Introduction to Kubernetes

Objectives

In this lab, you will:

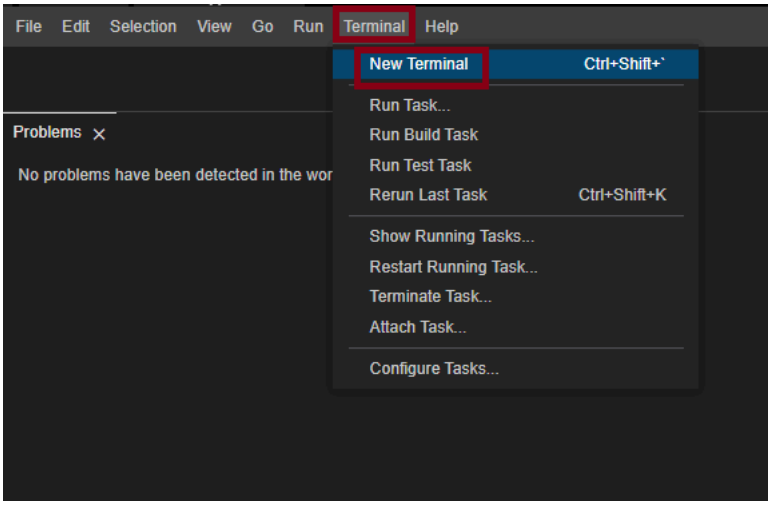
- Use the `kubectl` CLI
- Create a Kubernetes Pod
- Create a Kubernetes Deployment
- Create a ReplicaSet that maintains a set number of replicas
- Witness Kubernetes load balancing in action

Note: Kindly complete the lab in a single session without any break because the lab may go on offline mode and may cause errors. If you face any issues/errors during the lab process, please logout from the lab environment. Then clear your system cache and cookies and try to complete the lab.

Verify the environment and command line tools

1. If a terminal is not already open, open a terminal window by using the menu in the editor: `Terminal > New Terminal`.

   Note: Please skip this step if the terminal already appears.



2. Verify that `kubectl` CLI is installed.

```
kubectl version
```

You should see the following output, although the versions may be different:

```
theia@theiadocker /home/project$ kubectl version
Client Version: version.Info{Major:"1", Minor:"22", GitVersion:"v1.22.3", GitCommit:"c92036820499fedefec0f847e2054d824aea6cd1", GitTreeSt
"2021-10-27T18:41:28Z", GoVersion:"go1.16.9", Compiler:"gc", Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"21", GitVersion:"v1.21.11+IKS", GitCommit:"7d30e1c191e870ff995f9b6ba21452d0325db2ad", GitT
Date:"2022-03-17T16:12:51Z", GoVersion:"go1.16.15", Compiler:"gc", Platform:"linux/amd64"}
theia@theiadocker /home/project$
```

3. Change to your project folder.

   Note: Please skip this step if you are already on the '/home/project' directory

```
cd /home/project
```

4. Clone the git repository that contains the artifacts needed for this lab, if it doesn't already exist.

```
[ ! -d 'CC201' ] && git clone https://github.com/ibm-developer-skills-network/CC201.git
```

```
theia@theiadocker /home/project$ [ ! -d 'CC201' ] && git clone https://github.com/ibm-developer-skills-network/CC201.git
Cloning into 'CC201'...
remote: Enumerating objects: 20, done.
remote: Counting objects: 100% (20/20), done.
remote: Compressing objects: 100% (13/13), done.
remote: Total 20 (delta 6), reused 19 (delta 6), pack-reused 0
Unpacking objects: 100% (20/20), done.
theia@theiadocker /home/project$
```

5. Change to the directory for this lab by running the following command. `cd` will change the working/current directory to the directory with the name specified, in this case **CC201/labs/2_IntroKubernetes**.

```
cd CC201/labs/2_IntroKubernetes/
```

6. List the contents of this directory to see the artifacts for this lab.

```
ls
```

```
theia@theiadocker /home/project/CC201/labs/2_IntroKubernetes$ ls
app.js  Dockerfile  hello-world-apply.yaml  hello-world-create.yaml  package.json
theia@theiadocker /home/project/CC201/labs/2_IntroKubernetes$
```

Use the `kubectl` CLI

Recall that Kubernetes namespaces enable you to virtualize a cluster. You already have access to one namespace in a Kubernetes cluster, and `kubectl` is already set to target that cluster and namespace.

Let's look at some basic `kubectl` commands.

1. `kubectl` requires configuration so that it targets the appropriate cluster. Get cluster information with the following command:

```
kubectl config get-clusters
```

```
theia@theiadocker /home/project/CC201/labs/2_IntroKubernetes$ kubectl config get-clusters
NAME
labs-prod-kubernetes-sandbox/c8ana0sw0ljj8gkugn50
theia@theiadocker /home/project/CC201/labs/2_IntroKubernetes$
```

2. A `kubectl` context is a group of access parameters, including a cluster, a user, and a namespace. View your current context with the following command:

```
kubectl config get-contexts
```

```
theia@theiadocker-[redacted] /home/project/CC201/labs/2_IntroKubernetes$ kubectl config get-contexts
CURRENT   NAME                  CLUSTER                                          AUTHINFO        NAMESPACE
*         [redacted]-context    labs-prod-kubernetes-sandbox/c8ana0sw0ljj8gkugn50 [redacted]     sn-labs-[redacted]
theia@theiadocker-[redacted] /home/project/CC201/labs/2_IntroKubernetes$
```

3. List all the Pods in your namespace. If this is a new session for you, you will not see any Pods.

```
kubectl get pods
```

```
theia@theiadocker-[redacted] /home/project/CC201/labs/2_IntroKubernetes$ kubectl get pods
No resources found in sn-labs-[redacted] namespace.
theia@theiadocker-[redacted] /home/project/CC201/labs/2_IntroKubernetes$
```

Create a Pod with an imperative command

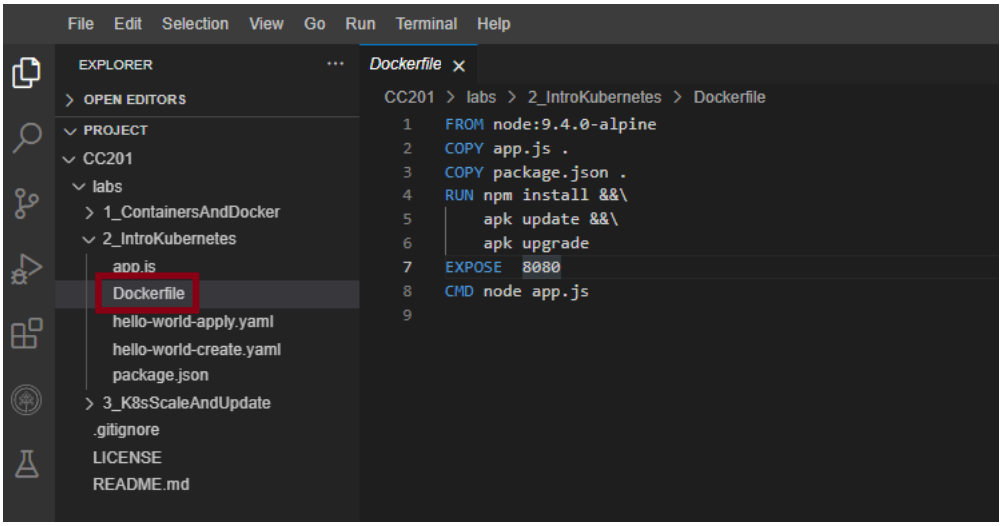
Now it's time to create your first Pod. This Pod will run the `hello-world` image you built and pushed to IBM Cloud Container Registry in the last lab. As explained in the videos for this module, you can create a Pod imperatively or declaratively. Let's do it imperatively first.

1. Export your namespace as an environment variable so that it can be used in subsequent commands.

```
export MY_NAMESPACE=sn-labs-$USERNAME
```

```
theia@theiadocker-[redacted] /home/project/CC201/labs/2_IntroKubernetes$ export MY_NAMESPACE=sn-labs-$USERNAME
theia@theiadocker-[redacted] /home/project/CC201/labs/2_IntroKubernetes$
```

2. Click the Explorer icon (it looks like a sheet of paper) on the left side of the window, and then navigate to the directory for this lab: `CC201 > labs > 2_IntroKubernetes`. Click on `Dockerfile`. This is the file that will be used to build our image.



3. Build and push the image again, as it may have been deleted automatically since you completed the first lab.

```
docker build -t us.icr.io/$MY_NAMESPACE/hello-world:1 . && docker push us.icr.io/$MY_NAMESPACE/hello-world:1
```

```
theia@theiadocker-[redacted]/home/project/CC201/labs/2_IntroKubernetes$ docker build -t us.icr.io/$MY_NAMESPACE/hello-world:1 . && doc
ACE/hello-world:1
Sending build context to Docker daemon  6.656kB
Step 1/6 : FROM node:9.4.0-alpine
9.4.0-alpine: Pulling from library/node
605ce1bd3f31: Pull complete
fe58b30348fe: Pull complete
46ef8987ccbd: Pull complete
Digest: sha256:9cd67a00ed111285460a83847720132204185e9321ec35dacec0d8b9bf674adf
Status: Downloaded newer image for node:9.4.0-alpine
 ---> b5f94997f35f
Step 2/6 : COPY app.js .
 ---> 28350e465969
Step 3/6 : COPY package.json .
 ---> 45bf6db4af5f
Step 4/6 : RUN npm install &&    apk update &&    apk upgrade
 ---> Running in a37db9ced1bc
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN hello-world-demo@0.0.1 No repository field.
npm WARN hello-world-demo@0.0.1 No license field.

added 50 packages in 2.085s
fetch http://dl-cdn.alpinelinux.org/alpine/v3.6/main/x86_64/APKINDEX.tar.gz
fetch http://dl-cdn.alpinelinux.org/alpine/v3.6/community/x86_64/APKINDEX.tar.gz
v3.6.5-44-gda55e27396 [http://dl-cdn.alpinelinux.org/alpine/v3.6/main]
v3.6.5-34-gf0ba0b43d5 [http://dl-cdn.alpinelinux.org/alpine/v3.6/community]
OK: 8448 distinct packages available
Upgrading critical system libraries and apk-tools:
(1/1) Upgrading apk-tools (2.7.5-r0 -> 2.7.6-r0)
Executing busybox-1.26.2-r9.trigger
Continuing the upgrade transaction with new apk-tools:
(1/7) Upgrading musl (1.1.16-r14 -> 1.1.16-r15)
(2/7) Upgrading busybox (1.26.2-r9 -> 1.26.2-r11)
Executing busybox-1.26.2-r11.post-upgrade
(3/7) Upgrading libressl2.5-libcrypto (2.5.5-r0 -> 2.5.5-r2)
(4/7) Upgrading libressl2.5-libssl (2.5.5-r0 -> 2.5.5-r2)
(5/7) Installing libressl2.5-libtls (2.5.5-r2)
```

4. Run the `hello-world` image as a container in Kubernetes.

```
kubectl run hello-world --image us.icr.io/$MY_NAMESPACE/hello-world:1 --overrides='{"spec":{"template":{"spec":{"imagePullSecrets":[{"name":"icr"}]
```

The `--overrides` option here enables us to specify the needed credentials to pull this image from IBM Cloud Container Registry. Note that this is an imperative command, as we told Kubernetes explicitly what to do: run `hello-world`.

```
theia@theiadocker-[redacted]/home/project/CC201/labs/2_IntroKubernetes$ kubectl run hello-world --image us.icr.io/$MY_NAMESPACE/hello-
{"template":{"spec":{"imagePullSecrets":[{"name":"icr"}]}}}}'
pod/hello-world created
theia@theiadocker-[redacted]/home/project/CC201/labs/2_IntroKubernetes$
```

5. List the Pods in your namespace.

```
kubectl get pods
```

```
theia@theiadocker-[redacted]/home/project/CC201/labs/2_IntroKubernetes$ kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
hello-world   1/1     Running   0          34s
theia@theiadocker-[redacted]/home/project/CC201/labs/2_IntroKubernetes$
```

Great, the previous command indeed created a Pod for us. You can see an auto-generated name was given to this Pod.

You can also specify the wide option for the output to get more details about the resource.

```
kubectl get pods -o wide
```

```
theia@theiadocker-[redacted]/home/project/CC201/labs/2_IntroKubernetes$ kubectl get pods -o wide
NAME          READY   STATUS    RESTARTS   AGE   IP               NODE           NOMINATED NODE   READINESS GATES
hello-world   1/1     Running   0          59s   172.17.183.177   10.241.64.24   <none>           <none>
theia@theiadocker-[redacted]/home/project/CC201/labs/2_IntroKubernetes$
```

6. Describe the Pod to get more details about it.

```
kubectl describe pod hello-world
```

```
theia@theiadocker-[redacted]/home/project/CC201/labs/2_IntroKubernetes$ kubectl describe pod hello-world
Name:                 hello-world
Namespace:            sn-labs-[redacted]
Priority:             1
Priority Class Name:  normal
Node:                 10.241.64.24/10.241.64.24
Start Time:           Fri, 08 Apr 2022 05:15:40 +0000
Labels:               run=hello-world
Annotations:          cni.projectcalico.org/containerID: c89fd419d56a582514d497f0b01b939cf745343036e9a45f135235e7d5bc528e
                      cni.projectcalico.org/podIP: 172.17.183.177/32
                      cni.projectcalico.org/podIPs: 172.17.183.177/32
                      kubernetes.io/limit-ranger:
                        LimitRanger plugin set: cpu, ephemeral-storage, memory request for container hello-world; cpu, ephemeral-storage,
                      kubernetes.io/psp: ibm-privileged-psp
Status:               Running
IP:                   172.17.183.177
IPs:
  IP:  172.17.183.177
Containers:
  hello-world:
    Container ID:   containerd://31c934f489c232a36729b3e3f013a5619f11fc8f95ee8a1007f9f540dc4d420a
    Image:          us.icr.io/sn-labs-[redacted]/hello-world:1
    Image ID:       us.icr.io/sn-labs-[redacted]/hello-world@sha256:a04a56181ae9136e4b7033d5284ce9d68fe812c21b28592ffb292d8b496b6b81
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Fri, 08 Apr 2022 05:15:46 +0000
    Ready:          True
    Restart Count:  0
    Limits:
      cpu:                500m
      ephemeral-storage:  5Gi
      memory:             512Mi
    Requests:
      cpu:                200m
      ephemeral-storage:  512Mi
      memory:             128Mi
    Environment:          <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-bjdzp (ro)
Conditions:
  Type              Status
```

Note: The output shows the pod parameters like **Namespace, Pod Name, IP address, the time when the pod started running** and also the container parameters like **container ID, image name & ID, running status and the memory/CPU limits.**

7. Delete the Pod.

```
kubectl delete pod hello-world
```

```
theia@theiadocker-[redacted]/home/project/CC201/labs/2_IntroKubernetes$ kubectl delete pod hello-world
pod "hello-world" deleted
theia@theiadocker-[redacted]/home/project/CC201/labs/2_IntroKubernetes$
```

This command takes a while to execute the deletion of the pod. Please wait till the terminal prompt appears again.

8. List the Pods to verify that none exist.

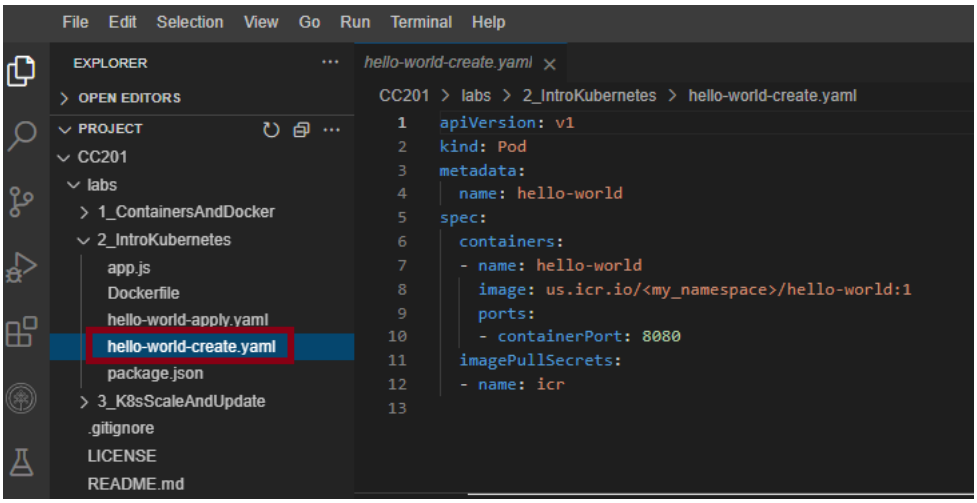
```
kubectl get pods
```

```
theia@theiadocker-                /home/project/CC201/labs/2_IntroKubernetes$ kubectl get pods
No resources found in sn-labs-                namespace.
theia@theiadocker-                /home/project/CC201/labs/2_IntroKubernetes$
```

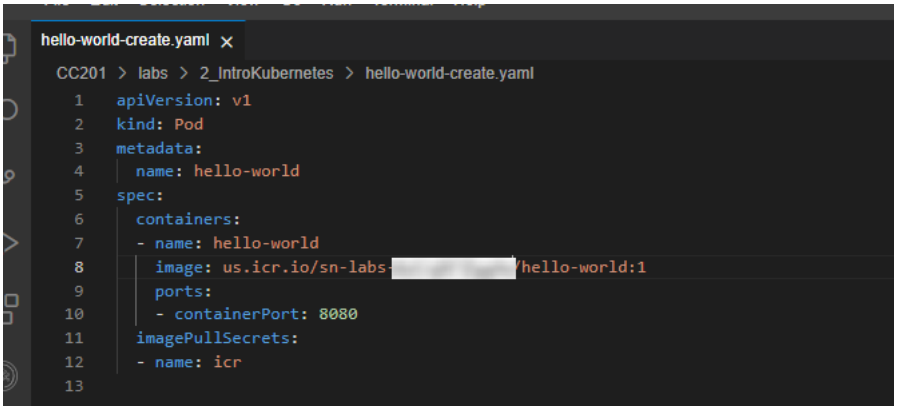
Create a Pod with imperative object configuration

Imperative object configuration lets you create objects by specifying the action to take (e.g., create, update, delete) while using a configuration file. A configuration file, `hello-world-create.yaml`, is provided to you in this directory.

1. Use the Explorer to view and edit the configuration file. Click the Explorer icon (it looks like a sheet of paper) on the left side of the window, and then navigate to the directory for this lab: `CC201 > labs > 2_IntroKubernetes`. Click `hello-world-create.yaml` to view the configuration file.



2. Use the Explorer to edit `hello-world-create.yaml`. You need to insert your namespace where it says `<my_namespace>`. Make sure to save the file when you're done.



3. Imperatively create a Pod using the provided configuration file.

```
kubectl create -f hello-world-create.yaml
```

Note that this is indeed imperative, as you explicitly told Kubernetes to *create* the resources defined in the file.

```
theia@theiadocker-                /home/project/CC201/labs/2_IntroKubernetes$ kubectl create -f hello-world-create.yaml
pod/hello-world created
theia@theiadocker-                /home/project/CC201/labs/2_IntroKubernetes$
```

4. List the Pods in your namespace.

```
kubectl get pods
```

```
theia@theiadocker-[redacted] /home/project/CC201/labs/2_IntroKubernetes$ kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
hello-world   1/1     Running   0          17s
theia@theiadocker-[redacted] /home/project/CC201/labs/2_IntroKubernetes$
```

5. Delete the Pod.

```
kubectl delete pod hello-world
```

```
theia@theiadocker-[redacted] /home/project/CC201/labs/2_IntroKubernetes$ kubectl delete pod hello-world
pod "hello-world" deleted
theia@theiadocker[redacted] /home/project/CC201/labs/2_IntroKubernetes$
```

This command takes a while to execute the deletion of the pod. Please wait till the terminal prompt appears again.

6. List the Pods to verify that none exist.

```
kubectl get pods
```

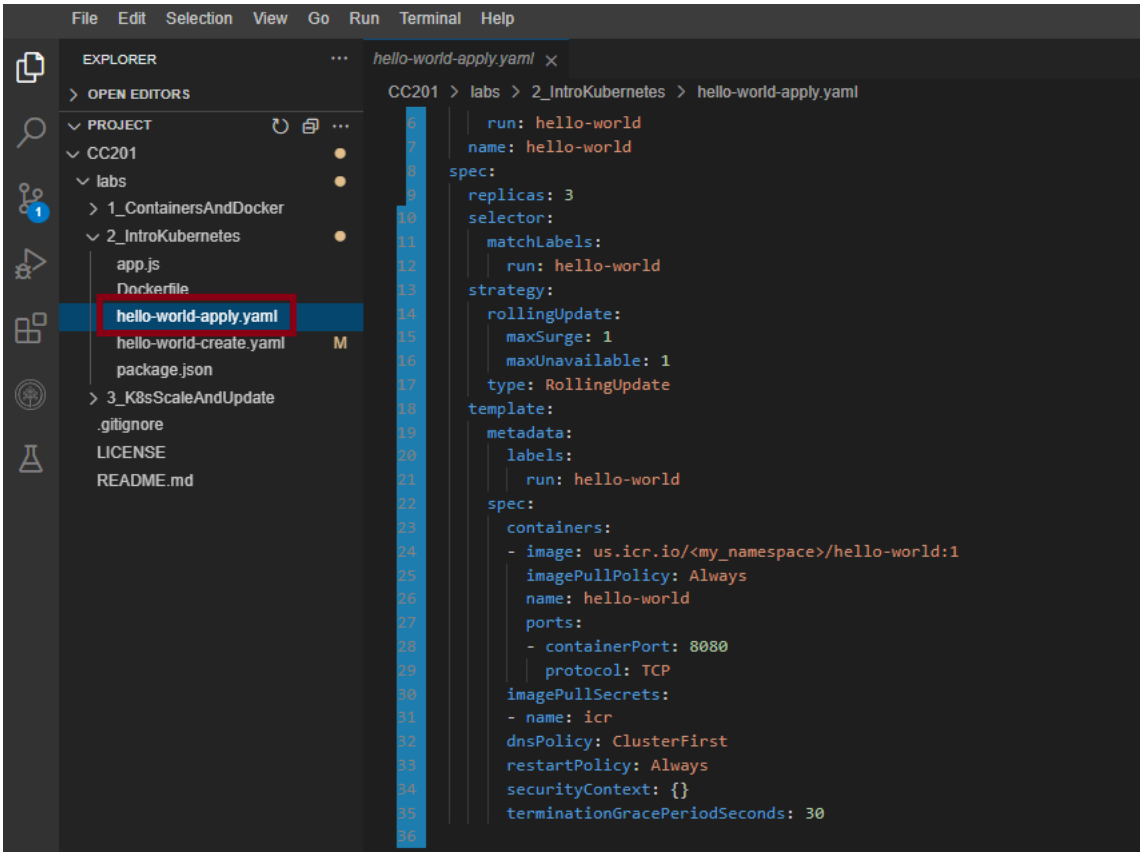
```
theia@theiadocker-[redacted] /home/project/CC201/labs/2_IntroKubernetes$ kubectl get pods
No resources found in sn-labs-[redacted] namespace.
theia@theiadocker-[redacted] /home/project/CC201/labs/2_IntroKubernetes$
```

Create a Pod with a declarative command

The previous two ways to create a Pod were imperative – we explicitly told `kubectl` what to do. While the imperative commands are easy to understand and run, they are not ideal for a production environment. Let's look at declarative commands.

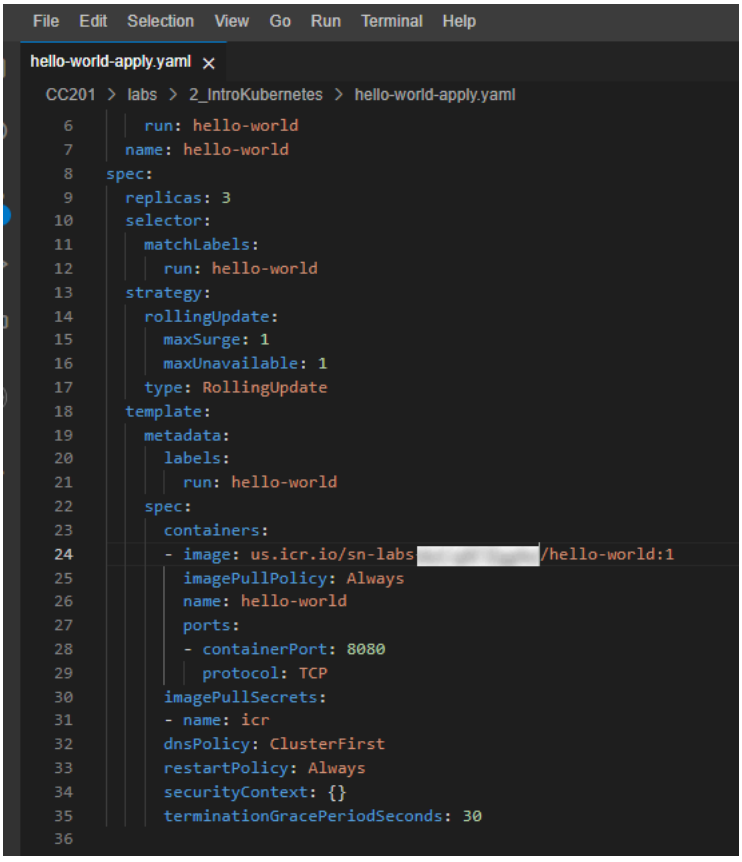
1. A sample `hello-world-apply.yaml` file is provided in this directory. Use the Explorer again to open this file. Notice the following:

- We are creating a Deployment (`kind: Deployment`).
- There will be three replica Pods for this Deployment (`replicas: 3`).
- The Pods should run the `hello-world` image (`- image: us.icr.io/<my_namespace>/hello-world:1`).



You can ignore the rest for now. We will get to a lot of those concepts in the next lab.

2. Use the Explorer to edit `hello-world-apply.yaml`. You need to insert your namespace where it says `<my_namespace>`. Make sure to save the file when you're done.



3. Use the `kubectl apply` command to set this configuration as the desired state in Kubernetes.

```
kubectl apply -f hello-world-apply.yaml
```

```
theia@theiadocker-[redacted]/home/project/CC201/labs/2_IntroKubernetes$ kubectl apply -f hello-world-apply.yaml
deployment.apps/hello-world created
```

4. Get the Deployments to ensure that a Deployment was created.

```
kubectl get deployments
```

```
theia@theiadocker-[redacted] /home/project/CC201/labs/2_IntroKubernetes$ kubectl get deployments
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
hello-world   3/3     3            3           22s
```

5. List the Pods to ensure that three replicas exist.

```
kubectl get pods
```

```
theia@theiadocker-[redacted] /home/project/CC201/labs/2_IntroKubernetes$ kubectl get pods
NAME                           READY   STATUS    RESTARTS   AGE
hello-world-774ddf45b5-86gn6   1/1     Running   0          42s
hello-world-774ddf45b5-9cbv2   1/1     Running   0          41s
hello-world-774ddf45b5-svpf7   1/1     Running   0          41s
theia@theiadocker-[redacted] /home/project/CC201/labs/2_IntroKubernetes$
```

With declarative management, we did not tell Kubernetes which actions to perform. Instead, `kubectl` inferred that this Deployment needed to be created. If you delete a Pod now, a new one will be created in its place to maintain three replicas.

6. Note one of the Pod names from the previous step, replace the `pod_name` in the following command with the pod name that you noted and delete that Pod and list the pods. To see one pod being terminated, there by having just 2 pods, we will follow the **delete**, immediately with **get**.

```
kubectl delete pod <pod_name> && kubectl get pods
```

```
hello-world-5b5467f896-wz45f   1/1     Running   0          10s
theia@theiadocker-ksundararaja:/home/project/CC201/labs/2_IntroKubernetes$ kubectl delete pod hello-world-5b5467f896-9brft && kubectl ge
pod "hello-world-5b5467f896-9brft" deleted
NAME                           READY   STATUS    RESTARTS   AGE
hello-world-5b5467f896-6jpnd   1/1     Running   0          3m7s
hello-world-5b5467f896-wz45f   1/1     Running   0          3m6s
theia@theiadocker-ksundararaja:/home/project/CC201/labs/2_IntroKubernetes$
```

This command takes a while to execute the deletion of the pod. Please wait till the terminal prompt appears again.

7. List the Pods to see a new one being created.

   You may have to run this command a few times as it may take a while to create the new pod.

```
kubectl get pods
```

```
NAME                   READY   STATUS    RESTARTS   AGE
hello-world-774ddf45b5-28k7j   1/1     Running   0          36s
hello-world-774ddf45b5-9cbv2   1/1     Running   0          112s
hello-world-774ddf45b5-svpf7   1/1     Running   0          112s
```

The output should reflect three pods running.

Load balancing the application

Since there are three replicas of this application deployed in the cluster, Kubernetes will load balance requests across these three instances. Let's expose our application to the internet and see how Kubernetes load balances requests.

1. In order to access the application, we have to expose it to the internet using a Kubernetes Service.

   ```
   kubectl expose deployment/hello-world
   ```

```
theia@theiadocker-[redacted]/home/project/CC201/labs/2_IntroKubernetes$ kubectl expose deployment/hello-world
service/hello-world exposed
```

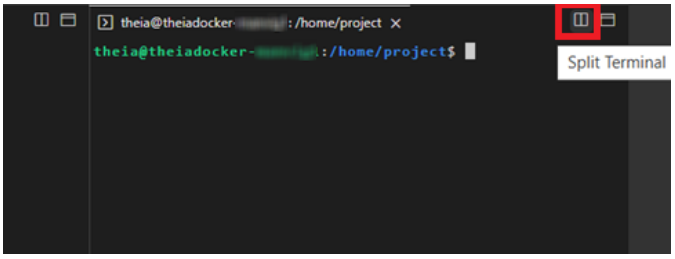
This command creates what is called a ClusterIP Service. This creates an IP address that accessible within the cluster.

2. List Services in order to see that this service was created.

   ```
   kubectl get services
   ```

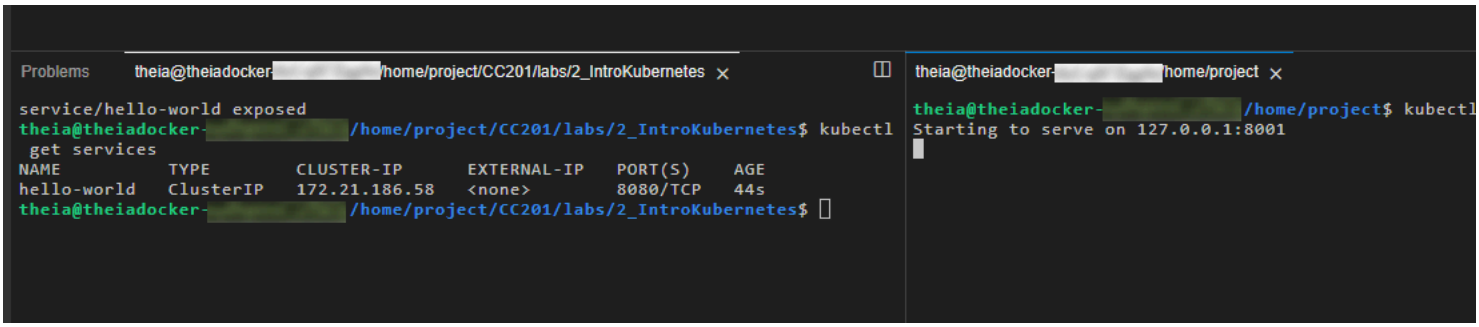
```
theia@theiadocker-[redacted]/home/project/CC201/labs/2_IntroKubernetes$ kubectl get services
NAME          TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
hello-world   ClusterIP   172.21.186.58   <none>        8080/TCP   44s
theia@theiadocker-[redacted]/home/project/CC201/labs/2_IntroKubernetes$
```

3. Open a new split terminal window by locate the split icon in the top-right corner of the terminal panel.



4. Since the cluster IP is not accessible outside of the cluster, we need to create a proxy. Note that this is not how you would make an application externally accessible in a production scenario. Run this command in the new terminal window since your environment variables need to be accessible in the original window for subsequent commands.

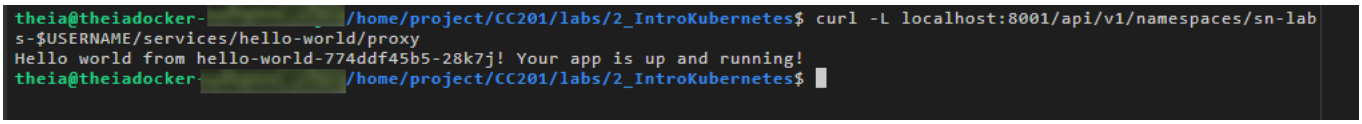
   ```
   kubectl proxy
   ```



This command doesn't terminate until you terminate it. Keep it running so that you can continue to access your app.

5. In the original terminal window, ping the application to get a response.

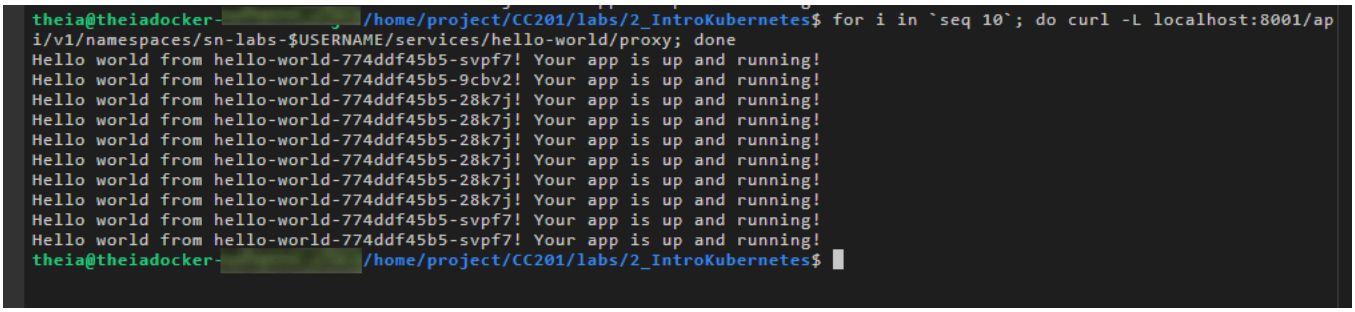
```
curl -L localhost:8001/api/v1/namespaces/sn-labs-$USERNAME/services/hello-world/proxy
```



Notice that this output includes the Pod name.

6. Execute the following command to send ten consecutive requests to the hello-world service via the Kubernetes API proxy. As each request is forwarded, note the pod name in the response (which shows which pod handled the request).

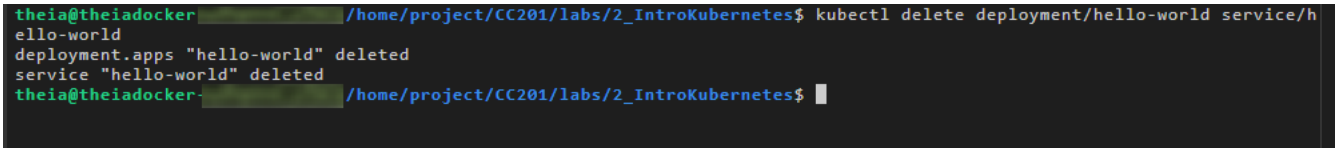
```
for i in `seq 10`; do curl -L localhost:8001/api/v1/namespaces/sn-labs-$USERNAME/services/hello-world/proxy; done
```



You should see more than one Pod name, and quite possibly all three Pod names, in the output. This is because Kubernetes load balances the requests across the three replicas, so each request could hit a different instance of our application.

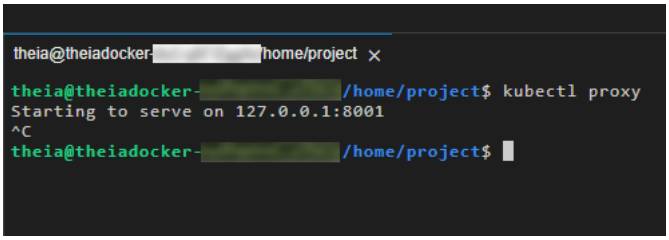
7. Delete the Deployment and Service. This can be done in a single command by using slashes.

```
kubectl delete deployment/hello-world service/hello-world
```



Note: If you face any issues in typing further commands in the terminal, press Enter.

8. Return to the terminal window running the `proxy` command and kill it using `Ctrl+C`.



Congratulations! You have completed the lab for the second module of this course.