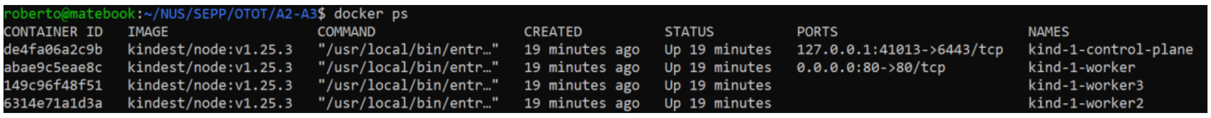
```
roberto@matebook:~/NUS/SEPP/OTOT/A2-A3$ kind create cluster --name kind-1 --config k8s/kind/cluster-config.yaml
Creating cluster "kind-1" ...
 ✓ Ensuring node image (kindest/node:v1.25.3) 🖼
 ✓ Preparing nodes 📦 📦 📦 📦
 ✓ Writing configuration 📜
 ✓ Starting control-plane 🕹️
 ✓ Installing CNI 🔌
 ✓ Installing StorageClass 💾
 ✓ Joining worker nodes 🚜
Set kubectl context to "kind-kind-1"
You can now use your cluster with:

kubectl cluster-info --context kind-kind-1

Not sure what to do next? 😅  Check out https://kind.sigs.k8s.io/docs/user/quick-start/
```

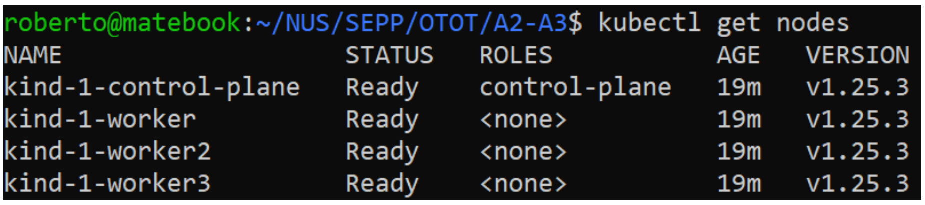
Commands to verify the cluster

1. The nodes can be seen running with the command “docker ps”.



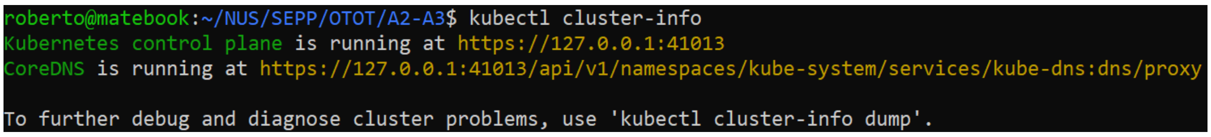
```
roberto@matebook:~/NUS/SEPP/OTOT/A2-A3$ docker ps
CONTAINER ID   IMAGE                  COMMAND                  CREATED          STATUS          PORTS                       NAMES
de4fa06a2c9b   kindest/node:v1.25.3   "/usr/local/bin/entr…"   19 minutes ago   Up 19 minutes   127.0.0.1:41013->6443/tcp   kind-1-control-plane
abae9c5eae8c   kindest/node:v1.25.3   "/usr/local/bin/entr…"   19 minutes ago   Up 19 minutes   0.0.0.0:80->80/tcp          kind-1-worker
149c96f48f51   kindest/node:v1.25.3   "/usr/local/bin/entr…"   19 minutes ago   Up 19 minutes                               kind-1-worker3
6314e71a1d3a   kindest/node:v1.25.3   "/usr/local/bin/entr…"   19 minutes ago   Up 19 minutes                               kind-1-worker2
```

2. “kubectl get nodes” can also be used for the same purpose.



```
roberto@matebook:~/NUS/SEPP/OTOT/A2-A3$ kubectl get nodes
NAME                   STATUS   ROLES           AGE   VERSION
kind-1-control-plane   Ready    control-plane   19m   v1.25.3
kind-1-worker          Ready    <none>          19m   v1.25.3
kind-1-worker2         Ready    <none>          19m   v1.25.3
kind-1-worker3         Ready    <none>          19m   v1.25.3
```

3. “kubectl cluster-info” can let us know the IP Address and ports of clusters that are currently running. We see that the open ports match with what is displayed in “docker ps”.



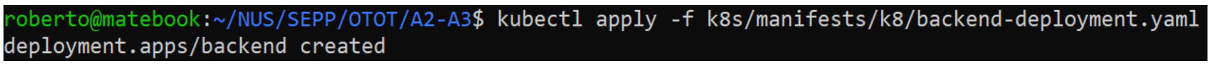
```
roberto@matebook:~/NUS/SEPP/OTOT/A2-A3$ kubectl cluster-info
Kubernetes control plane is running at https://127.0.0.1:41013
CoreDNS is running at https://127.0.0.1:41013/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
```

Task A2.2

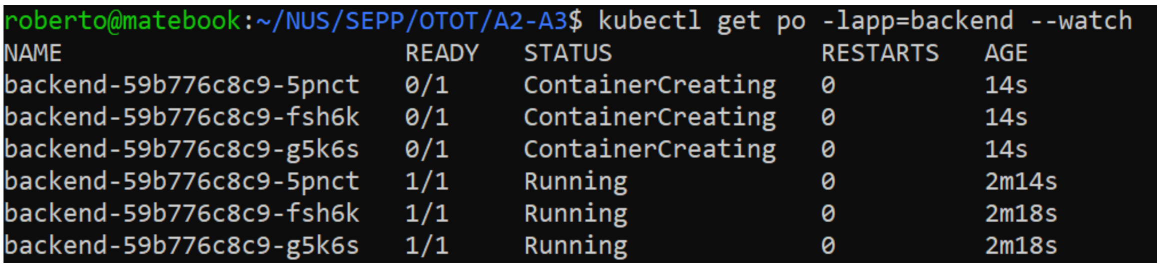
Commands to create and verify Deployment in Cluster

1. We run “kubectl apply -f k8s/manifests/k8/backend-deployment.yaml” to apply the deployment into the cluster. Note that the file was modified to point to the 8080 port which aligns with the port of “app” created in A1.



```
roberto@matebook:~/NUS/SEPP/OTOT/A2-A3$ kubectl apply -f k8s/manifests/k8/backend-deployment.yaml
deployment.apps/backend created
```

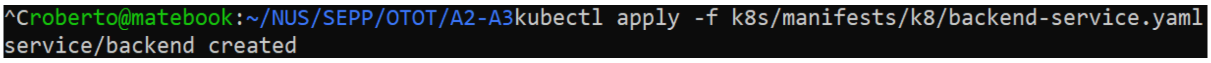
2. We run “kubectl get po -lapp=backend –watch” to check that the deployment is working properly. Since the status went from “ContainerCreating” to “Running”, we can confirm that the “app” image was correctly pulled from the online repository successfully (uploaded manually for this assignment).



```
roberto@matebook:~/NUS/SEPP/OTOT/A2-A3$ kubectl get po -lapp=backend --watch
NAME                       READY   STATUS              RESTARTS   AGE
backend-59b776c8c9-5pnct   0/1     ContainerCreating   0          14s
backend-59b776c8c9-fsh6k   0/1     ContainerCreating   0          14s
backend-59b776c8c9-g5k6s   0/1     ContainerCreating   0          14s
backend-59b776c8c9-5pnct   1/1     Running             0          2m14s
backend-59b776c8c9-fsh6k   1/1     Running             0          2m18s
backend-59b776c8c9-g5k6s   1/1     Running             0          2m18s
```

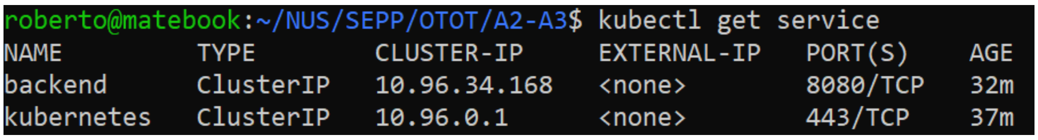
Commands to create and verify Service in Cluster

1. Run “kubectl apply -f k8s/manifests/k8/backend-service.yaml” to create the service in our cluster.



```
^Croberto@matebook:~/NUS/SEPP/OTOT/A2-A3kubectl apply -f k8s/manifests/k8/backend-service.yaml
service/backend created
```

2. Run “kubectl get service” to confirm the service is running correctly.

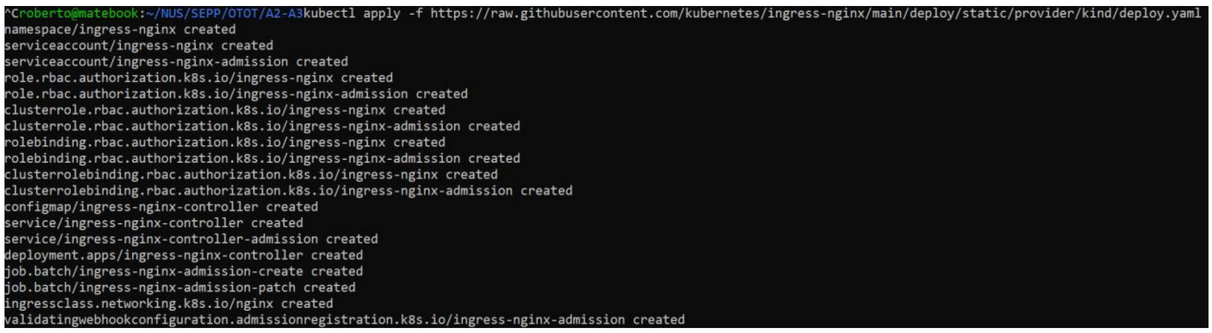


```
roberto@matebook:~/NUS/SEPP/OTOT/A2-A3$ kubectl get service
NAME         TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE
backend      ClusterIP   10.96.34.168   <none>        8080/TCP   32m
kubernetes   ClusterIP   10.96.0.1      <none>        443/TCP    37m
```

Task A2.3

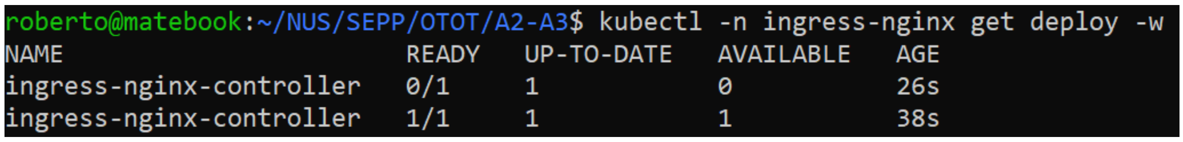
Commands create ingress-controller and verify it works

1. We run “kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/main/deploy/static/provider/kind/deploy.yaml” to pull the ingress controller file from the internet and apply it to our cluster.



```
^Croberto@matebook:~/NUS/SEPP/OTOT/A2-A3kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/main/deploy/static/provider/kind/deploy.yaml
namespace/ingress-nginx created
serviceaccount/ingress-nginx created
serviceaccount/ingress-nginx-admission created
role.rbac.authorization.k8s.io/ingress-nginx created
role.rbac.authorization.k8s.io/ingress-nginx-admission created
clusterrole.rbac.authorization.k8s.io/ingress-nginx created
clusterrole.rbac.authorization.k8s.io/ingress-nginx-admission created
rolebinding.rbac.authorization.k8s.io/ingress-nginx created
rolebinding.rbac.authorization.k8s.io/ingress-nginx-admission created
clusterrolebinding.rbac.authorization.k8s.io/ingress-nginx created
clusterrolebinding.rbac.authorization.k8s.io/ingress-nginx-admission created
configmap/ingress-nginx-controller created
service/ingress-nginx-controller created
service/ingress-nginx-controller-admission created
deployment.apps/ingress-nginx-controller created
job.batch/ingress-nginx-admission-create created
job.batch/ingress-nginx-admission-patch created
ingressclass.networking.k8s.io/nginx created
validatingwebhookconfiguration.admissionregistration.k8s.io/ingress-nginx-admission created
```

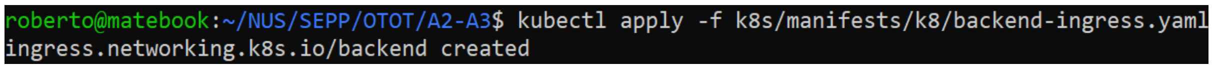
2. We run “kubectl -n ingress-nginx get deploy -w” to check when the ingress controller starts working and check if it is functioning correctly.



```
roberto@matebook:~/NUS/SEPP/OTOT/A2-A3$ kubectl -n ingress-nginx get deploy -w
NAME                       READY   UP-TO-DATE   AVAILABLE   AGE
ingress-nginx-controller   0/1     1            0           26s
ingress-nginx-controller   1/1     1            1           38s
```

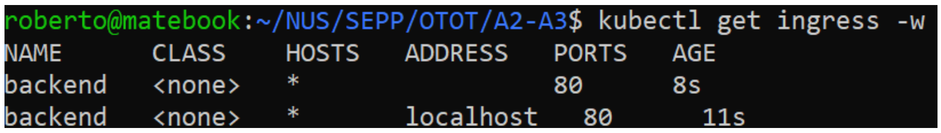
Commands create Ingress Object and verify it works

1. Run “kubectl apply -f k8s/manifests/k8/backend-ingress.yaml” to configure Ingress Object.



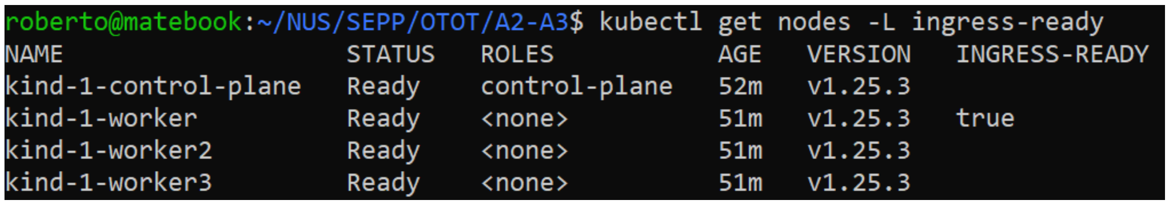
```
roberto@matebook:~/NUS/SEPP/OTOT/A2-A3$ kubectl apply -f k8s/manifests/k8/backend-ingress.yaml
ingress.networking.k8s.io/backend created
```

2. Run “kubectl get ingress -w” to make sure the Ingress Object is running and configured correctly.



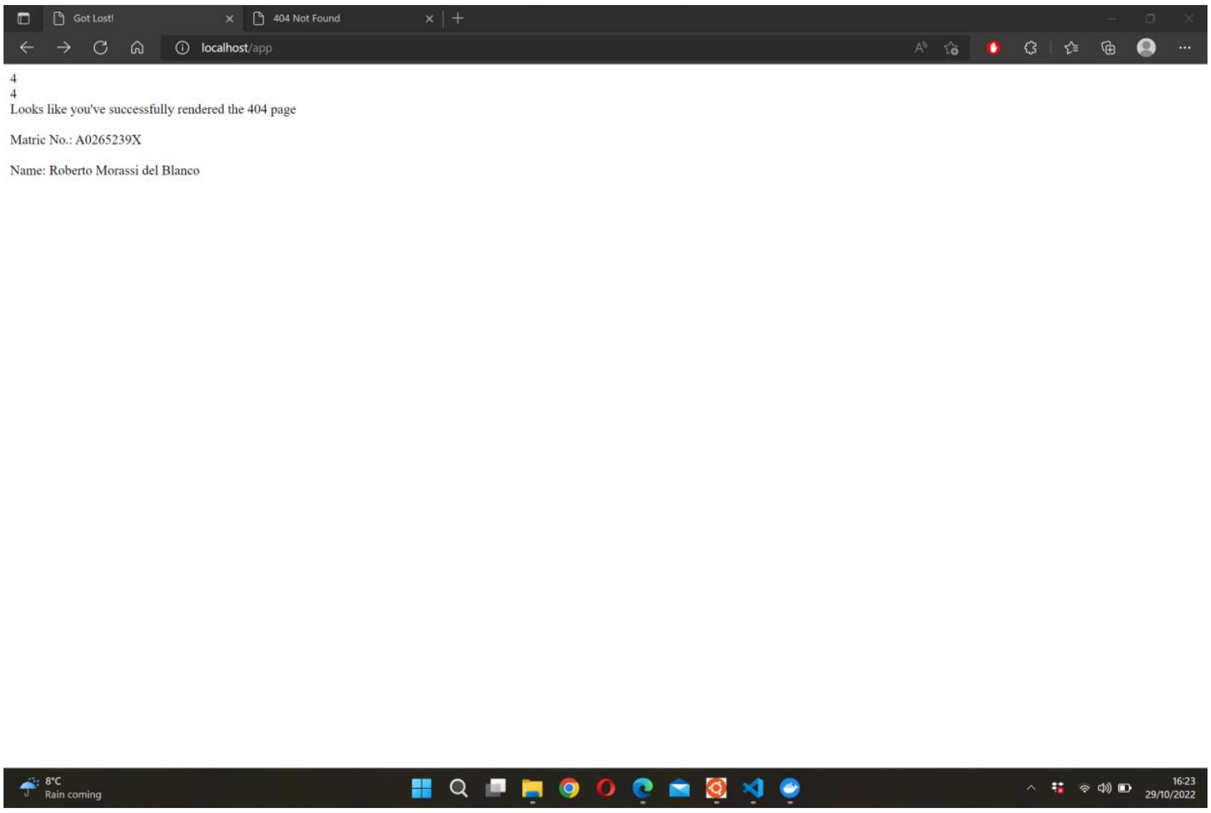
```
roberto@matebook:~/NUS/SEPP/OTOT/A2-A3$ kubectl get ingress -w
NAME      CLASS    HOSTS   ADDRESS     PORTS   AGE
backend   <none>   *                   80      8s
backend   <none>   *       localhost   80      11s
```

3. Run “kubectl get nodes -L ingress-ready” to check if the ingress is configured correctly.



```
roberto@matebook:~/NUS/SEPP/OTOT/A2-A3$ kubectl get nodes -L ingress-ready
NAME                   STATUS   ROLES           AGE   VERSION   INGRESS-READY
kind-1-control-plane   Ready    control-plane   52m   v1.25.3
kind-1-worker          Ready    <none>          51m   v1.25.3   true
kind-1-worker2         Ready    <none>          51m   v1.25.3
kind-1-worker3         Ready    <none>          51m   v1.25.3
```

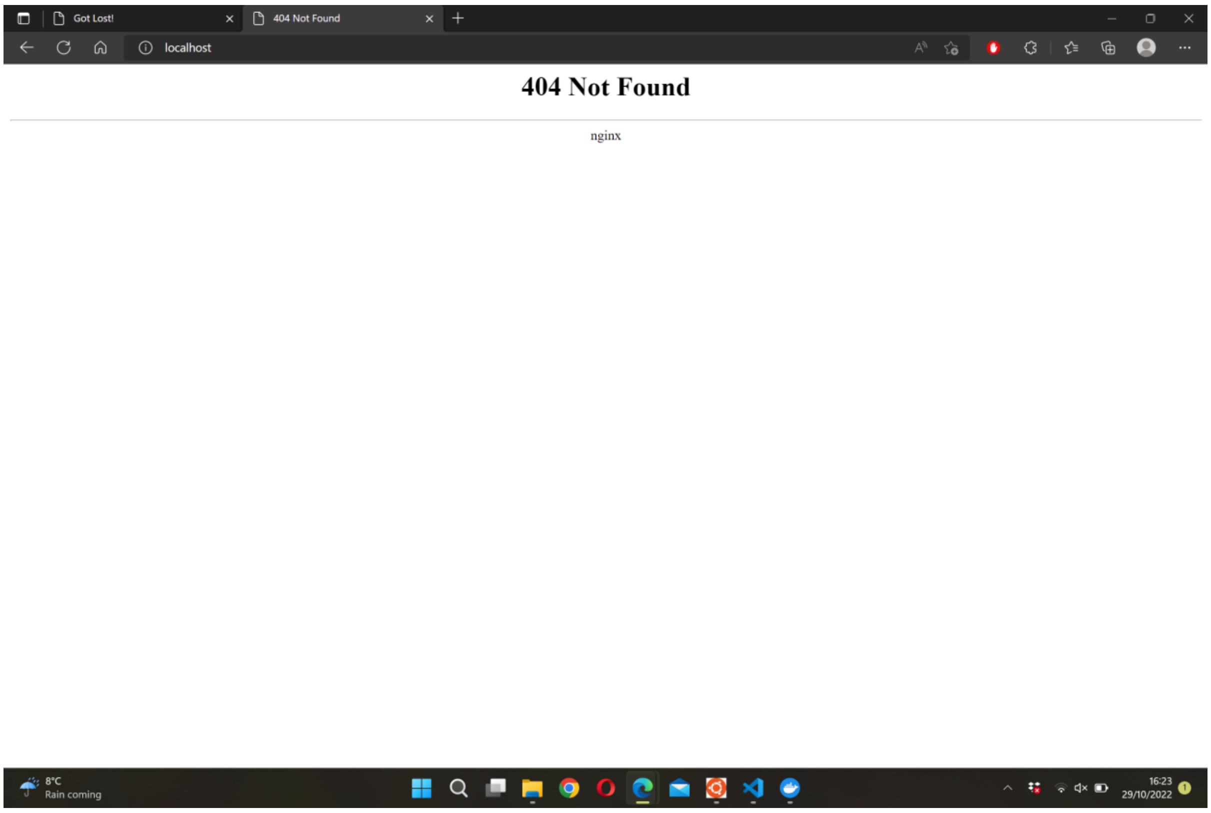
Screenshot proof of the websites working as usual



```
4
4
Looks like you've successfully rendered the 404 page

Matric No.: A0265239X

Name: Roberto Morassi del Blanco
```



```
404 Not Found
nginx
```