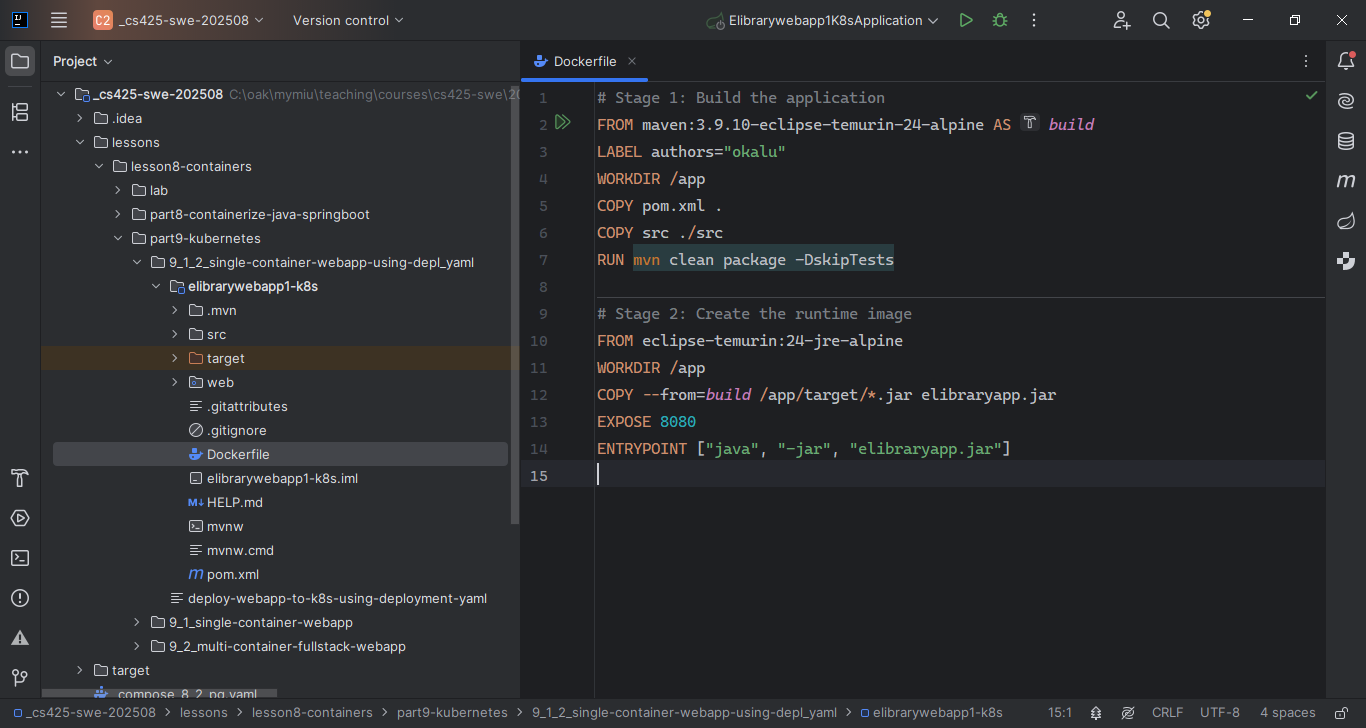
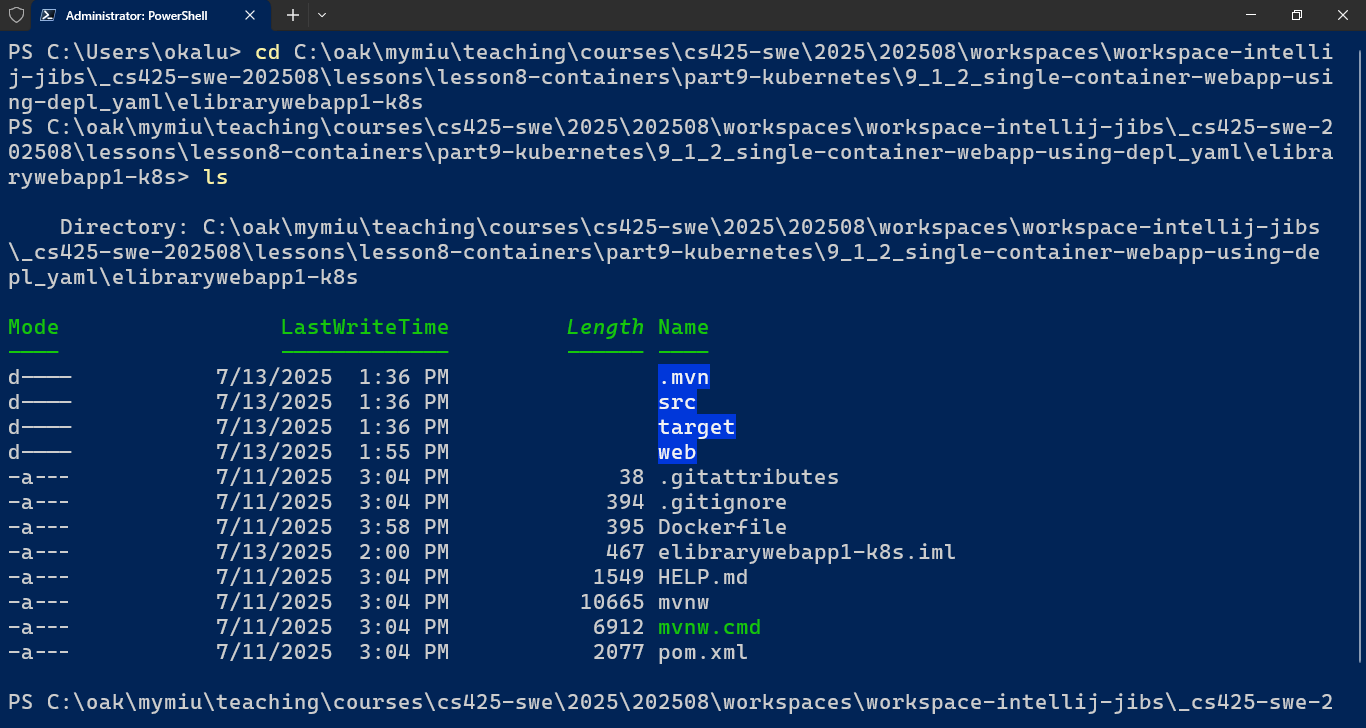
**Deploy a single-container Spring Boot WebApp to Local Kubernetes using deployment.yaml file**

1. Create a Dockerfile for building a container image of the webapp

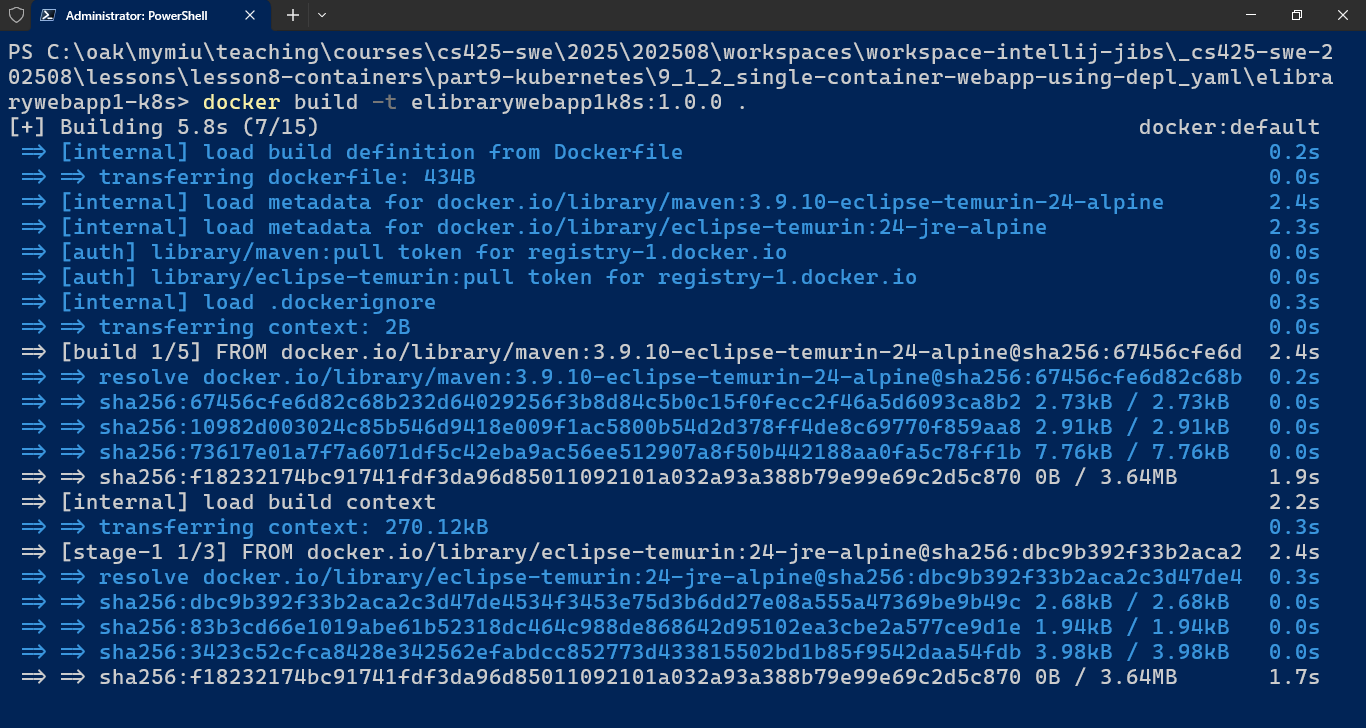


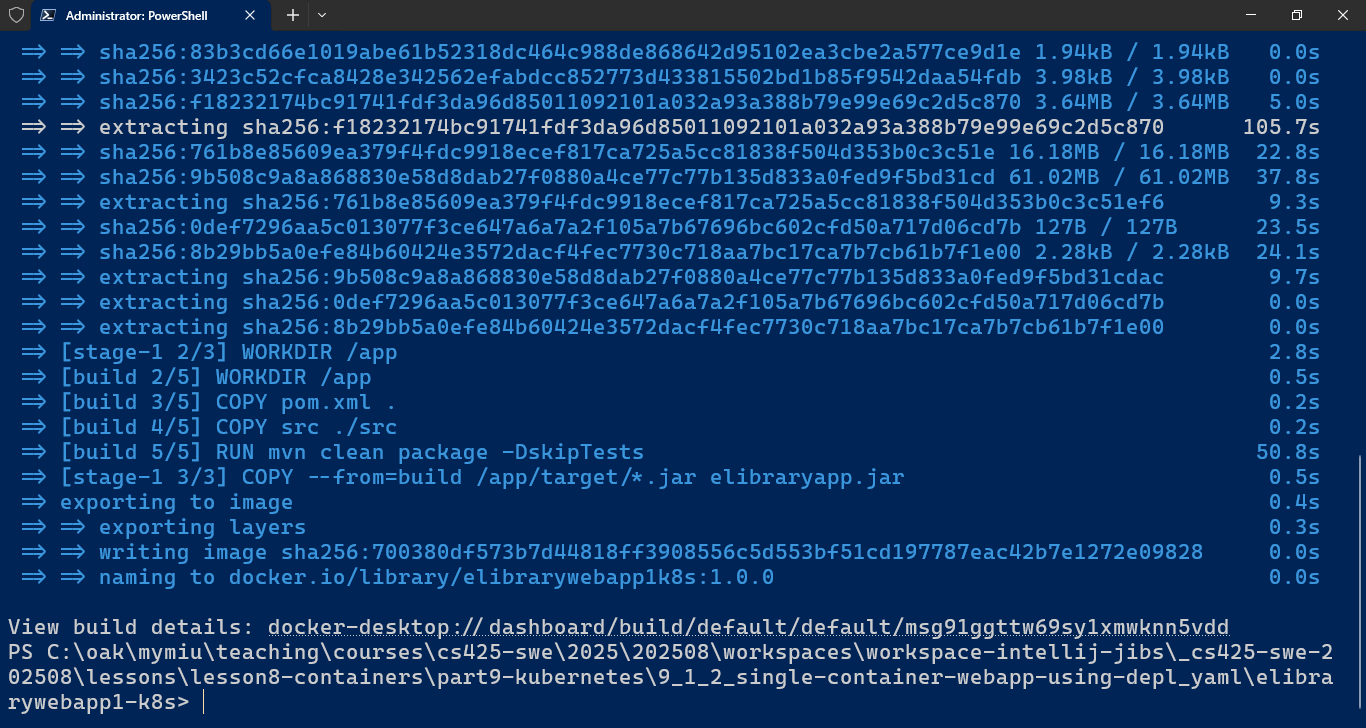
1. Chdir into the webapp’s root directory:

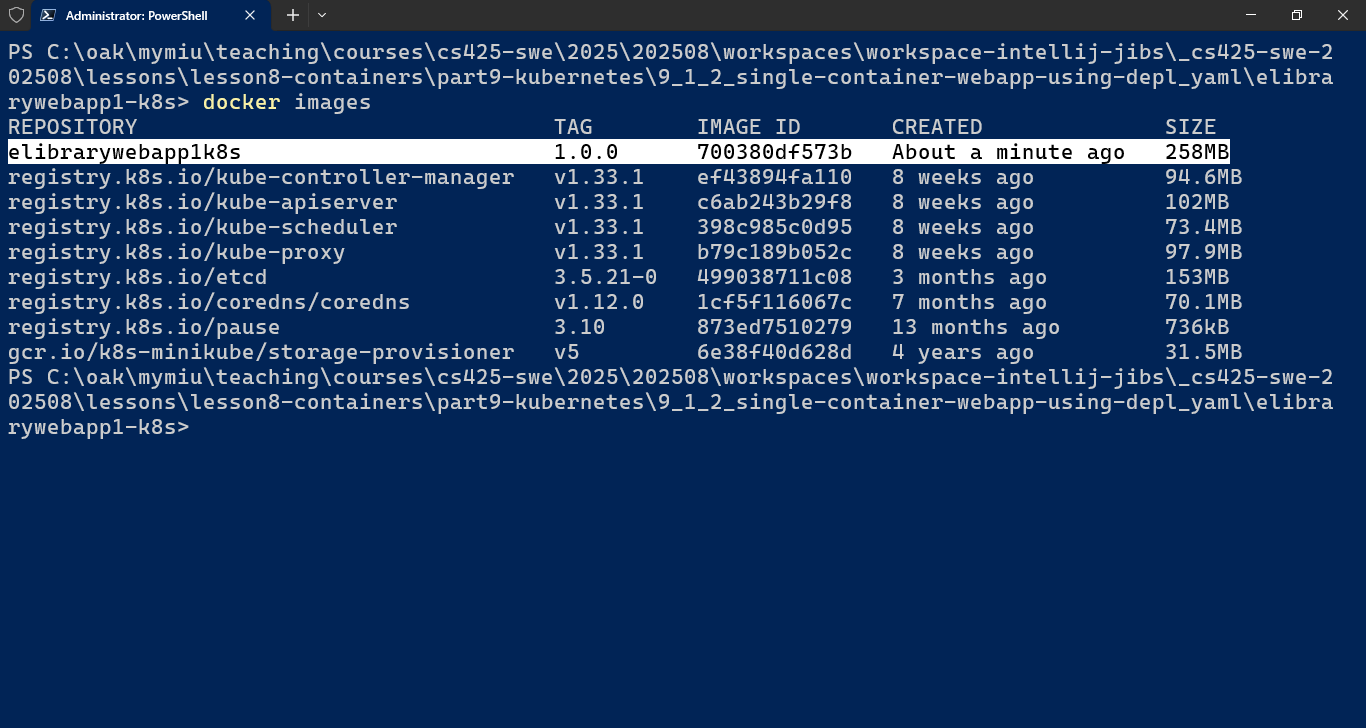


1. Execute Docker CLI cmd to build/create the docker image for the webapp:

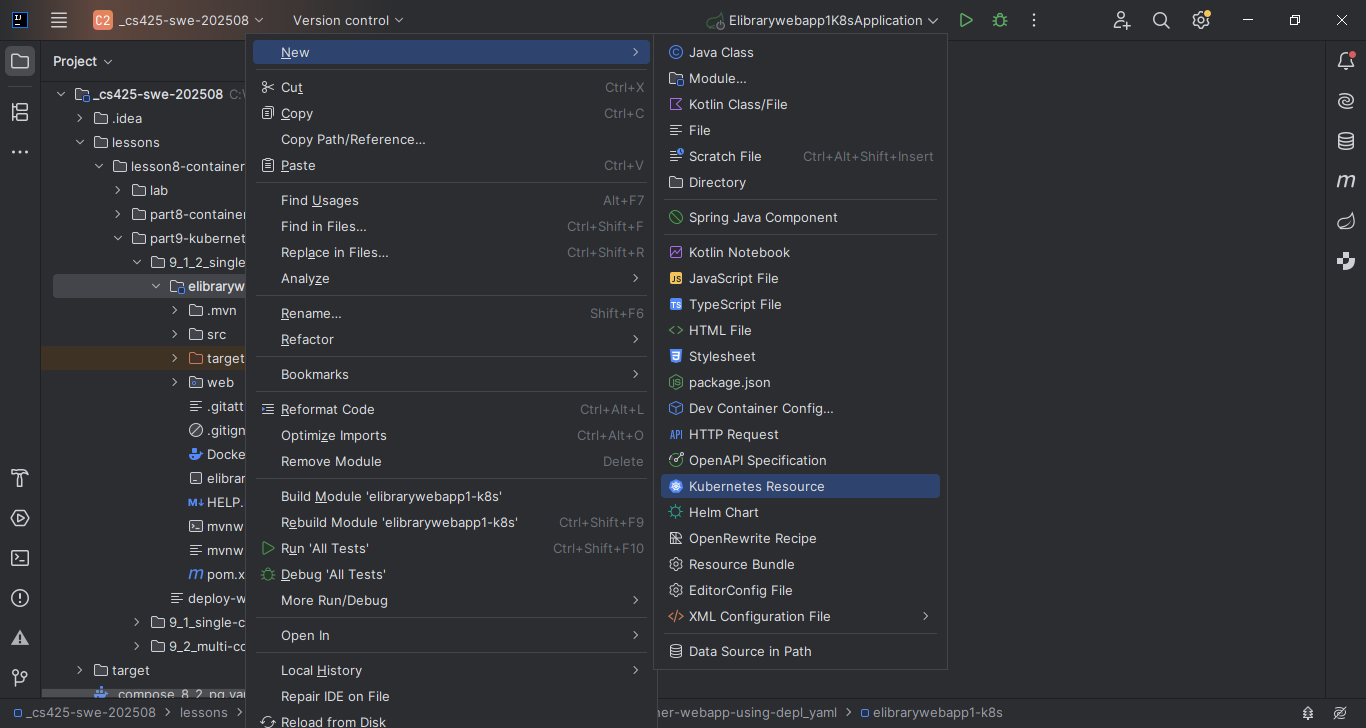
> docker build –t [image-name] .

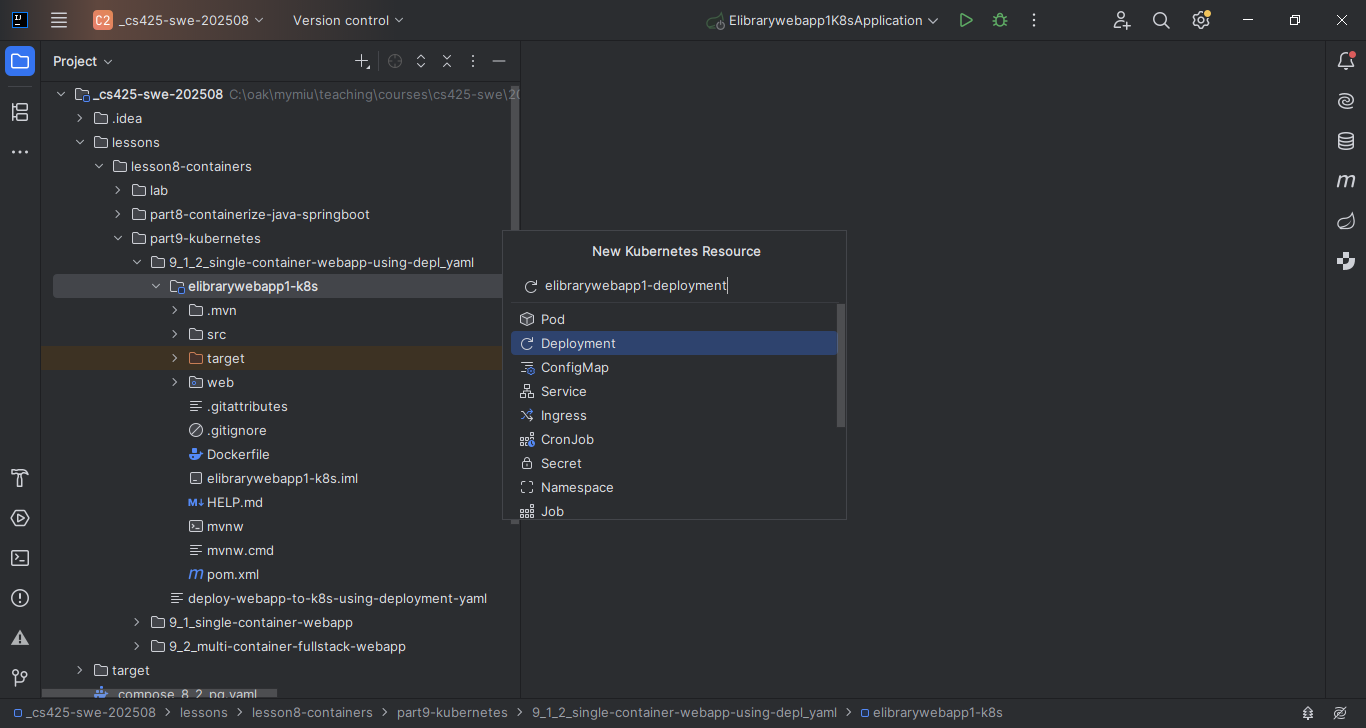


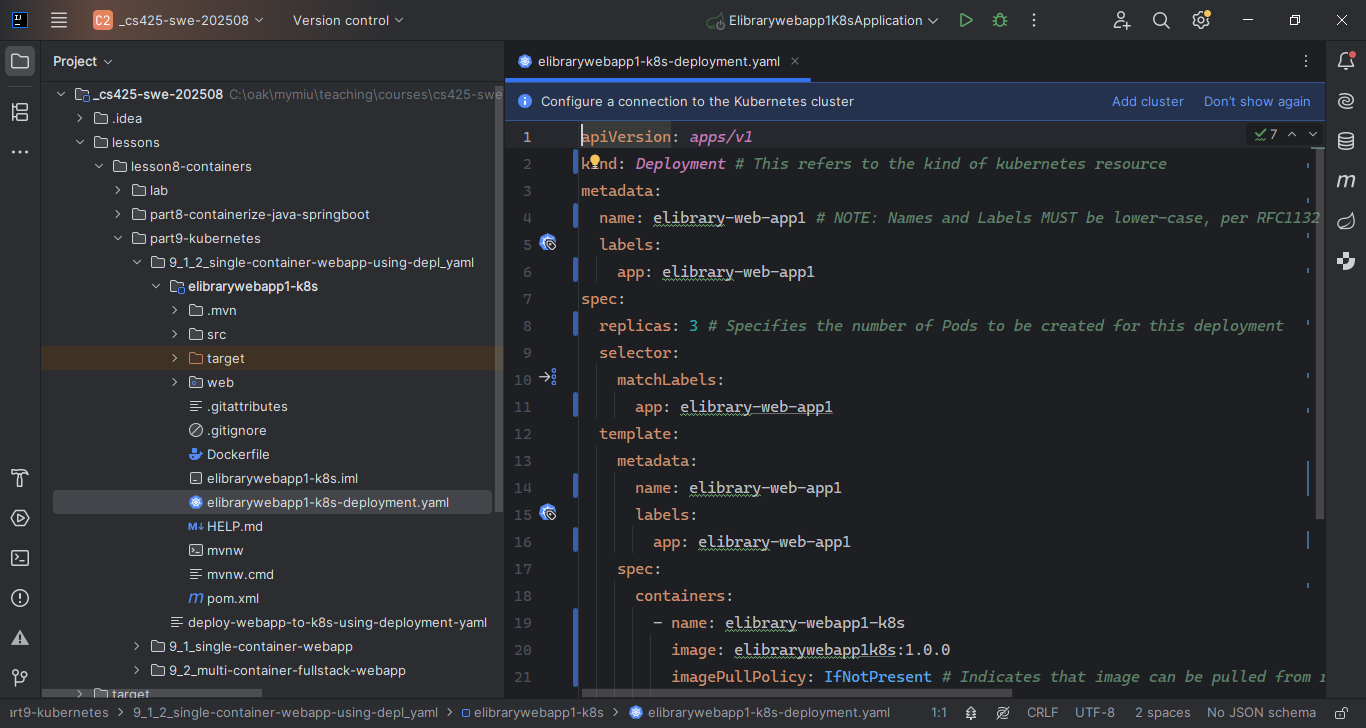


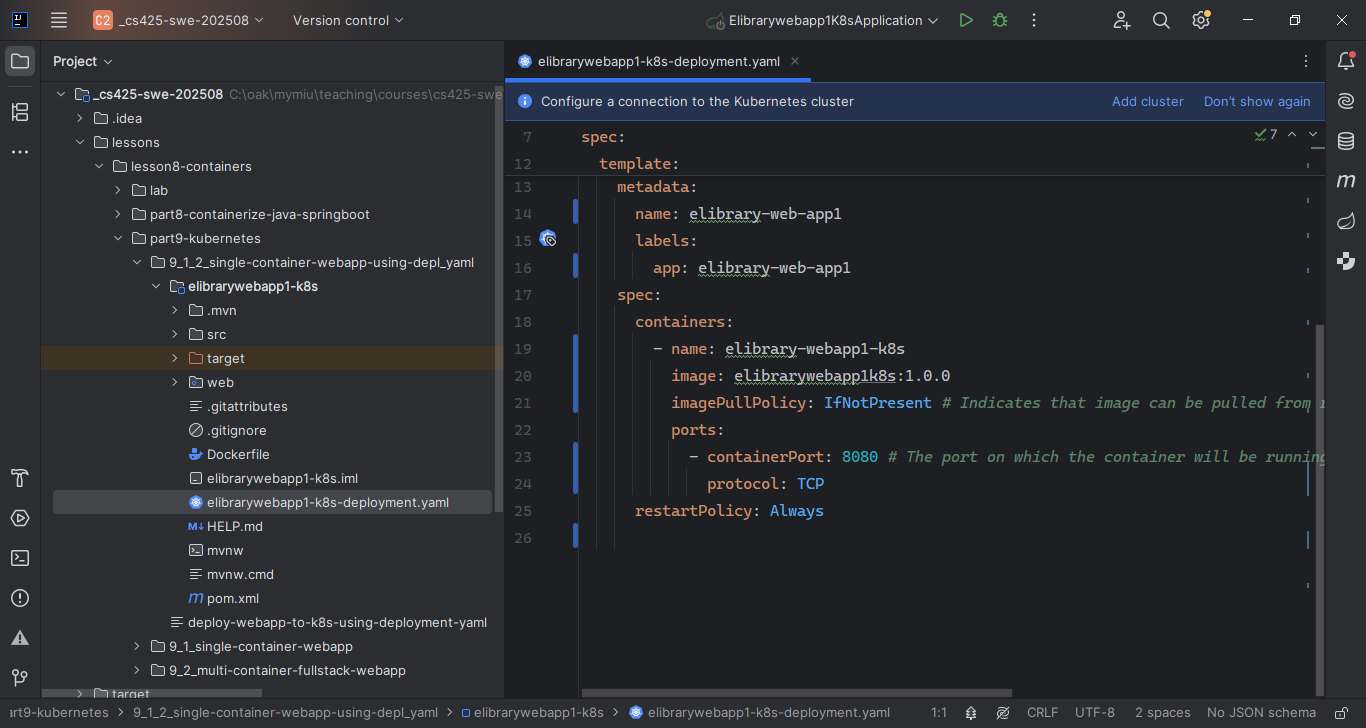


1. Next, to deploy the containerized webapp to the local minikube kubernetes cluster, create a deployment.yaml file



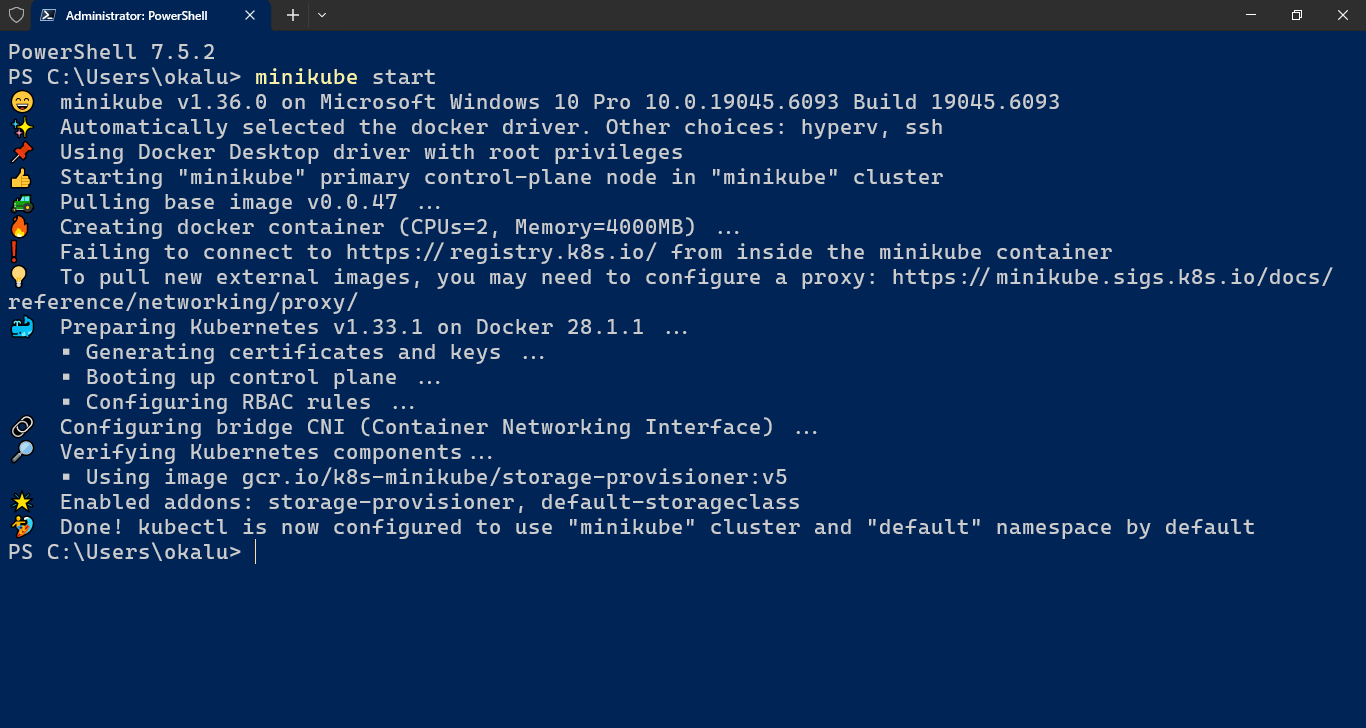




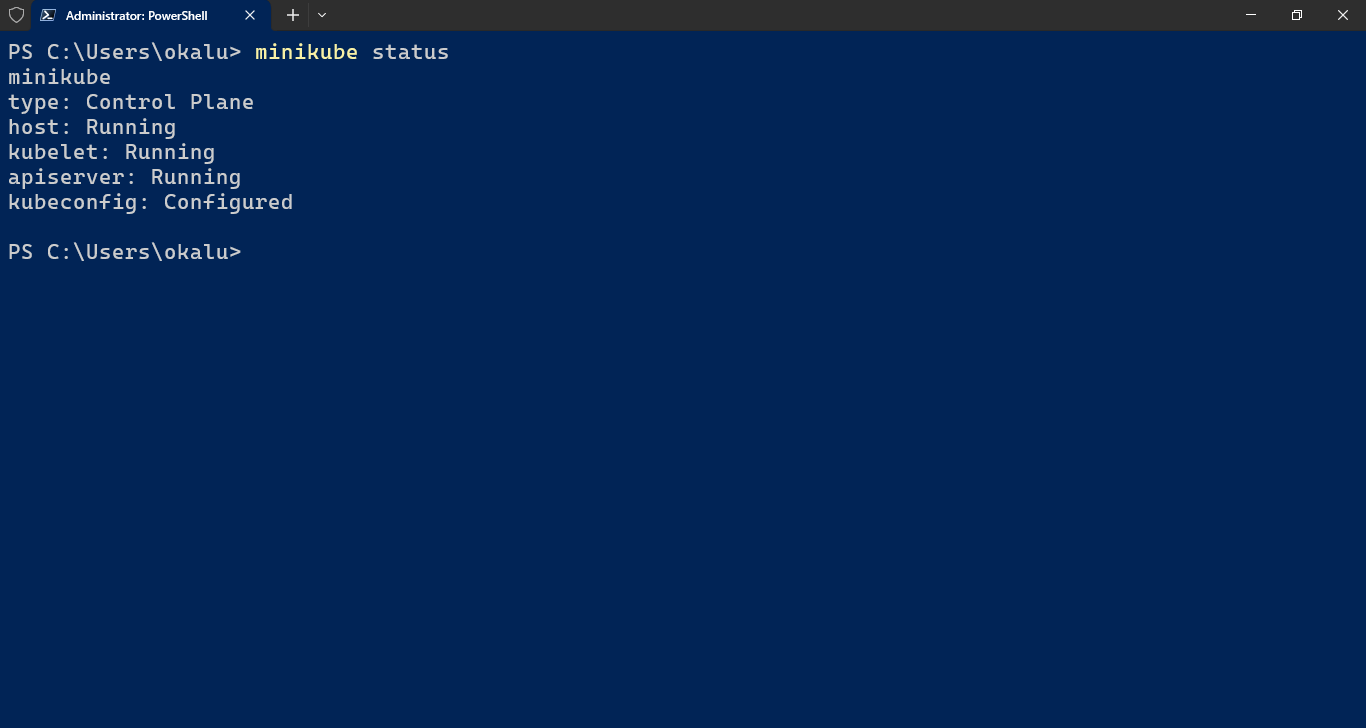


1. With the deployment.yaml file in place, we can execute it to create the desired/specified deployment object in a k8s cluster hosted by minikube, using kubectl cmd.
2. But first, Start Minikube. Open a command line terminal/shell and execute

> *minikube start*

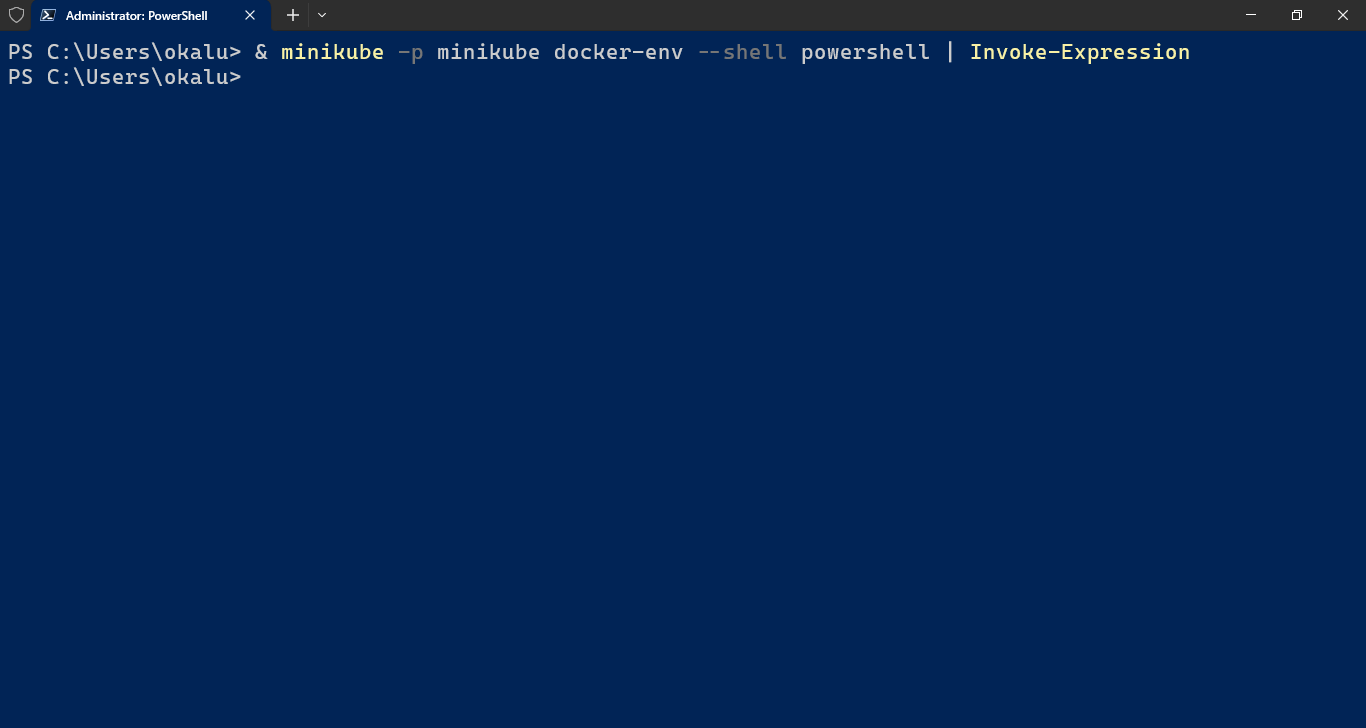


1. Verify that Minikube started and is running: > *minikube status*



1. Enable Docker operations to execute within the Minikube environment, by configuring the Docker CLI to communicate with the Docker daemon within the Minikube cluster. To do this run the appropriate command for your Shell:
   1. For Windows Powershell:

*& minikube -p minikube docker-env --shell powershell | Invoke-Expression*



* 1. For Windows CMD prompt:

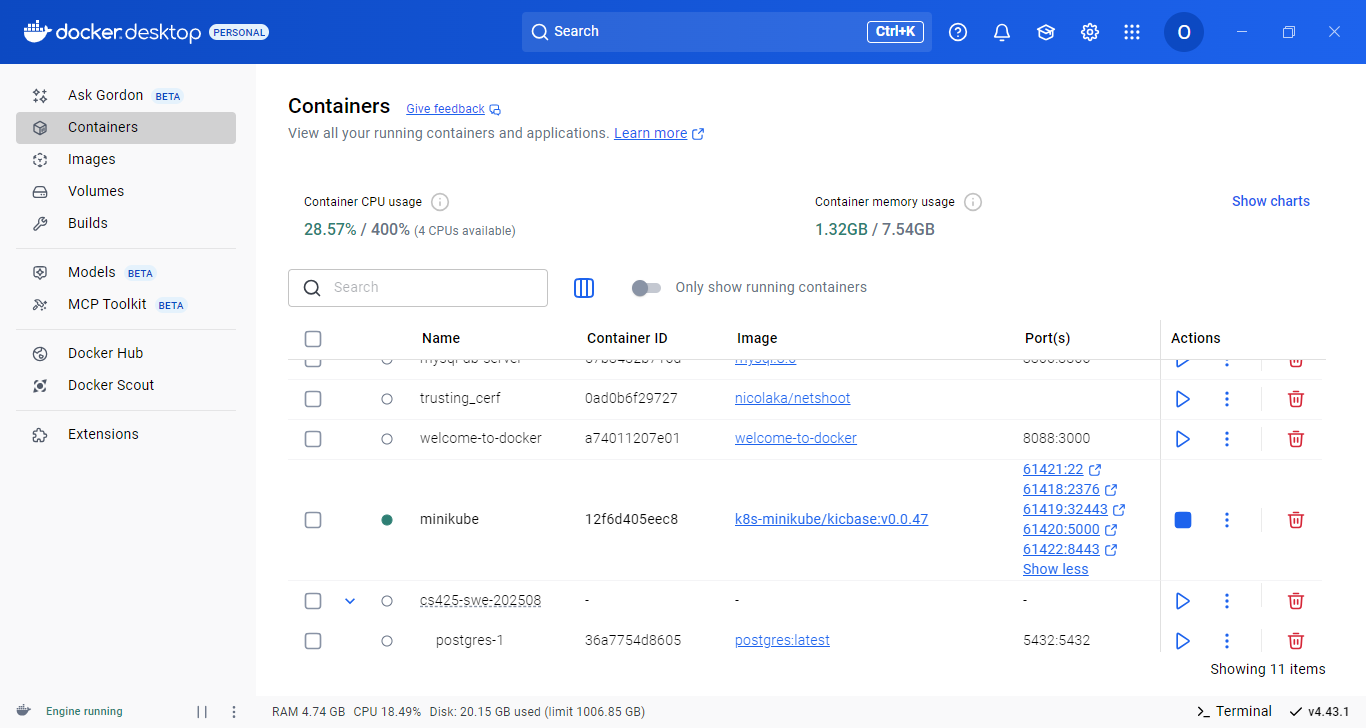
*@FOR /f "tokens=\*" %i IN ('minikube -p minikube docker-env --shell cmd') DO @%i*

* 1. For MacOS bash or zsh etc:

*eval $(minikube docker-env)*

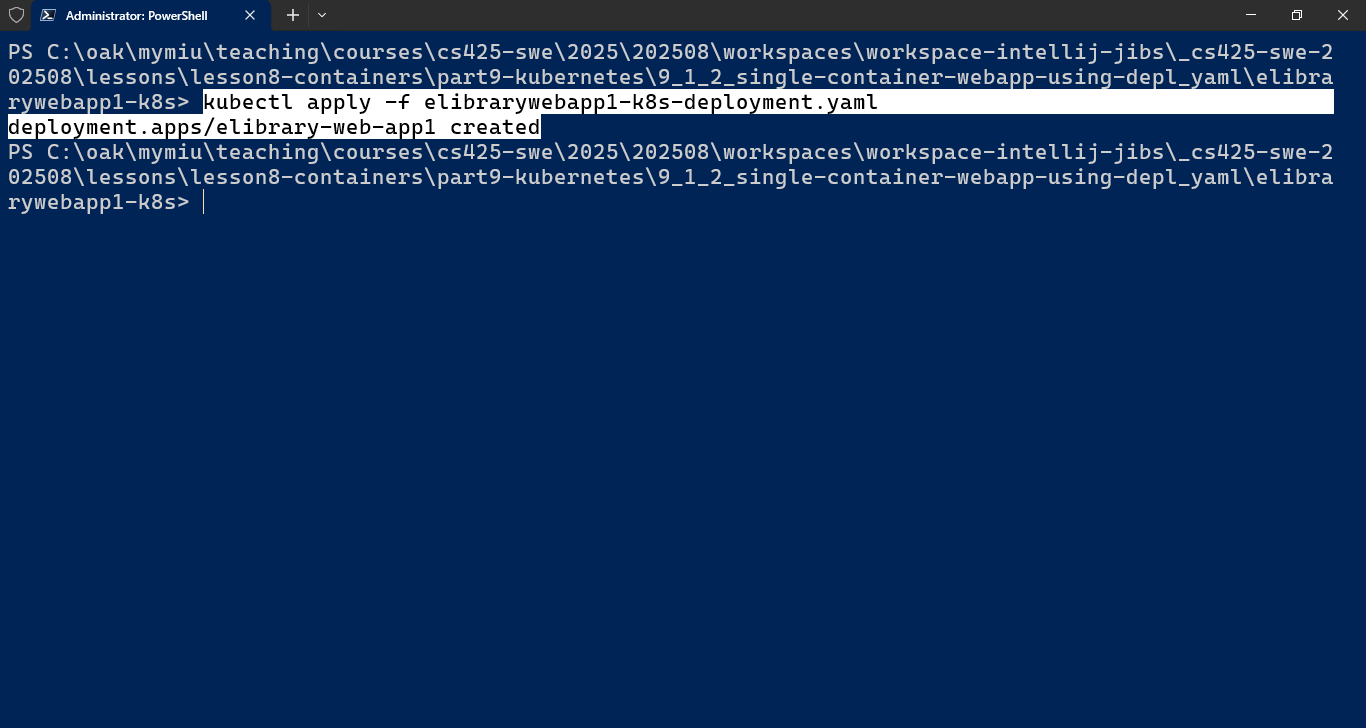
Note: These commands configure the Docker environment only for the current terminal session. Closing the terminal will revert to using the system's default Docker daemon. You must re-run the relevant command each time a new terminal session is opened or the Minikube cluster is restarted, especially with container-based drivers like Docker or Podman.

1. See Minikube running as a container in Docker

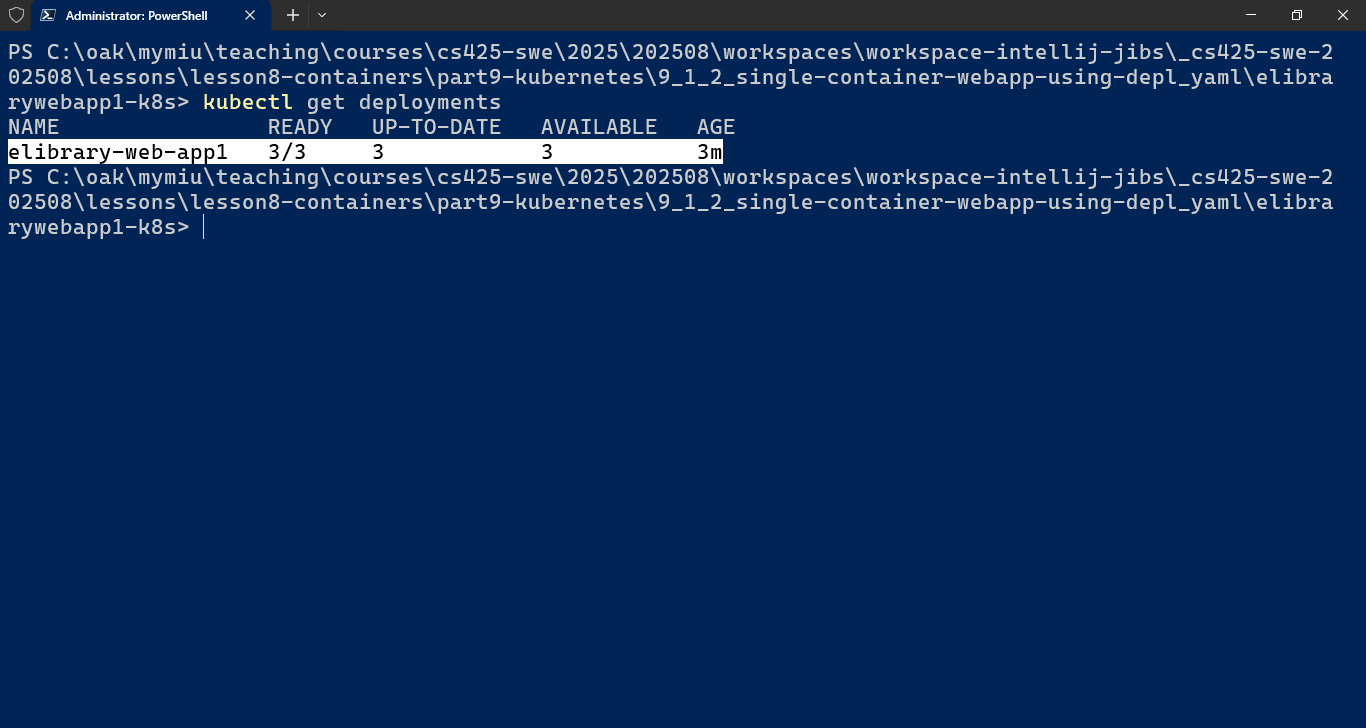


1. Create the desired/specified deployment object by executing the following kubectl cmd.

> kubectl apply –f [deployment-file-name.yaml]

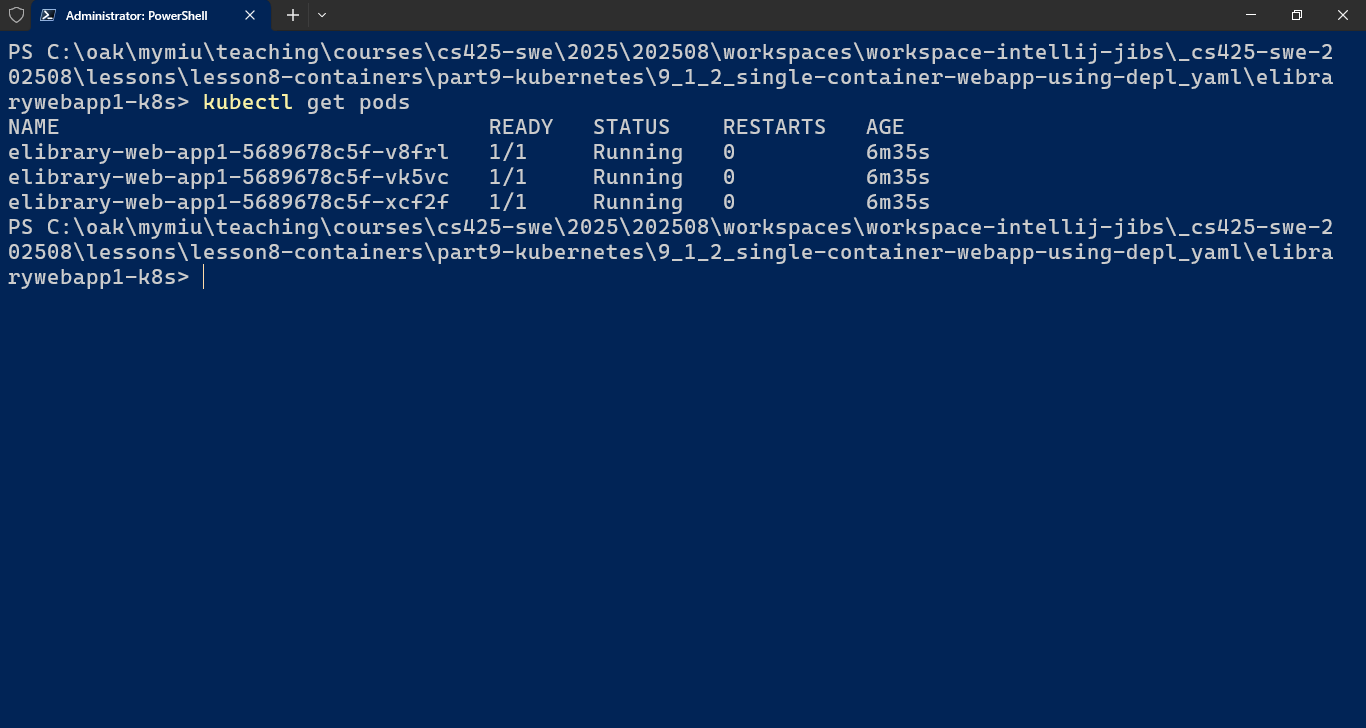


1. Verify the creation > kubectl get deployments

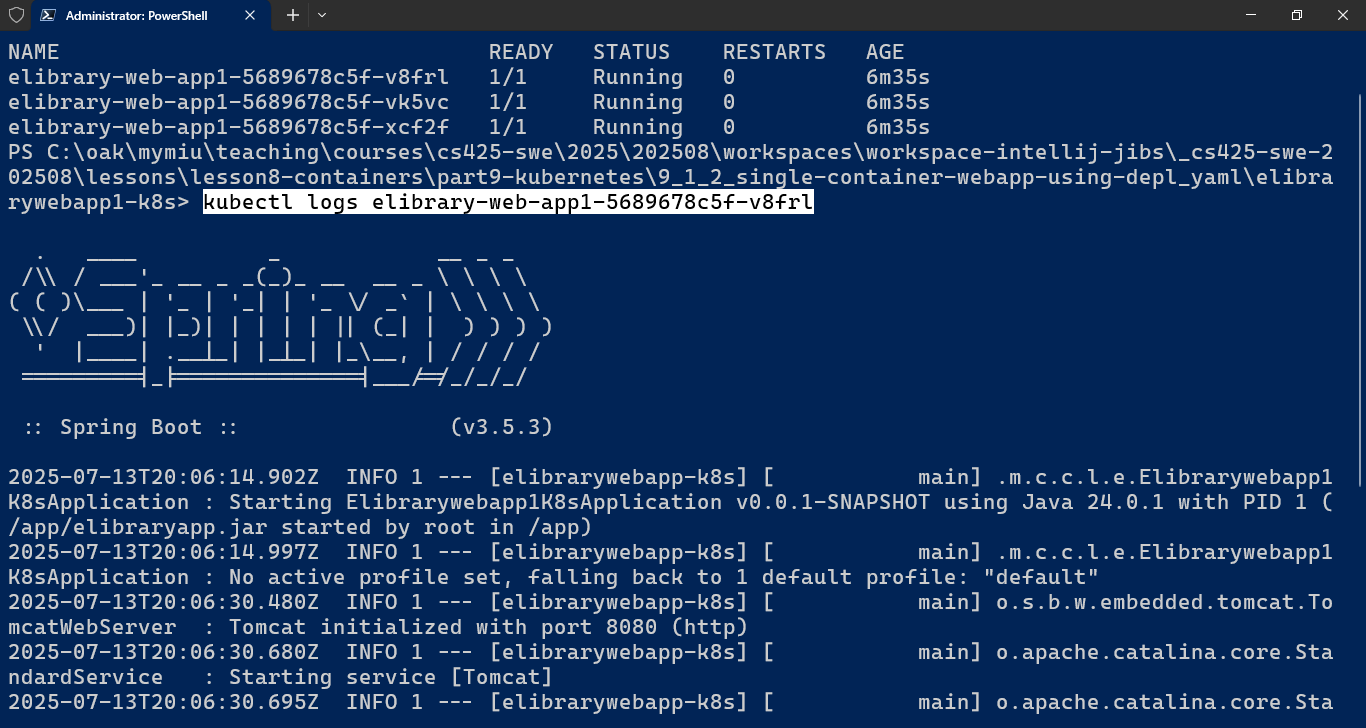


Notice that there are 3/3 Pod Replicas under READY, as specified in the deployment.yaml spec file

1. View the Pods > kubectl get pods



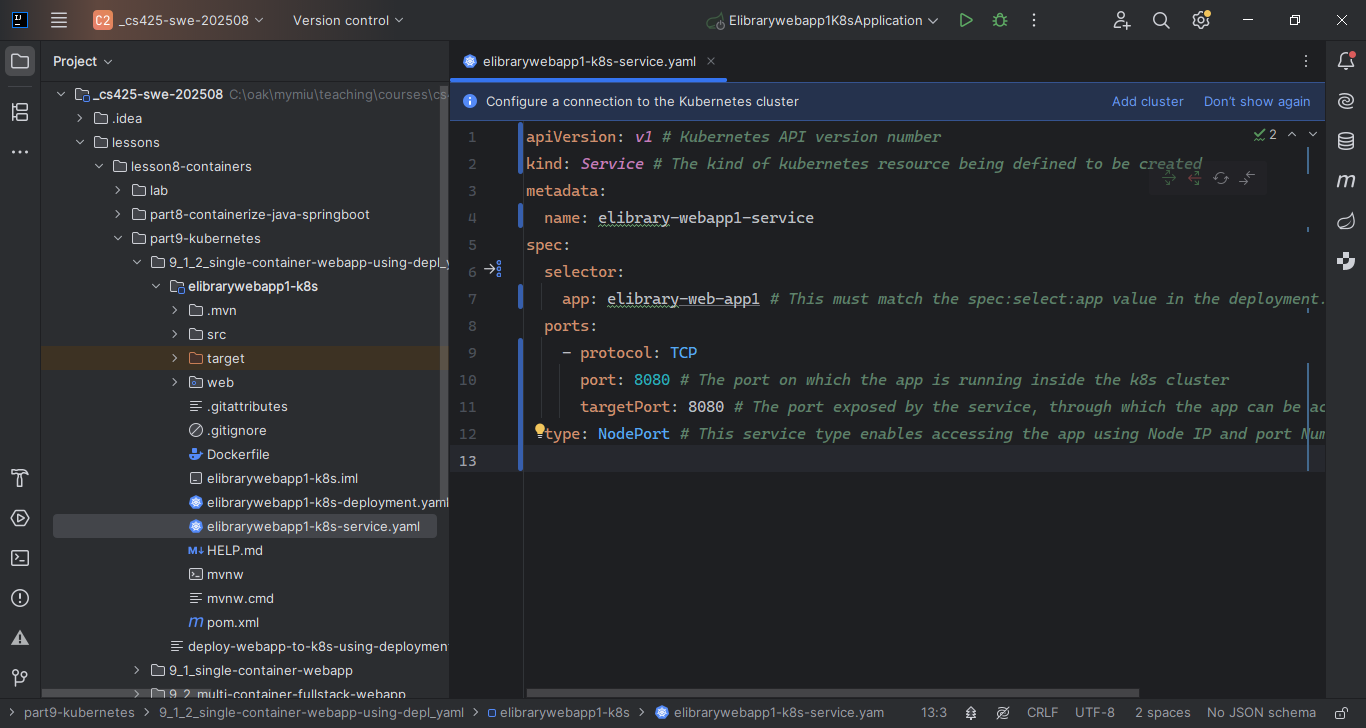
1. View the log from each Pod > kubectl logs [pod-name]



1. View the kubernetes dashboard > *minikube dashboard*

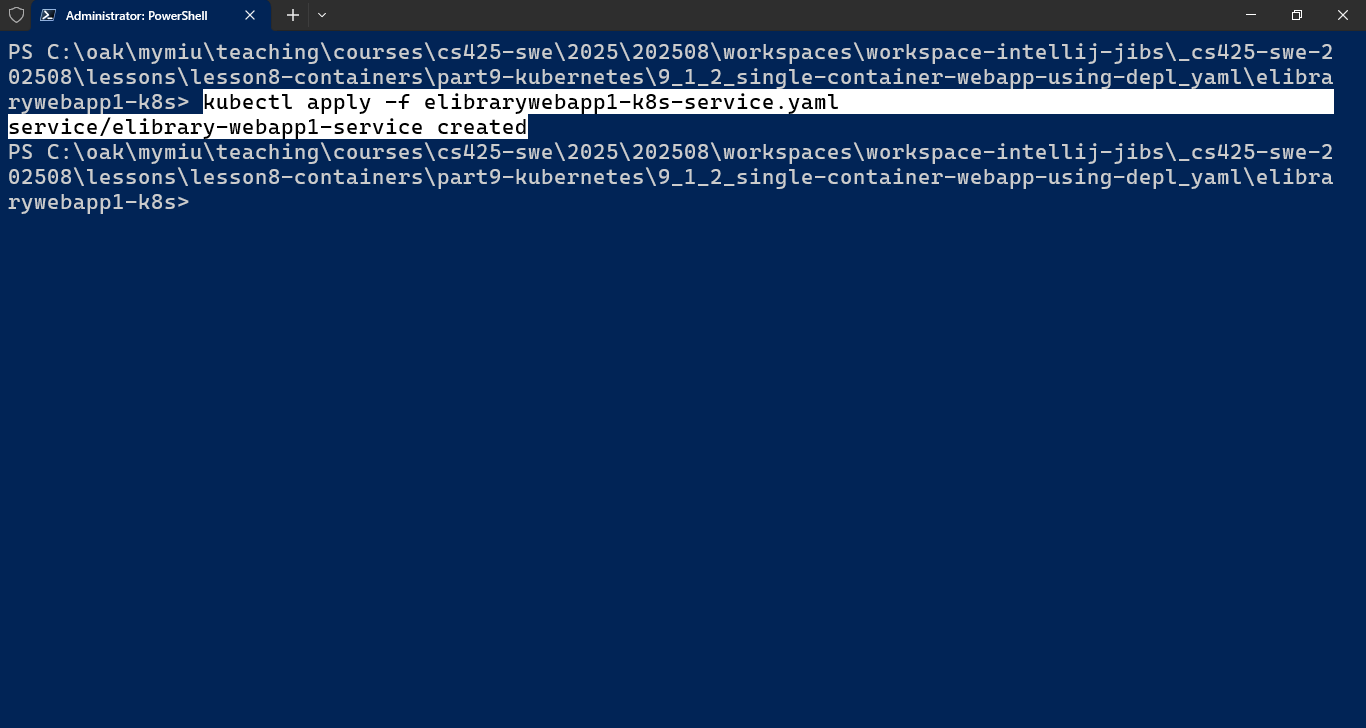


1. Now, since there are 3 Pods (logical grouping of instances of the containerized webapp) deployed/running in the single-node cluster, there is a need for an object/mechanism to expose the access to the deployed containerized webapp and/or handle the routing of requests coming to the application. Hence, a Service object of type NodePort needs to be created.
2. Create a service.yaml file

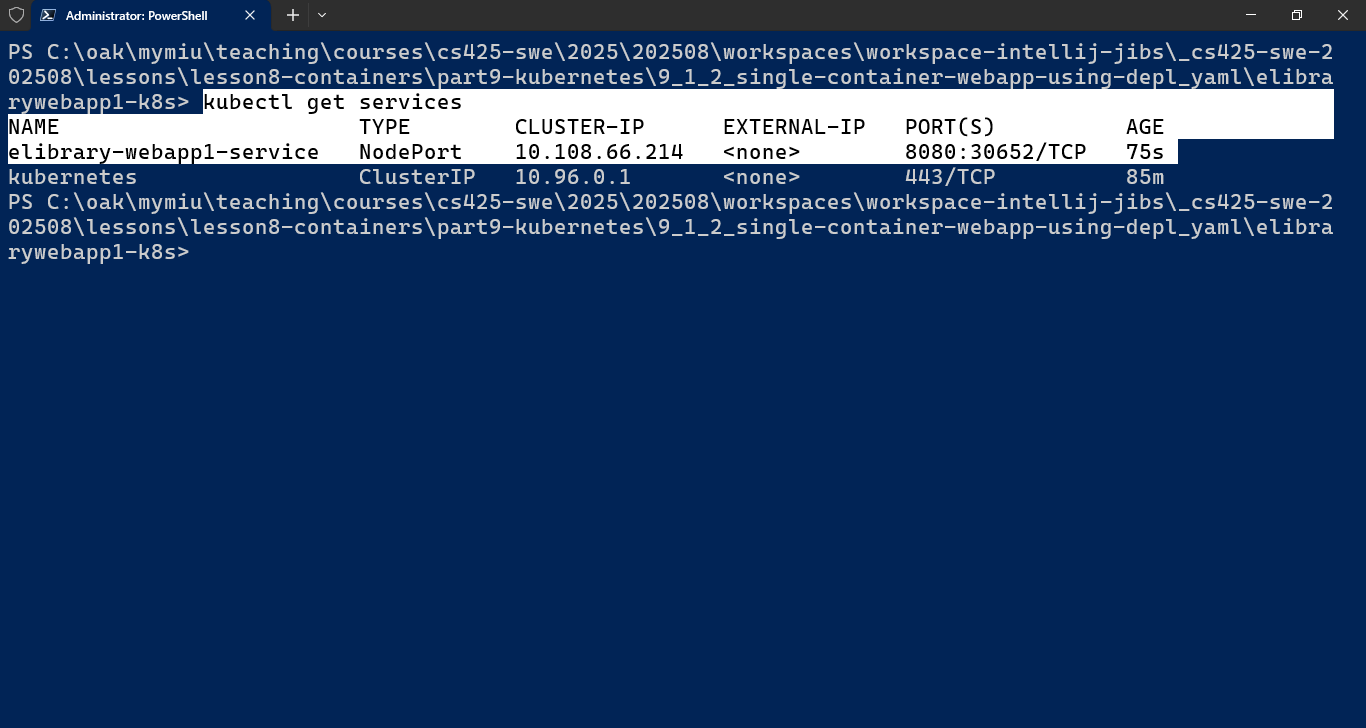


1. Execute kubectl cmd to create the service

> kubectl apply –f [svc-file-name.yaml]

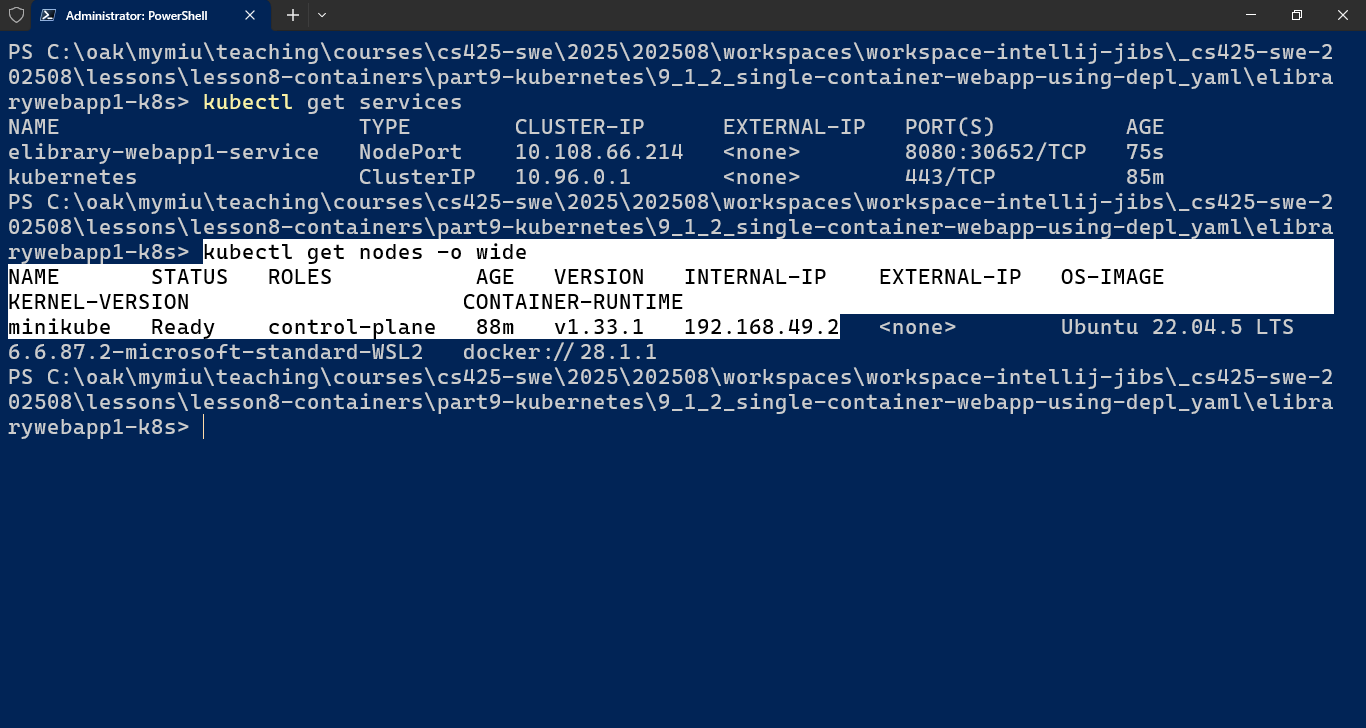


1. See the service created > kubectl get services



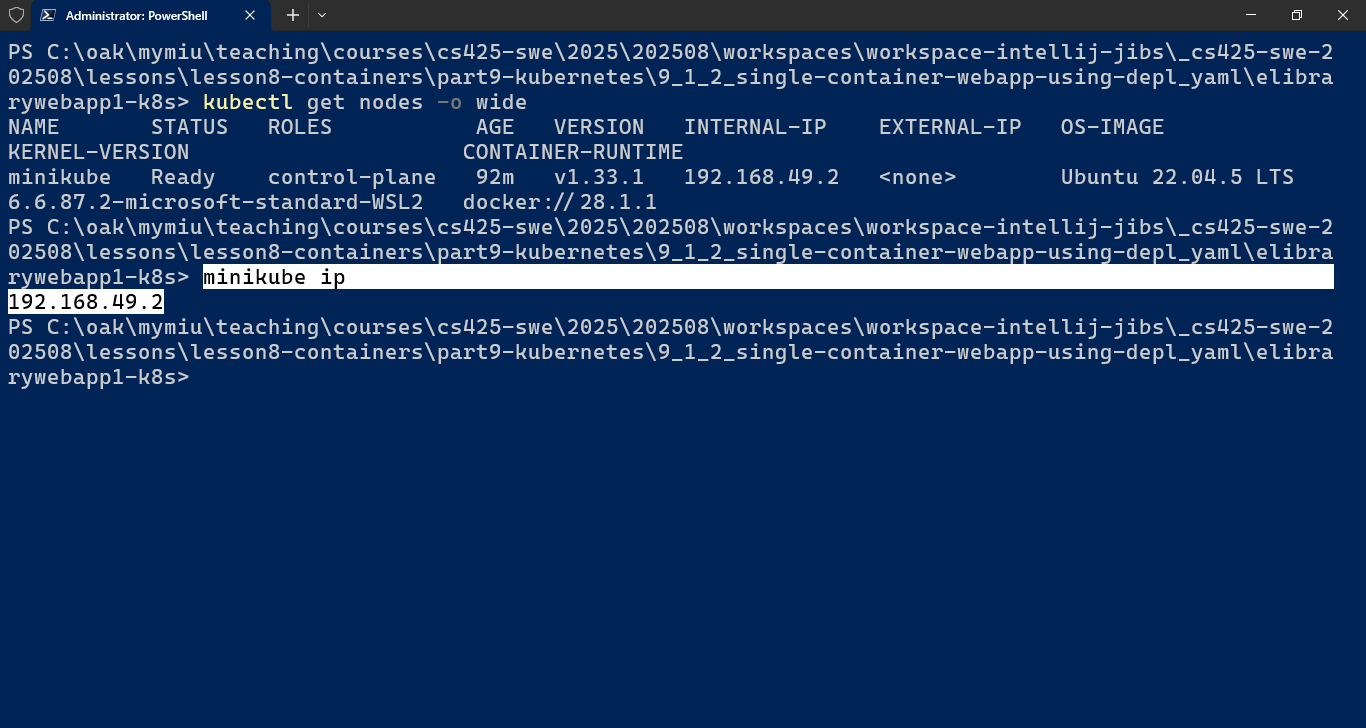
Notice the Port number on which the service is exposed on.

1. To get the IP address > *kubectl get nodes -o wide*



Alternatively, view/obtain the Minikube IP address to use for accessing the service

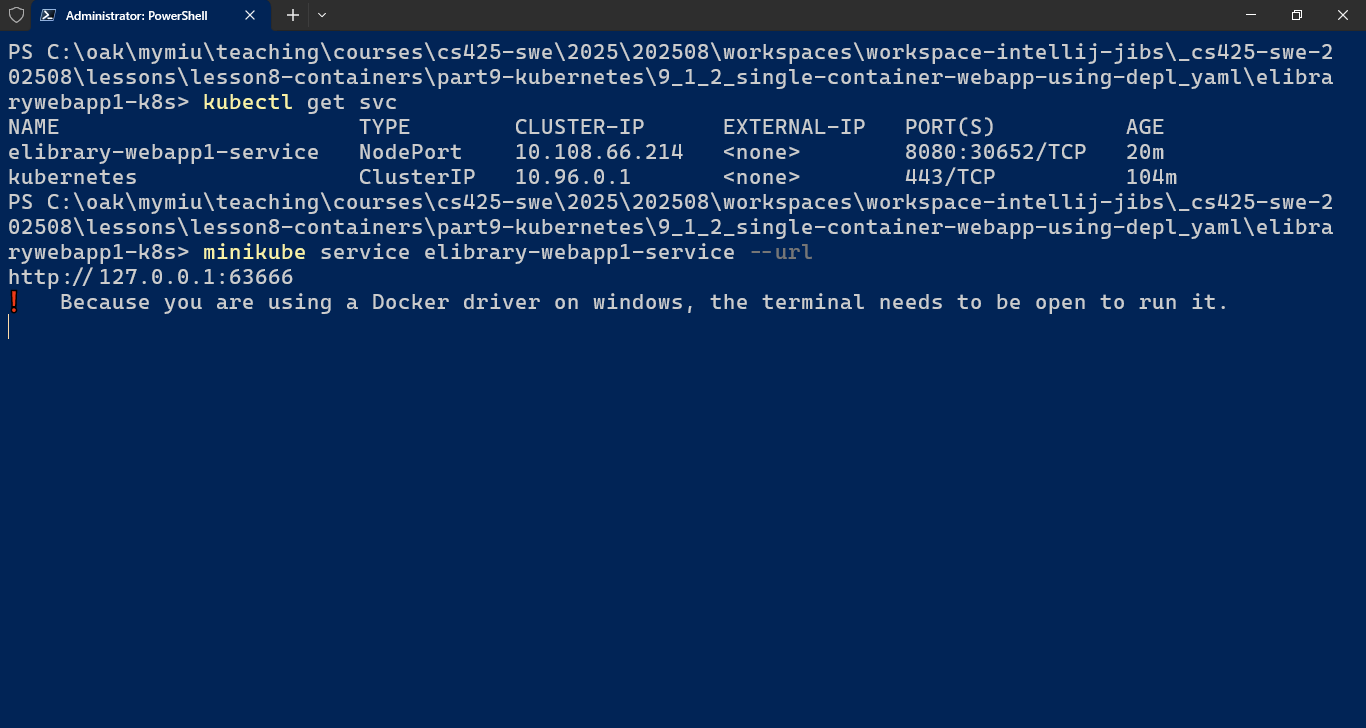
> minikube ip



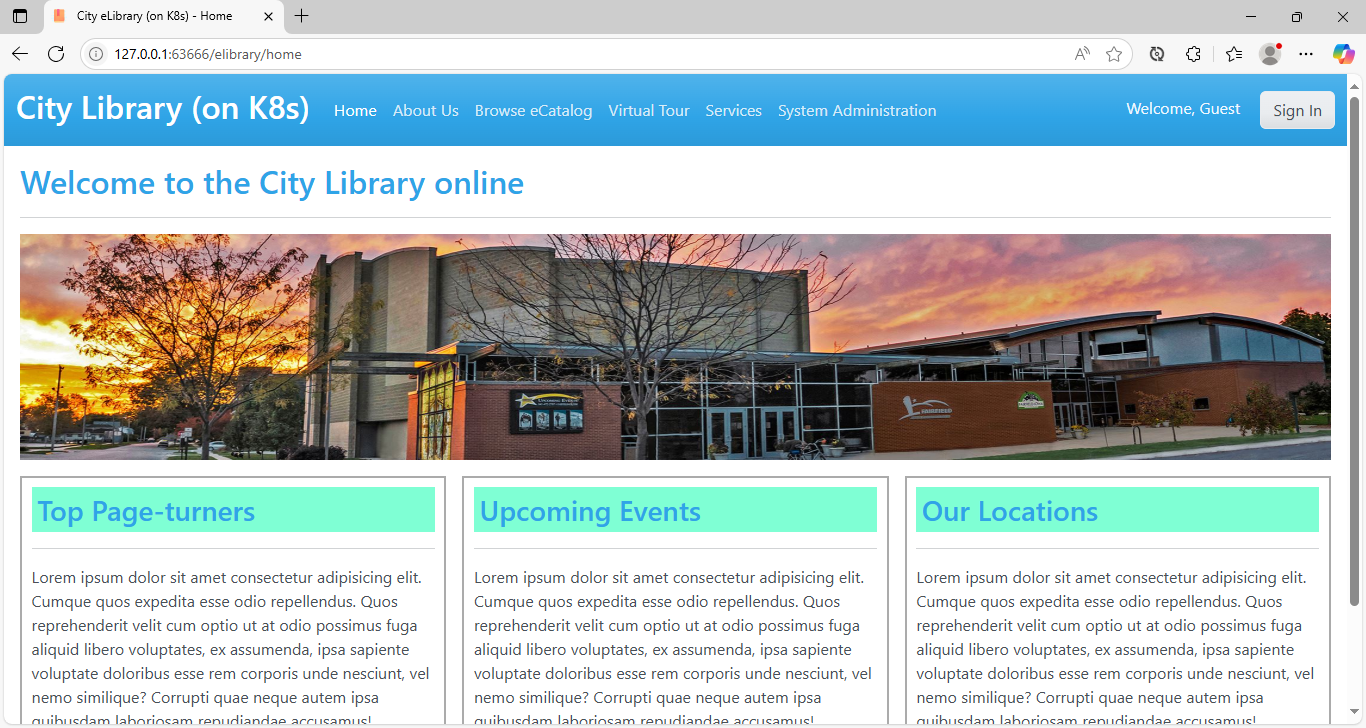
The above IP and Port will probably work for Linux and MacOS systems.

1. For Windows, to obtain the External IP and Port number for accessing the webapp running inside the k8s cluster, from the outside, execute

> minikube service [svc-name] --url

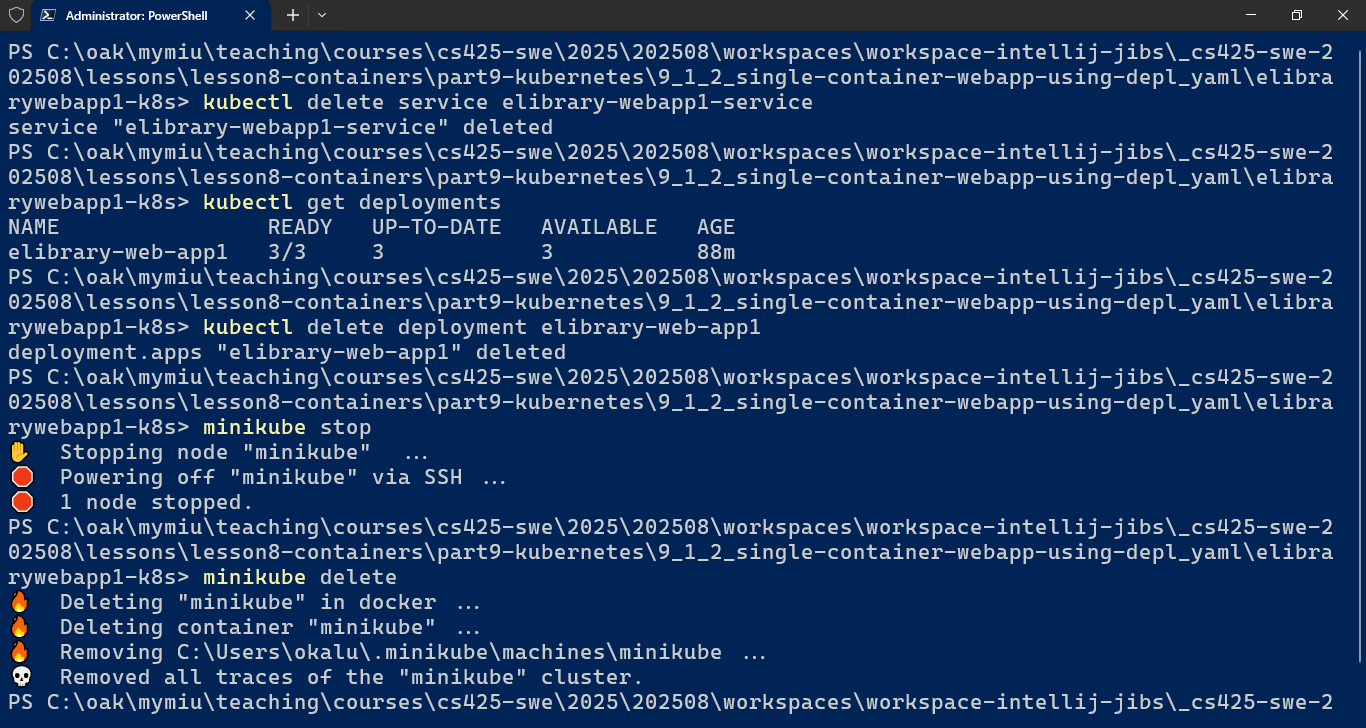


1. Now, access the webapp using the URL (with IP and PortNumber) given:



**Note: Make sure to keep the Terminal/Shell with the minikube service... cmd open/running!!!**

1. Finally, clean up the environment by deleting the service, deployment, stop minikube and delete minikube containers



1. ... The End!!!