1. Kubectl cluster-info
2. kubectl create ns mynamespace
3. kubectl run httpd --image=httpd -n mynamespace
4. kubectl create deployment kubecont1 --image=nginx:1.23 --replicas=5

kubectl expose deployment kubecont1 --type=NodePort --port=80

kubectl get services kubecont1

kubectl get rs

kubectl get services

kubectl get pods

1. kubectl create deployment kubecont1 --image=nginx:1.23 --replicas=3

kubectl get services

1. kubectl set image deployment/kubecont1 nginx=nginx:1.24

verify the updates: kubectl describe deployment kubecont4

Maintain History: kubectl rollout history deployment/kubecont1

1. kubectl rollout undo deployment/kubecont1
2. kubectl expose deployment kubecont1 --type=NodePort --port=80

Access url: kubectl port-forward service/kubecont4-service 8080:80

Go to browser and search for the localhost port

1. docker pull ubuntu

kubectl run ubuntu-pod --image=ubuntu:latest --command -- /bin/bash -c "echo 'hello world'"

kubectl describe pod ubuntu-pod

1. kubectl get pods

kubectl exec -it -n <namespace> <pod-name> -- /bin/bash

1. kubectl logs my-pod -c my-container
2. A "context" refers to the set of files and directories that are sent to the Docker daemon when you build a Docker image. When you issue a docker build command, Docker needs to access all the necessary files required to create the image. This collection of files and directories is referred to as the "build context."

The build context is sent to the Docker daemon and serves as the basis for building the Docker image. It includes the Dockerfile, along with any other files that are referenced in the Dockerfile's COPY or ADD instructions.

When you run the docker build command, the build context is compressed into a tarball, and then it's sent to the Docker daemon. The Docker daemon then unpacks the tarball and uses the files inside it to create the image layer by layer.

By managing the build context efficiently, you can optimize the Docker image-building process and reduce image sizes, leading to faster and more efficient Docker builds.

1. A "config file" typically refers to configuration files used within a Docker container that is running an Ubuntu-based image.

When you run a Docker container, you often need to configure the application or service running inside the container. Configuration files are used to set various parameters, customize behavior, and provide settings specific to the environment in which the application is deployed.

To work with configuration files in a Docker container, you can use the following techniques:

Building the configuration into the Docker image: You can create a custom Docker image that includes the necessary configuration files as part of the image. When you run a container using this image, the configuration files will be available inside the container.

Mounting configuration files as volumes: Another approach is to mount configuration files from the host machine into the container as volumes. This allows you to modify the configuration outside the container and have the changes reflected inside the running container without rebuilding the Docker image.