Day-1 Activates of K8S

Activity 1.1 : To Setup Docker Development Environment on CentOS (Docker Installation)

wget <https://download.docker.com/linux/centos/7/x86_64/stable/Packages/docker-ce-18.09.4-3.el7.x86_64.rpm>

wget https://download.docker.com/linux/centos/7/x86\_64/stable/Packages/containerd.io-1.2.5-3.1.el7.x86\_64.rpm

wget https://download.docker.com/linux/centos/7/x86\_64/stable/Packages/docker-ce-cli-18.09.4-3.el7.x86\_64.rpm

yum install -y containerd.io-1.2.5-3.1.el7.x86\_64.rpm

yum install -y docker-ce-cli-18.09.4-3.el7.x86\_64.rpm

yum install docker-ce-18.09.4-3.el7.x86\_64.rpm

systemctl status docker

systemctl start docker :(Run this command start docker)

systemctl status docker

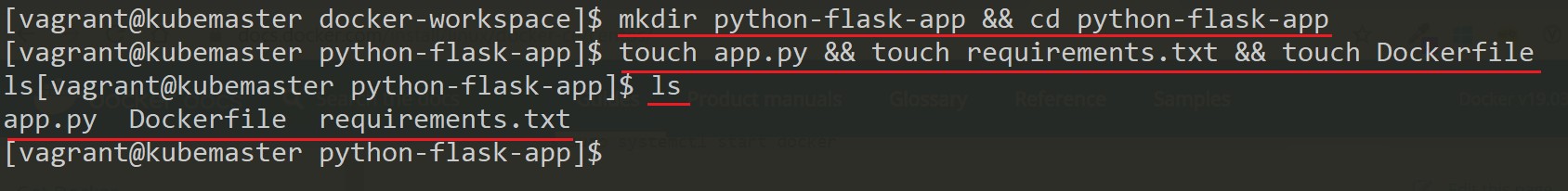
docker run hello-world (Run this command everything working in Docker)

docker images.

## Activity 1.2 - Dockerizing an App

mkdir python-flask-app && cd python-flask-app

1. Create a new app.py, requirements.txt and Dockerfile file using the below command



touch app.py && touch requirements.txt && touch Dockerfile ls

1. Add the below content to the respective file as given. Use any text editor of your choice either vim, nano, gedit

### app.py

**from** flask **import** Flask

**import** os

**import** socket

app = Flask( name )

**@app.route("/") def hello**():

html = "<h3>Welcome {name}!</h3>" \ "<b>Hostname:</b> {hostname}<br/>"

**return** html.format(name=os.getenv("NAME", " from LKM Delivery Team of Accenture"), hostname=socket.gethostname())

**if** name == " main ": app.run(host='0.0.0.0', port=80)

**requirements.txt**

Flask

**Dockerfile**

*# Use an official Python runtime as a parent image*

**FROM** python:2.7-slim

*# Set the working directory to /app*

**WORKDIR** app

**ADD** . ./

*# Install any needed packages specified in requirements.txt*

**RUN** pip install --trusted-host pypi.python.org -r requirements.txt

*# Make port 80 available to the world outside this container*

**EXPOSE** 80

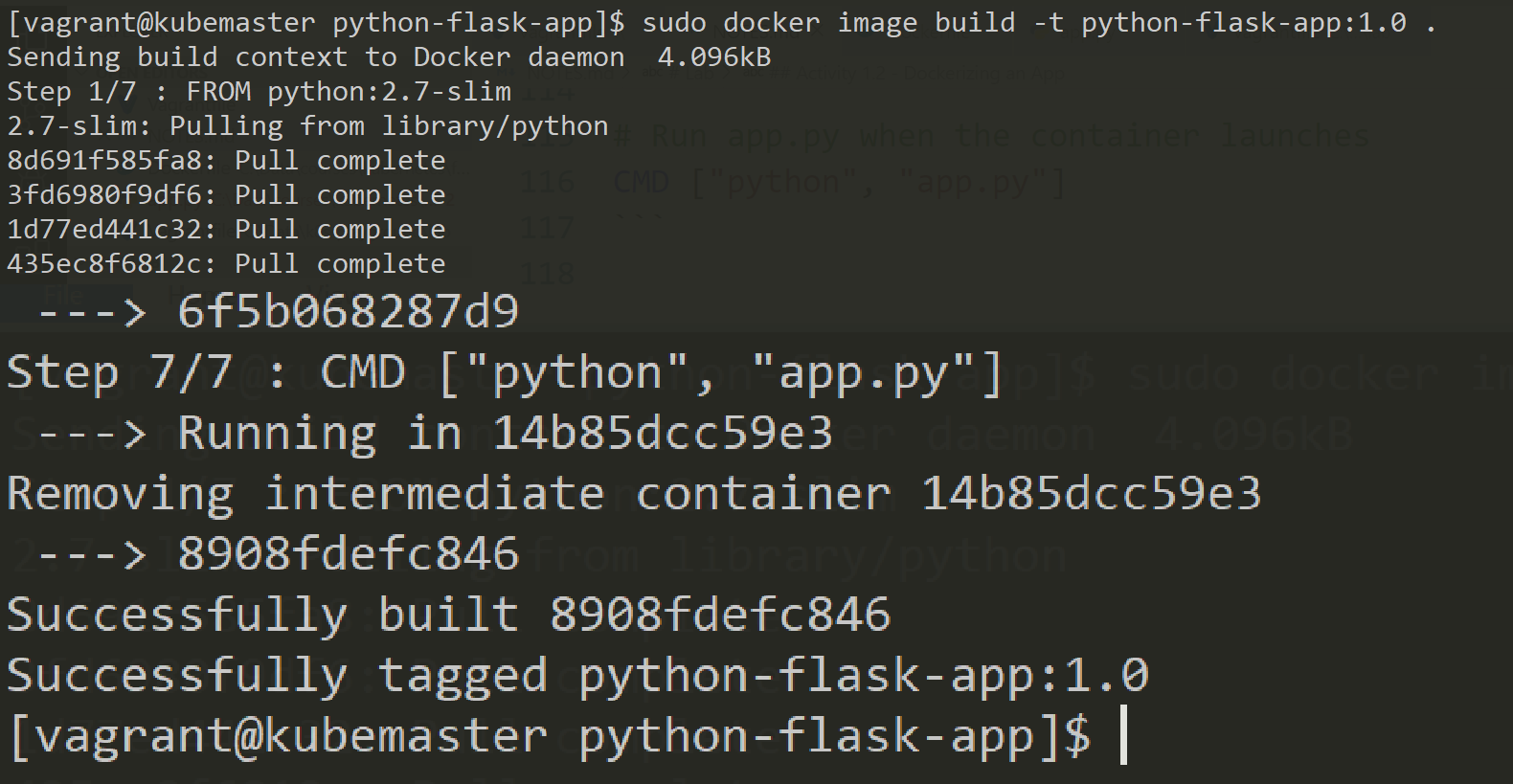
*# Define environment variable Acc LKM*

**ENV** NAME '<YOUR FULL NAME>'

*# Run app.py when the container launches*

**CMD** ["python", "app.py"]

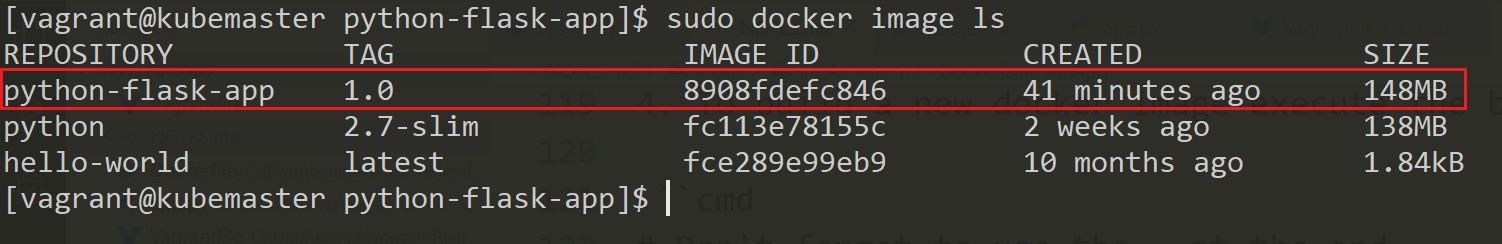
1. To build a new docker image execute the below command



# Don't forget to use the . at the end

sudo docker image build -t python-flask-app:1.0 .

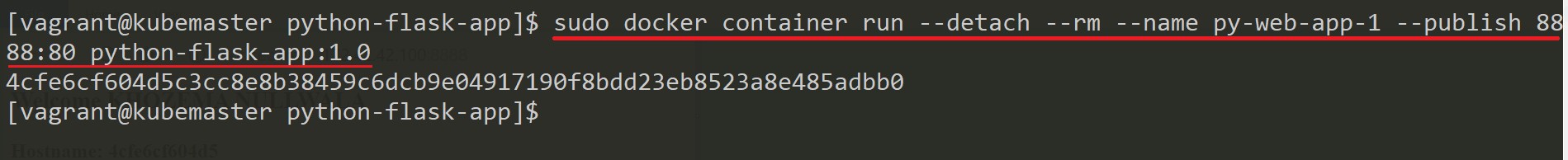
1. To see the newly created docker image execute the below command



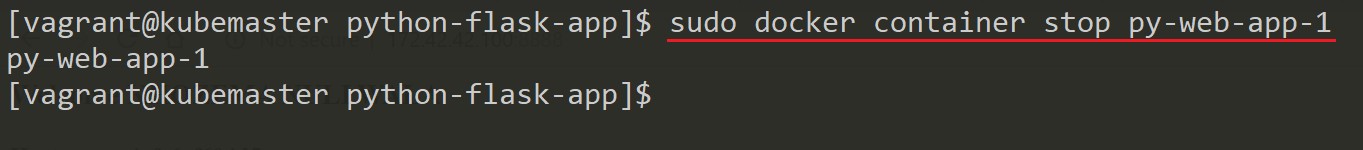
sudo docker image ls

1. To view the content open a new browser tab and use localhost with port number 8888 to view the content of your application

sudo docker container run –detach –rm –name py-web-app-1 –publish 8888:80 python-flask-app:1.0

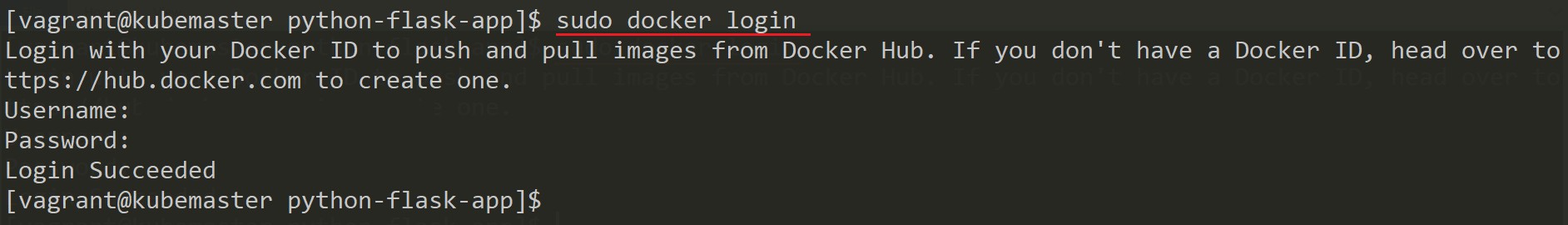


1. To stop the running container execute the command below -



sudo docker container stop py-web-app-1

1. To push the image to your repository ensure that you have a validated account on https://hub.docker.com else you can create a new one.
2. To log in to your account on the docker hub registry use the below command and provide your docker hub id and password -

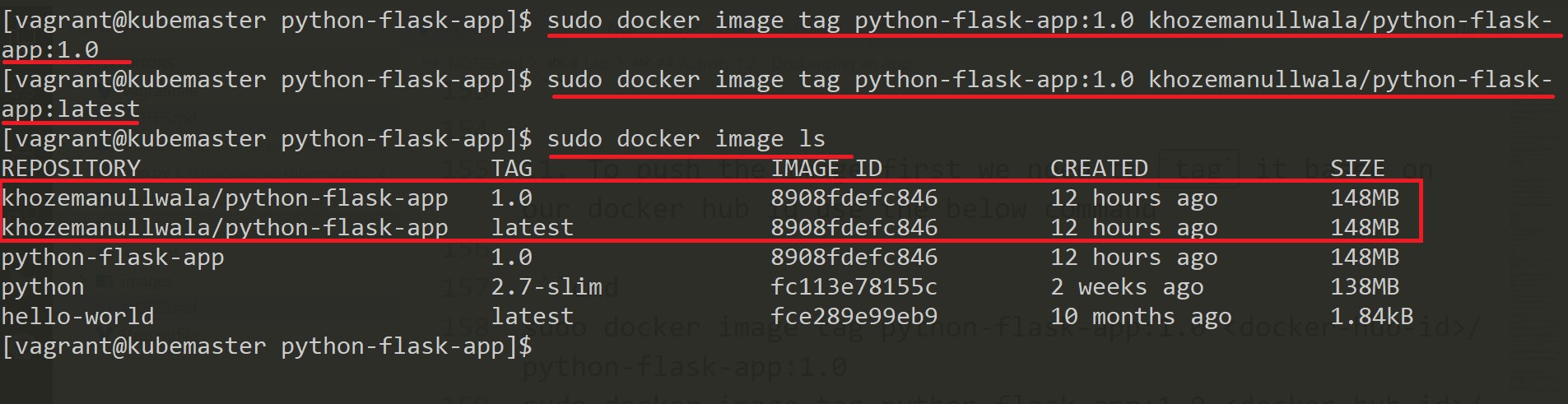


sudo docker login

1. To push the image first we need to tag it based on our docker hub id use the below command

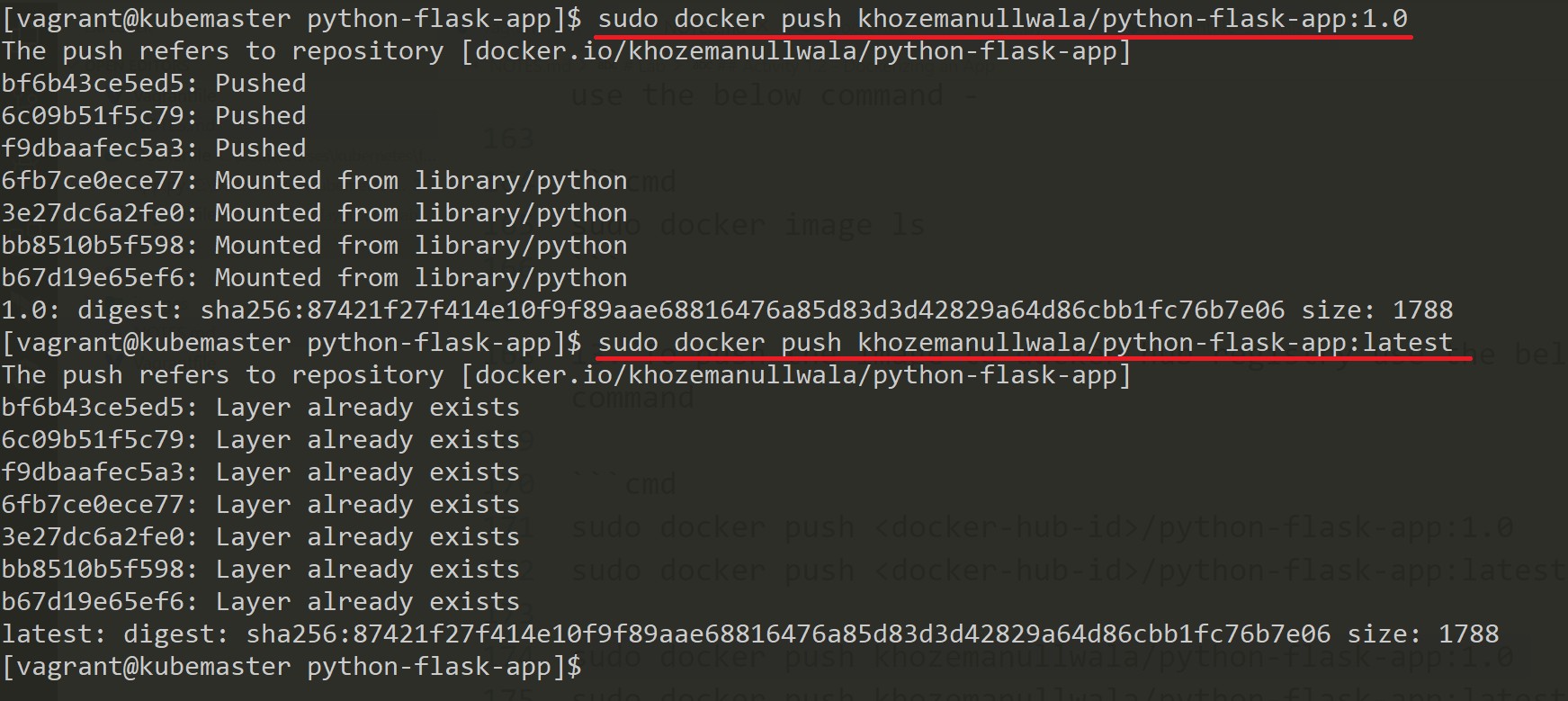
sudo docker image tag python-flask-app:1.0 <your-hub-id>/python-flask-app:1.0 sudo docker image tag python-flask-app:1.0 <your-hub-id>/python-flask-app:latest

1. To view the repository image and tag available locally use the below command -



sudo docker image ls

1. To push the image to docker hub registry use the below command



sudo docker push joshuadaniel/python-flask-app:1.0

sudo docker push docker123shweta/python-flask-app:latest

1. Once the image is pushed you can view the image in your docker hub registered account as shown below -
2. We can remove all the unused local image from docker host using the below command -

sudo docker image prune --all --force



1. Check that there are no images left related to python-flask-app

sudo docker image ls

2.1 **Setting up single node kunernetes cluster using Minikube**:

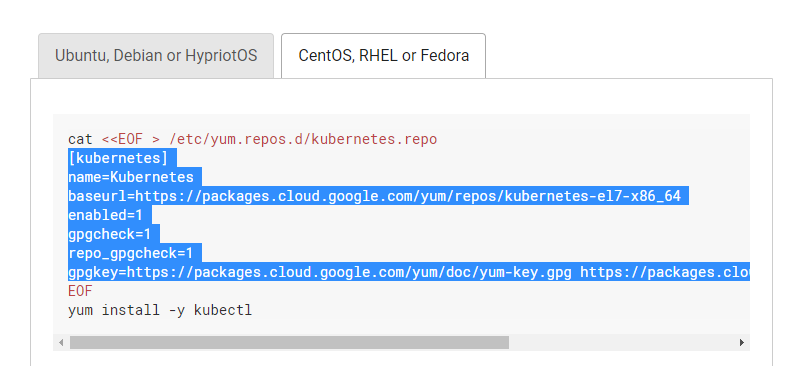
Setting up single node Kubernetes cluster using Minikube-|]

1] cd /etc/yum.repos.d

2] vi kubernetes.repo

3] Open this url and copy the content related to the type of Linux OS you have. For example, in our case it is CentOS. Paste that content in kubernetes.repo

<https://kubernetes.io/docs/tasks/tools/install-kubectl/>



[kubernetes]

name=Kubernetes

baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86\_64

enabled=1

gpgcheck=1

repo\_gpgcheck=1

gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg

4] yum install -y kubectl

5] kubectl (kubectl installation done)

6] cd /home/user1/Downloads

7] To download minikube into the downloads

curl -Lo minikube https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64

8] sudo chmod +x minikube

9] sudo mv minikube /usr/bin/

10] systemctl status firewalld

11] systemctl stop firewalld

12] systemctl disable firewalld

13] yum install conntrack -y

14] minikube start --vm-driver=none

15] minikube status

16] kubectl version

17]minikube version

18] kubectl get nodes

19] minikube dashboard

# Module 03 - Kubernetes Cluster Basic

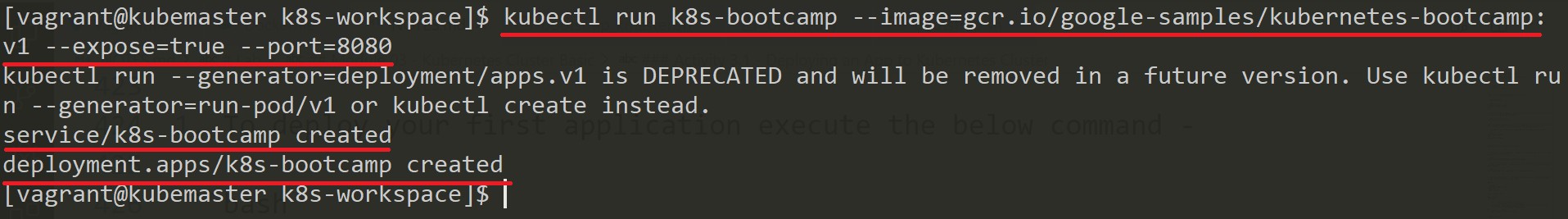
## Activity 3.1 - Deploying an App to Kubernetes Cluster

Deploying first app on Kubernetes cluster using kubectl

1. To deploy your first application, execute the below command -

kubectl run k8s-bootcamp --image=gcr.io/google-samples/kubernetes-bootcamp:v1 -- expose=true --port=8080

The above run command creates a new deployment with k8s-bootcamp name and exposes it through service on port 8080 because of --expose=true. The image location is specified through --image (include full url for images hosted outside docker hub)



1. To view all the deployments execute the below command -

kubectl get deployments OR

kubectl get deploy

1. To view all the pods execute the below command -

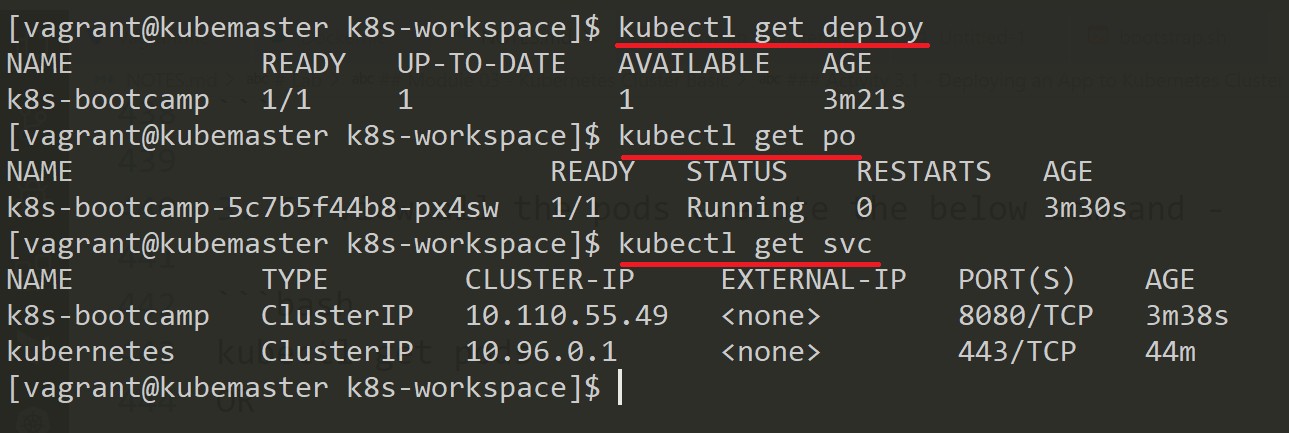
kubectl get pods OR

kubectl get po

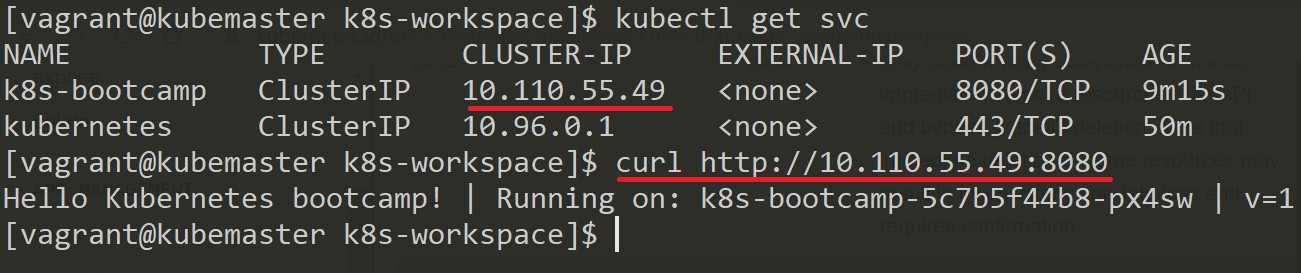
1. To view all the services execute the below command -

kubectl get services OR

kubectl get svc



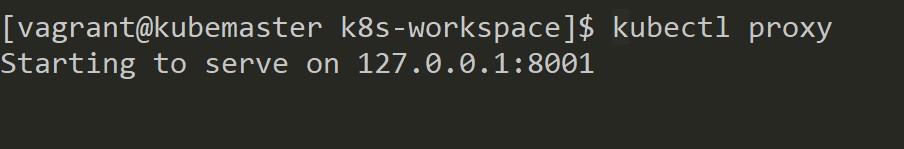
1. To view the content of the application that is exposed through service internally execute the below command (check the screen capture) -



curl http://<Cluster-IP>:8080

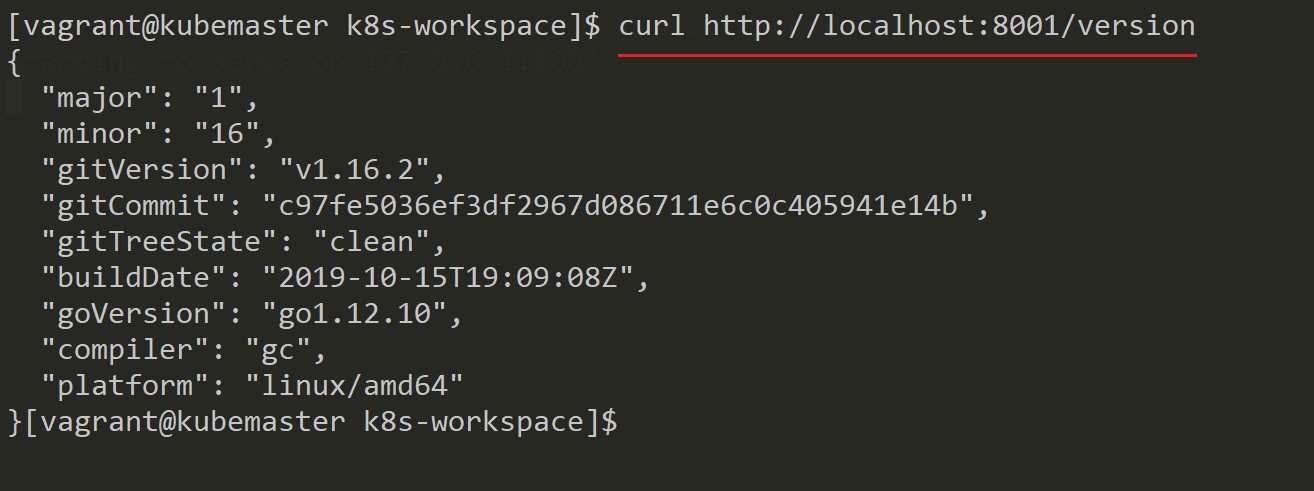
1. Open a new terminal window the type the below command to start a proxy that will forward all the communication into the cluster-wide private network. The proxy can be terminate by pressing the key combination Ctrl + C. It won't show any output while running.

kubectl proxy

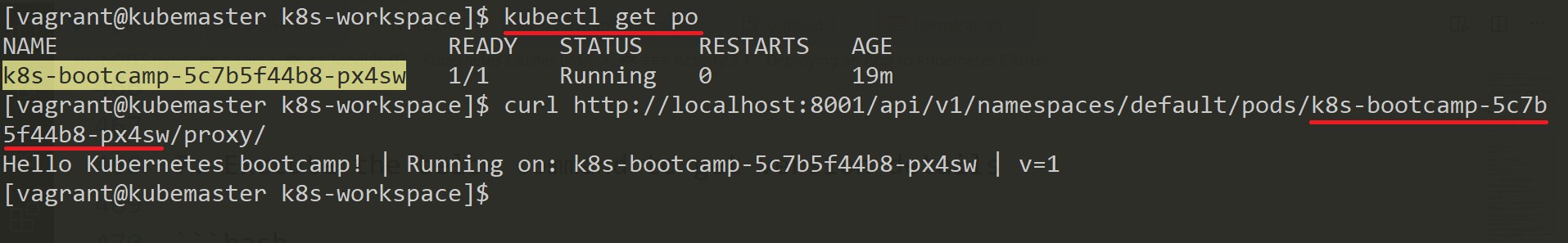


1. Execute the below command to get version details

curl http://localhost:8001/version

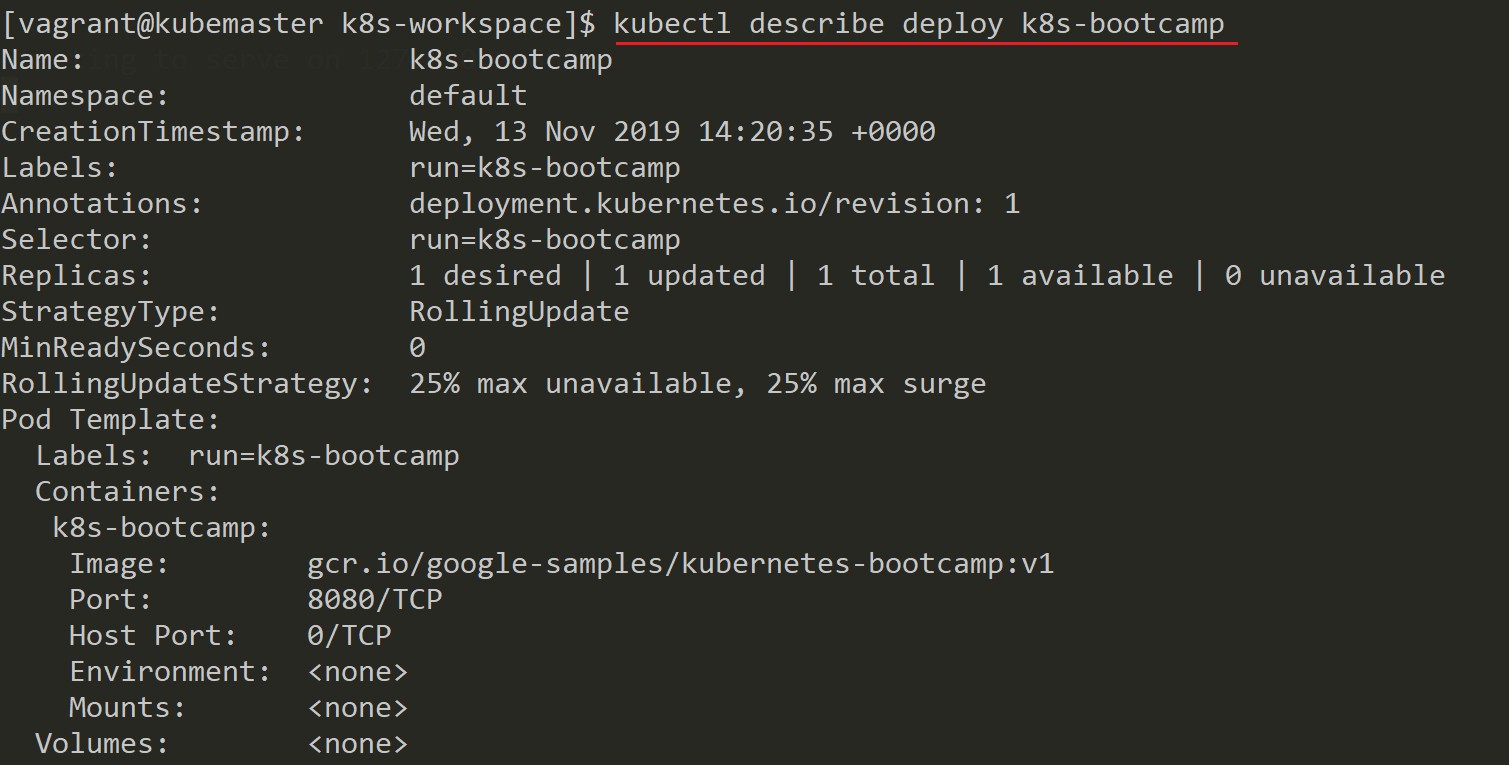


1. To access the app ui execute the below command (refer the screen capture) -



curl http://localhost:8001/api/v1/namespaces/default/pods/<your-pod-name>/proxy/

1. To get more details about the deployment execute the below command -



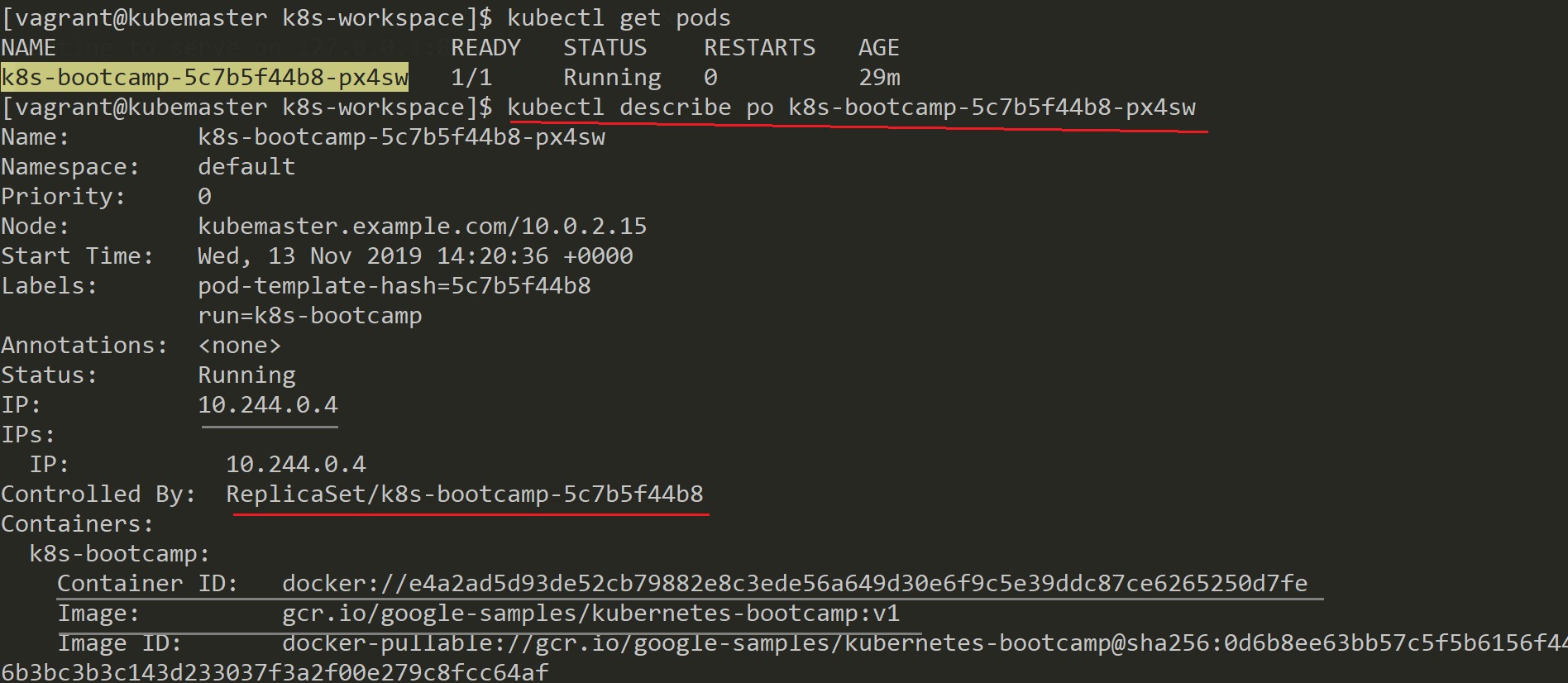
kubectl describe deployment k8s-bootcamp OR

kubectl describe deploy k8s-bootcamp

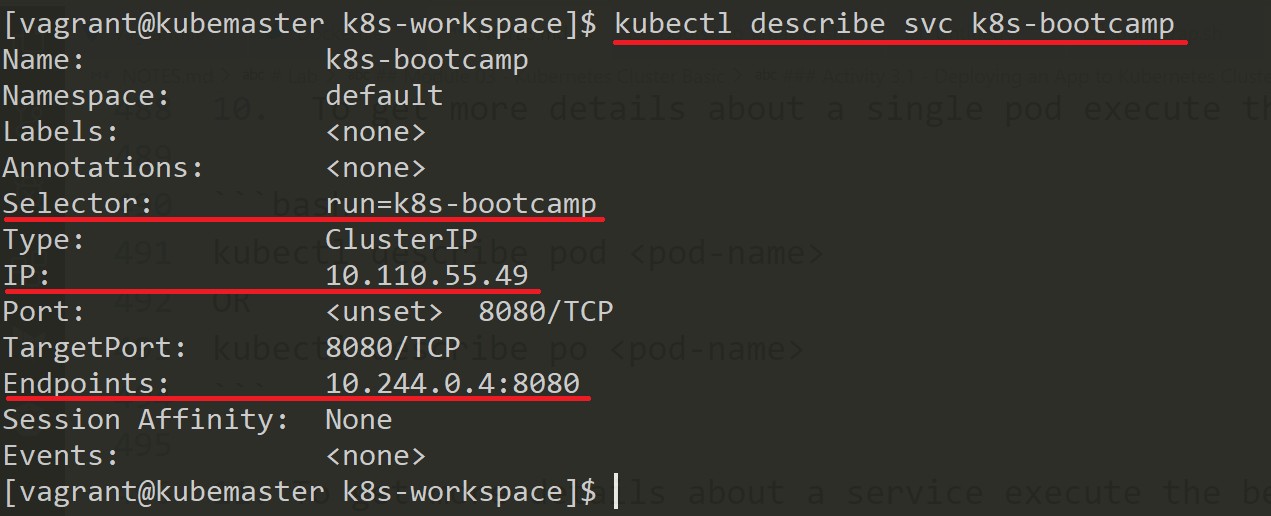
1. To get more details about a single pod execute the below command -

kubectl describe pod <pod-name> OR

kubectl describe po <pod-name>



1. To get more details about a service execute the below command -



kubectl describe service k8s-bootcamp OR

kubectl describe svc k8s-bootcamp

1. To delete the deployment created earlier execute the below command -

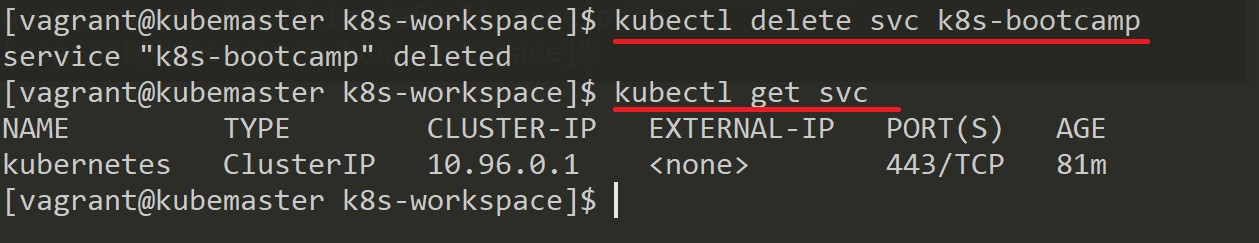
kubectl delete deployment k8s-bootcamp OR

kubectl delete deploy k8s-bootcamp

**NOTE:** Deletion of deployment delete the pods associated with a deployment but it does not delete the service as can be seen from screen capture.



1. To delete the service created earlier execute the below command -



kubectl delete service k8s-bootcamp OR

kubectl delete svc k8s-bootcamp

## Activity 3.2 - Launcing a Pod in Kubernetes Cluster using yaml and troubleshooting it

1. Create a new k8s-bootcamp-pod.yml file using the below command -

vim k8s-bootcamp-pod.yml

1. Add the below content to k8s-bootcamp-pod.yml file -

**---**

apiVersion: v1 kind: Pod metadata:

name: k8s-bootcamp-pod labels:

app: k8s-bootcamp spec:

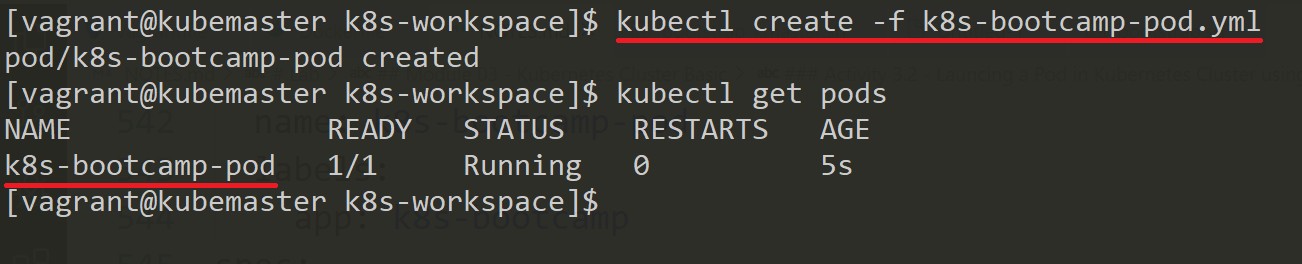
containers:

- name: k8s-bootcamp-container

image: gcr.io/google-samples/kubernetes-bootcamp:v1 ports:

- containerPort: 8080

1. To create a Pod from k8s-bootcamp-pod.yml file execute the below command -

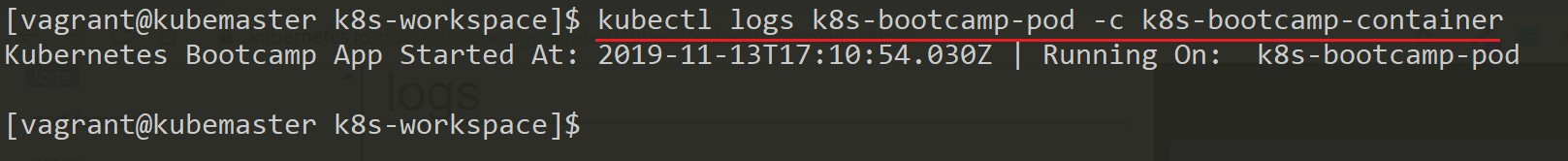


kubectl create -f k8s-bootcamp-pod.yml

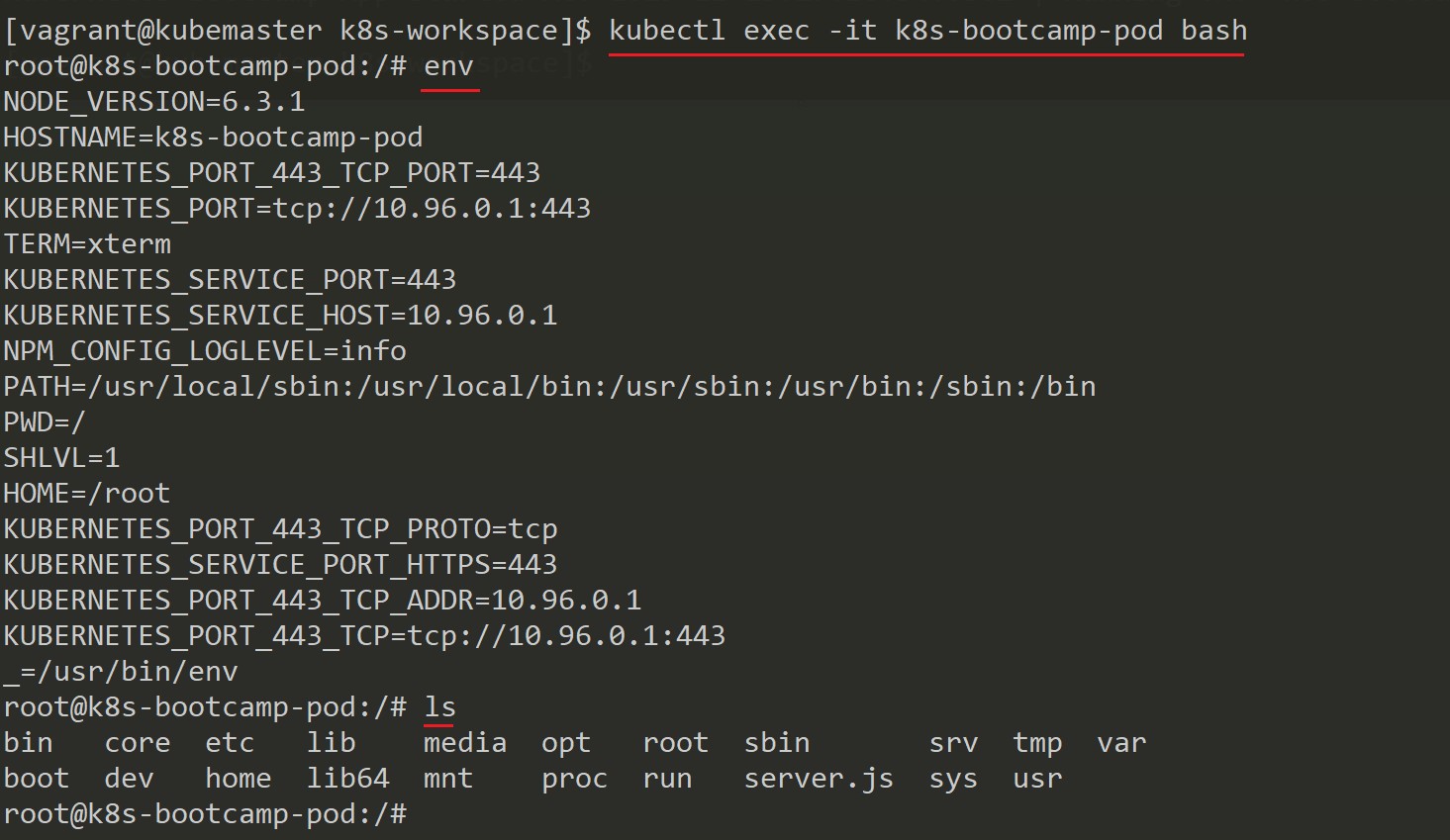
1. To print the logs for a container in a pod or specified resource. If the pod has only one container, the container name is optional else specify the container name using the -c option.

kubectl logs k8s-bootcamp-pod OR

kubectl logs k8s-bootcamp-pod -c k8s-bootcamp-container



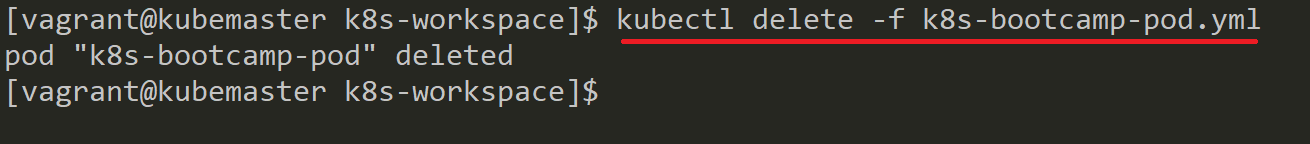
1. To execute command directly inside the container use the below command -



kubectl exec -it k8s-bootcamp-pod bash

**NOTE:** Container name. If omitted using -c option, the first container in the pod will be chosen.

1. Type exit to come out the container.
2. To delete the resource created using file can be deleted by executing the below command -



kubectl delete -f k8s-bootcamp-pod.yml

## Activity 3.3 - Deploying App and Exposing it using Services

1. Create a new k8s-bootcamp-deployment.yml file using the below command -

vim k8s-bootcamp-deployment.yml

1. Add the below content to k8s-bootcamp-deployment.yml file -

**---**

apiVersion: apps/v1 *# Note the changes from Pod file*

kind: Deployment *# Note the change*

metadata:

name: k8s-bootcamp-deploy labels:

app: k8s-bootcamp-app spec:

replicas: 1 *# number of replicas*

selector: *# selectors*

matchLabels:

app: k8s-bootcamp *# this should match the Pod label in spec*

template: *# template*

metadata:

name: k8s-bootcamp-pod labels:

app: k8s-bootcamp *# Pod label*

spec:

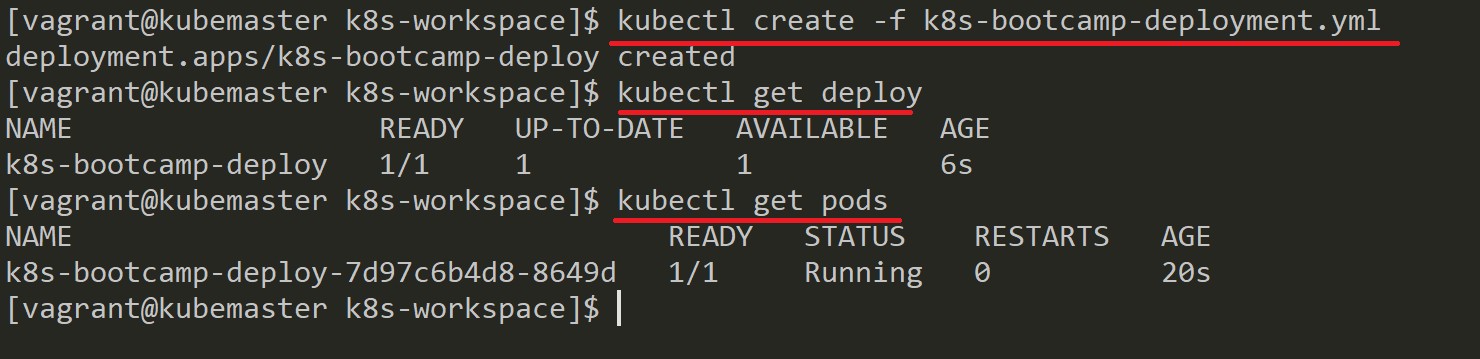
containers:

- name: k8s-bootcamp-container

image: gcr.io/google-samples/kubernetes-bootcamp:v1 ports:

- containerPort: 8080

1. To create the deployment execute using the below command -



kubectl create -f k8s-bootcamp-deployment.yml

1. To expose the deployment through a service execute the below command OR check step 5 -

kubectl expose deployment/k8s-bootcamp-deploy --type=NodePort --port=8080

1. You can create a new k8s-bootcamp-service.yml add the below content to it -

apiVersion: v1 kind: Service metadata:

name: k8s-bootcamp-svc labels:

app: k8s-bootcamp-app spec:

type: NodePort *# Default is ClusterIP if not specified*

selector:

app: k8s-bootcamp *# match the Pod label*

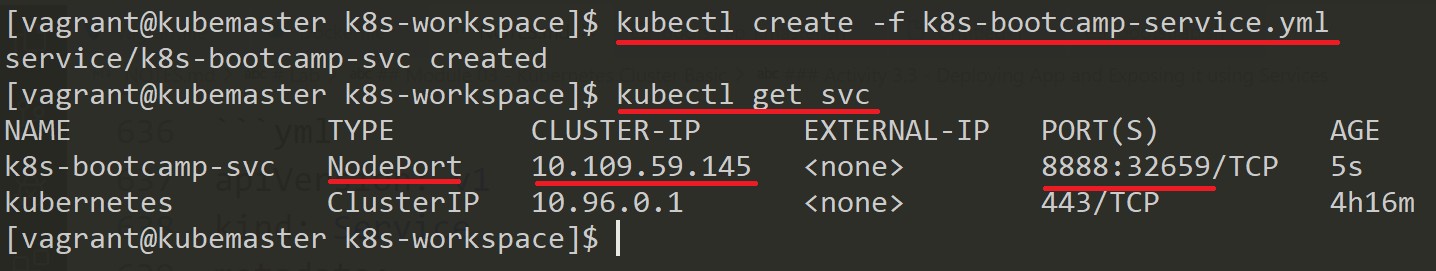
ports:

- port: 8888

targetPort: 8080 *# targetPort is optional if same as port number*

protocol: TCP

1. To create the service execute the below command -



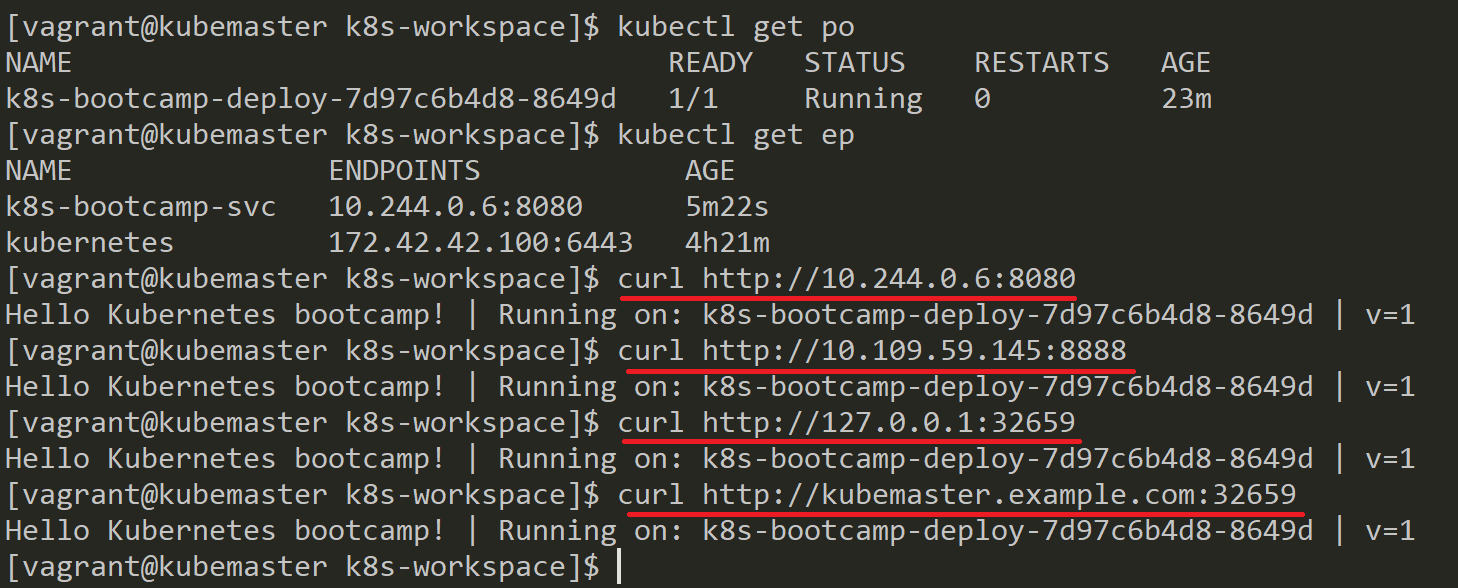
kubectl create -f k8s-bootcamp-service.yml

1. Try the below different options to access the App UI

curl http://<node-ip>:<container-port>

curl http://<service-ip>:<service-port>

[curl http://localhost:<node-port>](http://localhost/)

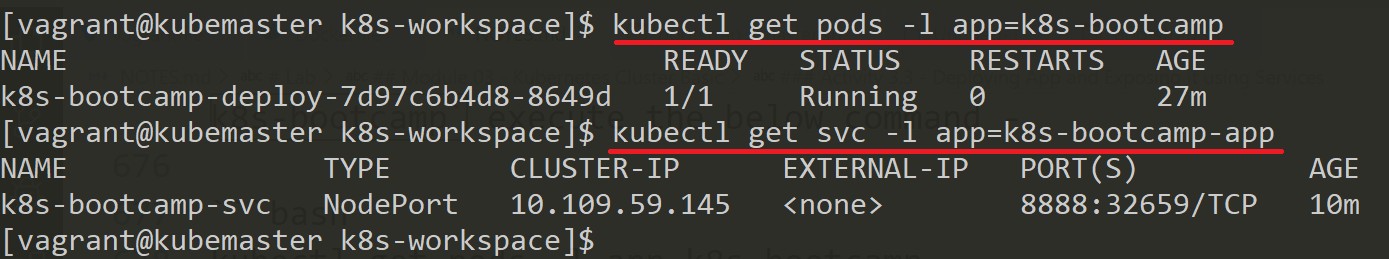


1. With deployments it would become difficult to find individual Pod hence labels are used to identify any Pod easily. To find a Pod labeled as app: k8s-bootcamp execute the below command -

kubectl get pods -l app=k8s-bootcamp OR

kubectl get po -l app=k8s-bootcamp

1. Similarly we can easily find a service. To find a Service labeled as app: k8s-bootcamp-app execute the below command -



kubectl get service -l app=k8s-bootcamp-app OR

kubectl get svc -l app=k8s-bootcamp-app

1. To apply a new label use the label command followed by object type, object name and new label to apply.

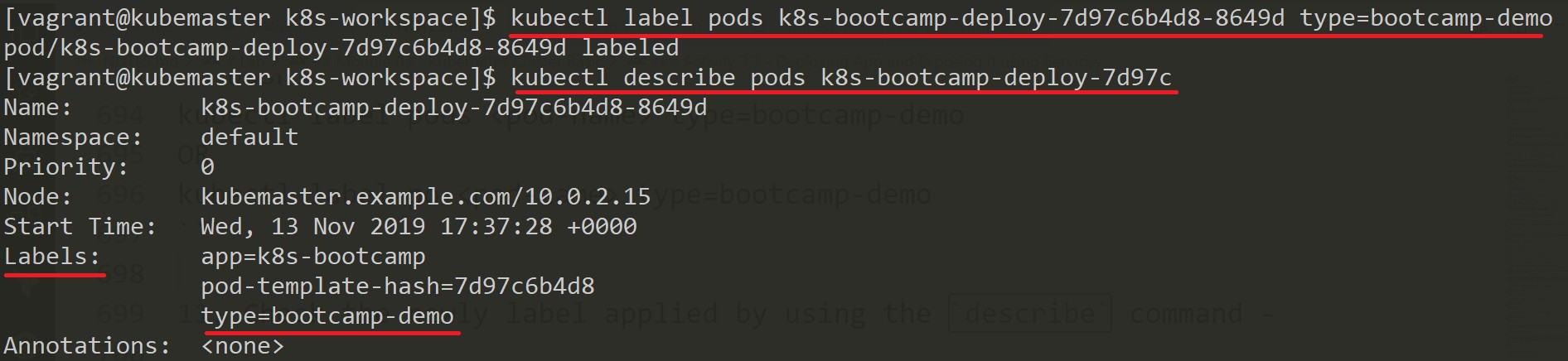
kubectl label pods <pod-name> type=bootcamp-demo OR

kubectl label po <pod-name> type=bootcamp-demo

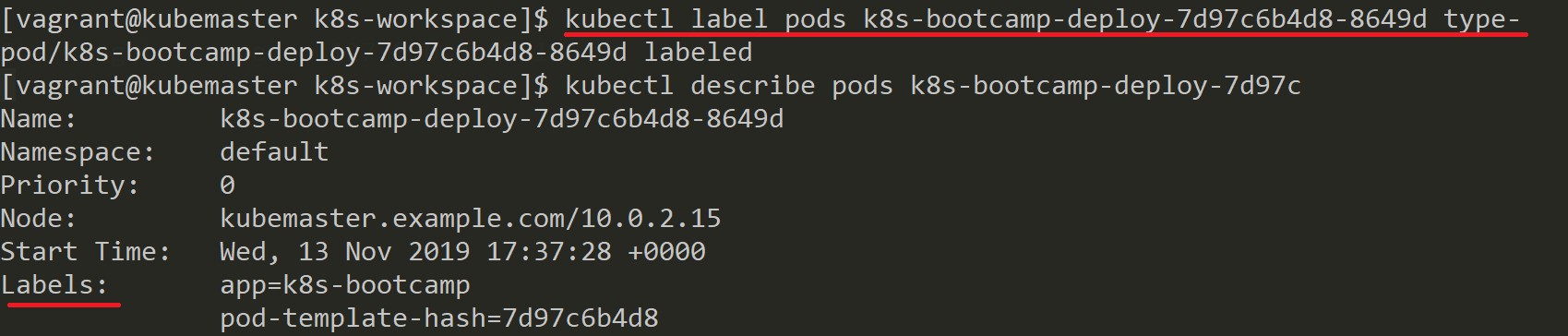
1. Check the newly label applied by using the describe command -

kubectl describe pods <pod-name> OR

kubectl describe po <pod-name>



1. To remove the label added previously use the below command at the end of the label name specify a - (minus) symbol -



kubectl label pods <pod-name> type- OR

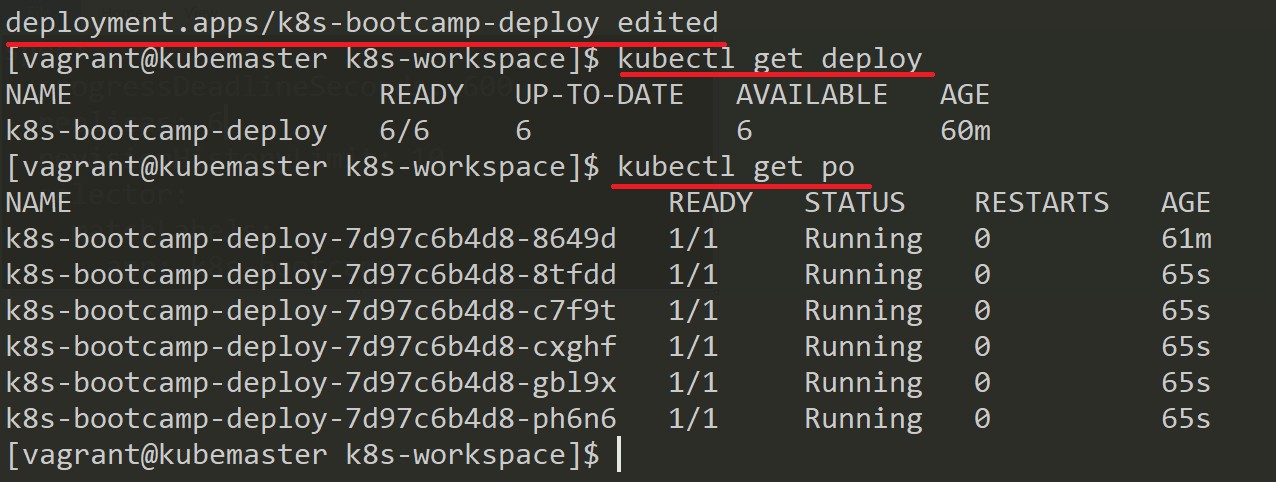
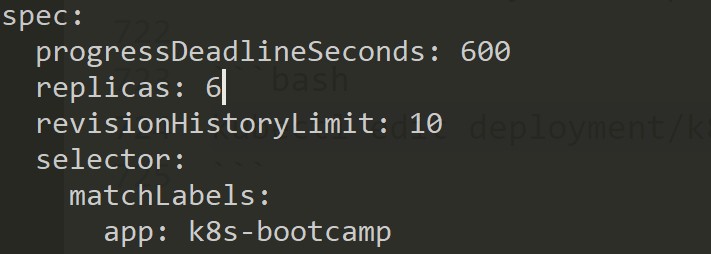
kubectl label po <pod-name> type-

## Activity 3.4 - Scaling a Deployed App

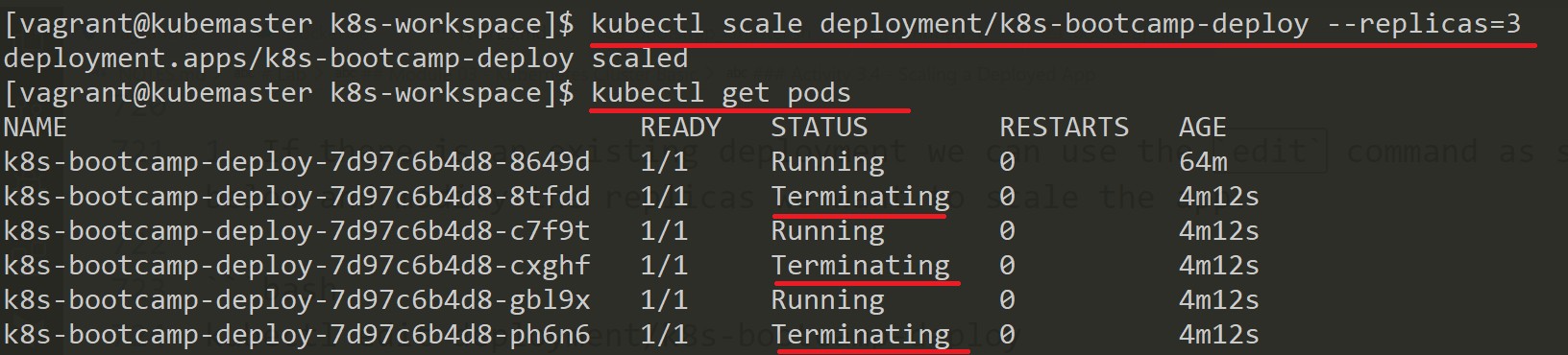
1. If there is an existing deployment we can use the edit command as shown below and modify the replicas we want to scale the app.

kubectl edit deployment/k8s-bootcamp-deploy

**NOTE:** Notice the number of Pods has changed to 6 based on replicas desired as shown in screen capture.

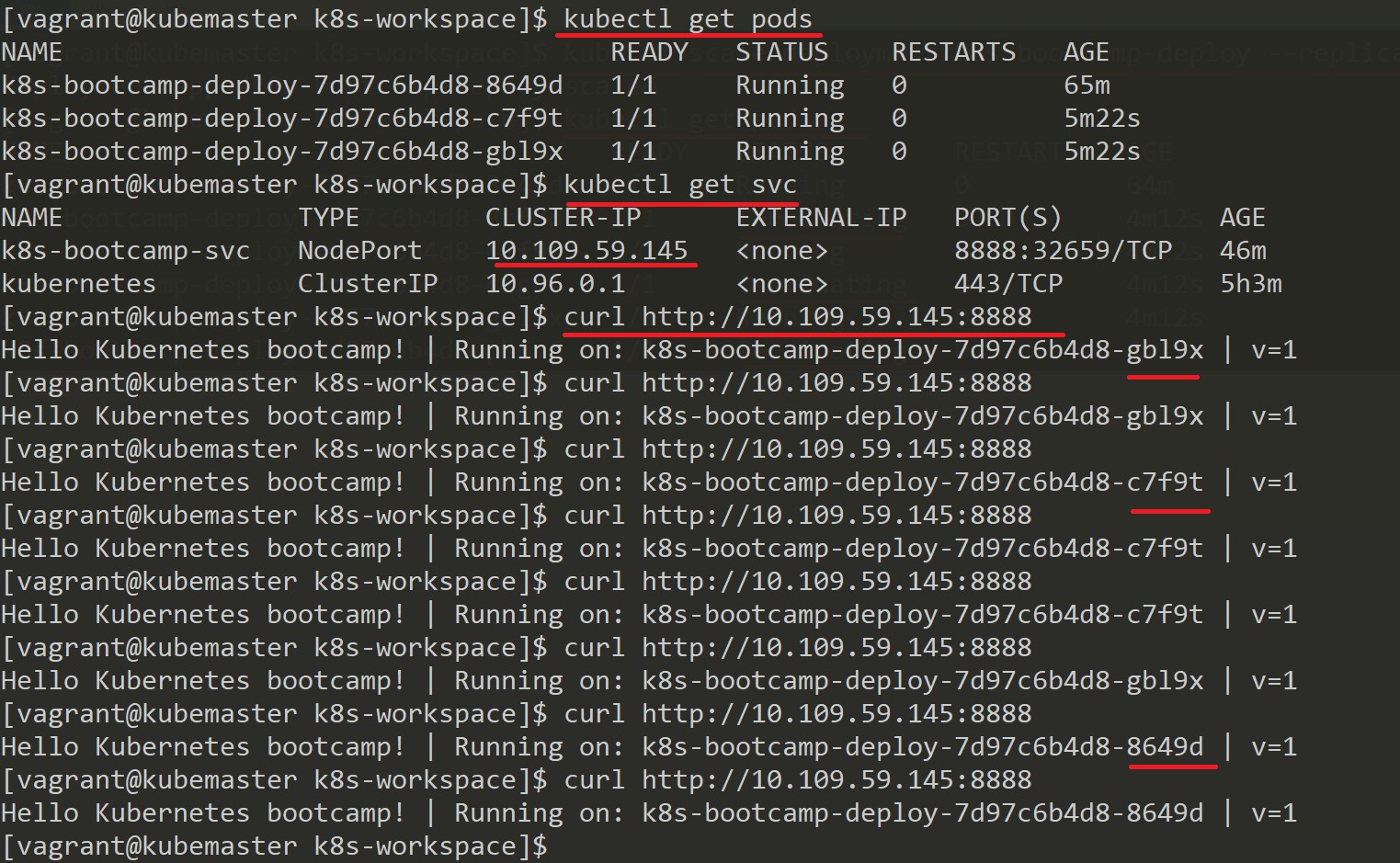


1. We can also use the scale command to modify the existing replicas. Execute the below command to scale the down the replicas to 3 instances.



kubectl scale deployment/k8s-bootcamp-deploy --replicas=3

1. Once the desired instances are up use the service ip and port number to access the app ui. Notice the load balancer working and giving output from different instances by balancing the load.



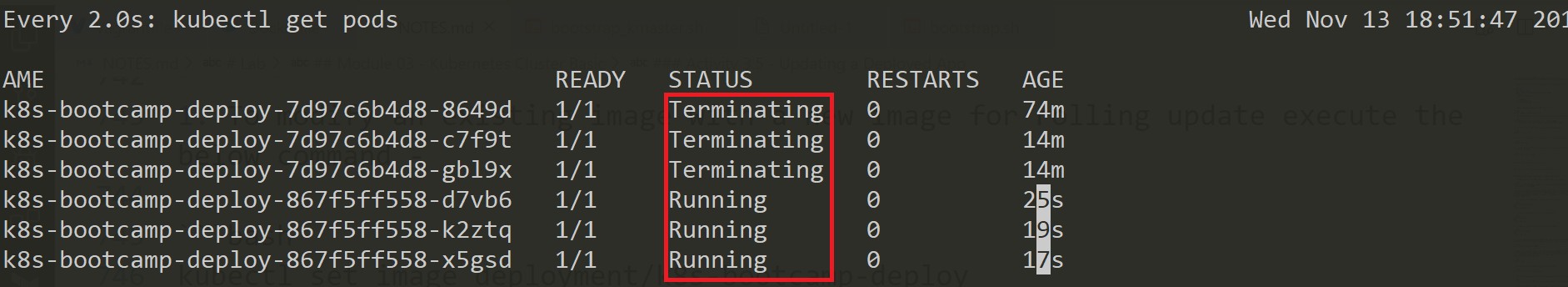
## Activity 3.5 - Updating a Deployed App

1. To modify an existing image of a deployment with a new image and initiate a rolling update execute the below command -



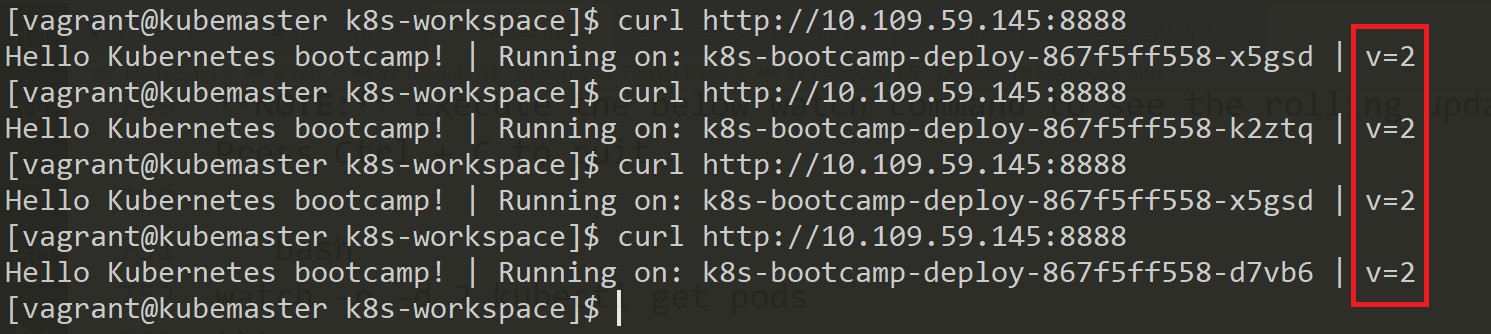
kubectl set image deployment/k8s-bootcamp-deploy k8s-bootcamp- container=jocatalin/kubernetes-bootcamp:v2

**NOTE:** Execute the below watch command to see the rolling update happening. Press Ctrl + C to quit.

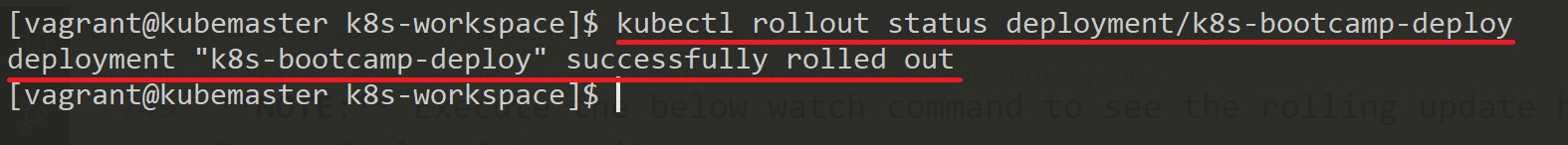


watch -d -n 2 kubectl get pods

1. Using the service access the app UI again to see the new image being used.



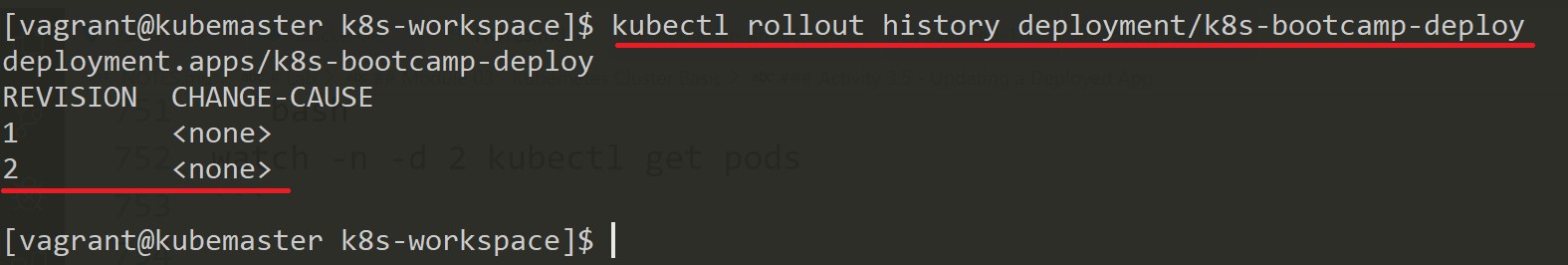
1. To check the rollout status of deployment execute the below command -



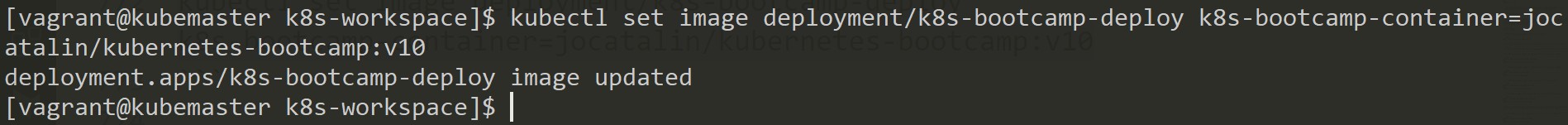
kubectl rollout status deployment/k8s-bootcamp-deploy

1. To check the rollout history of deployment execute the below command -

kubectl rollout history deployment/k8s-bootcamp-deploy

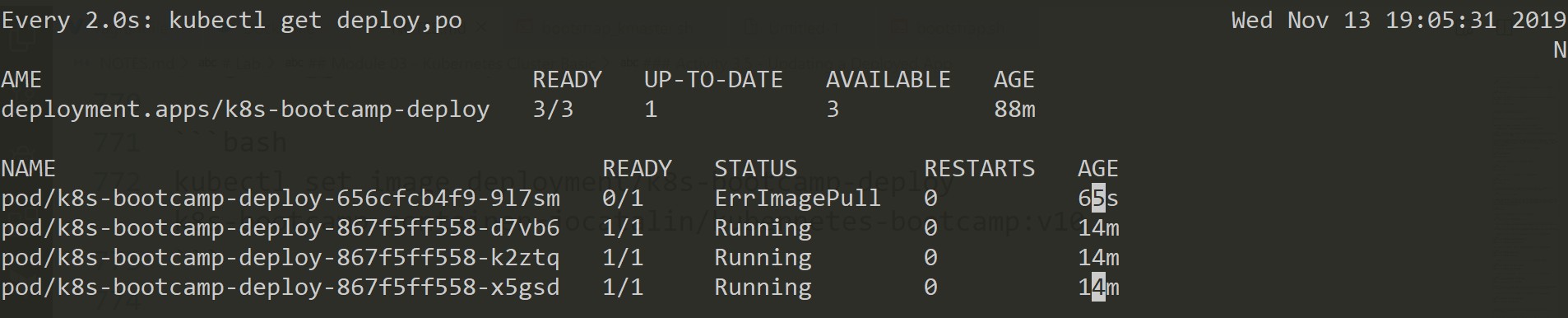


1. This time use an image tagged as v10 (this is intentional as there is no image tagged as v10)



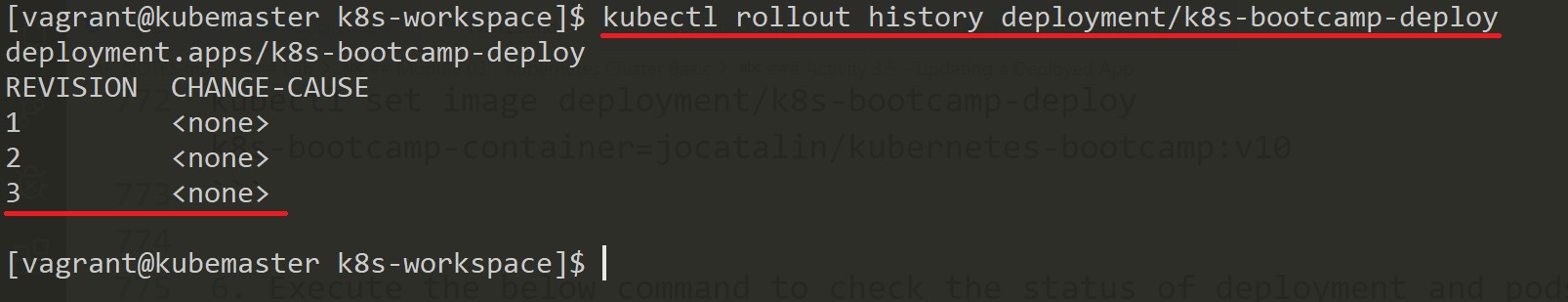
kubectl set image deployment/k8s-bootcamp-deploy k8s-bootcamp- container=jocatalin/kubernetes-bootcamp:v10

1. Execute the below command to check the status of deployment and pods. Press Ctrl + C to quit



watch -d -n kubectl get deploy,po

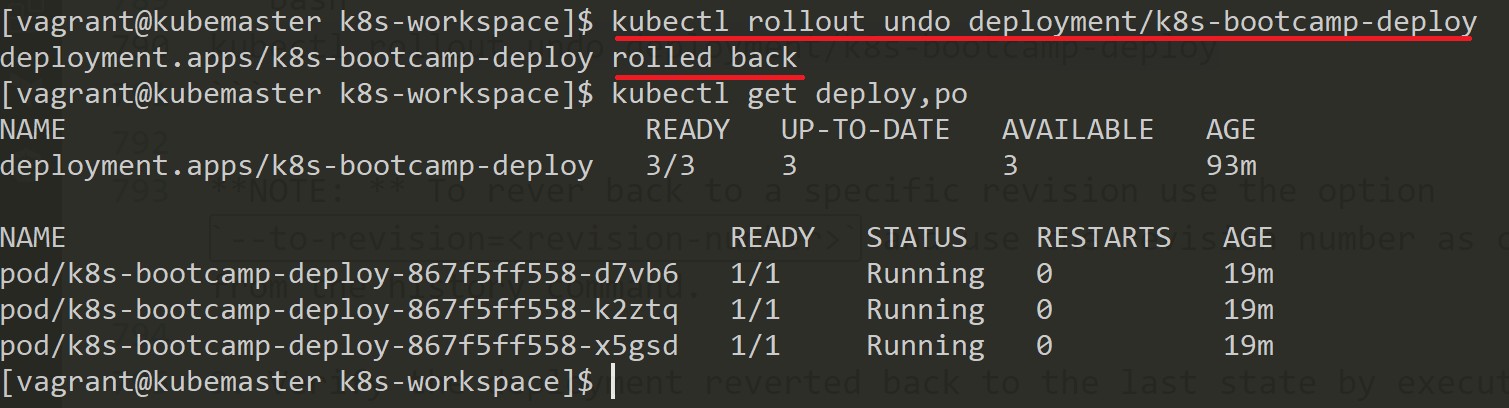
1. To check the rollout history again execute the below command -



kubectl rollout history deployment/k8s-bootcamp-deploy

1. To revert back the deployment to the previous know state execute the below command -

kubectl rollout undo deployment/k8s-bootcamp-deploy



**NOTE:** To rever back to a specific revision use the option --to-revision=<revision-number> and use the revision number as can be seen from the history command.

1. Verify the deployment reverted back to the last state by executing the command -

kubectl get deploy, pod

1. To delete the objects create using the file execute the below command -

kubectl delete -f k8s-bootcamp-deployment.yml -f k8s-bootcamp-service.yml



1. Both the files can be combined into one single file say k8s-bootcamp-app.yml file and content of the file -

**---**

apiVersion: apps/v1 *# Note the changes from Pod file*

kind: Deployment *# Note the change*

metadata:

name: k8s-bootcamp-deploy labels:

app: k8s-bootcamp-app spec:

replicas: 1 *# number of replicas*

selector: *# selector*

matchLabels:

app: k8s-bootcamp *# this should match the Pod label in spec*

template: *# template*

metadata:

name: k8s-bootcamp-pod labels:

app: k8s-bootcamp *# Pod label*

spec:

containers:

- name: k8s-bootcamp-container

image: gcr.io/google-samples/kubernetes-bootcamp:v1 ports:

- containerPort: 8080

**---**

apiVersion: v1 kind: Service metadata:

name: k8s-bootcamp-svc labels:

app: k8s-bootcamp-app spec:

type: NodePort *# Default is ClusterIP if not specified*

selector:

app: k8s-bootcamp *# match the Pod label*

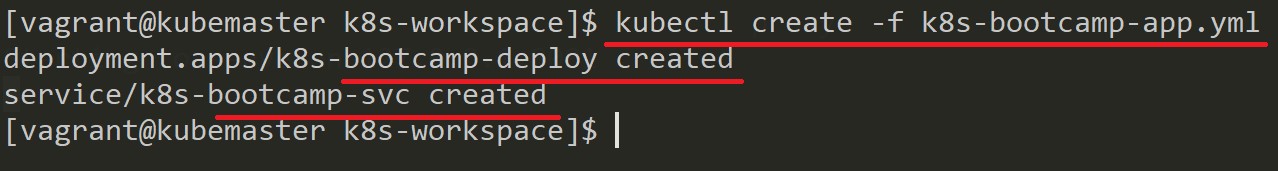
ports:

- port: 8888

targetPort: 8080 *# targetPort is optional if same as port number*

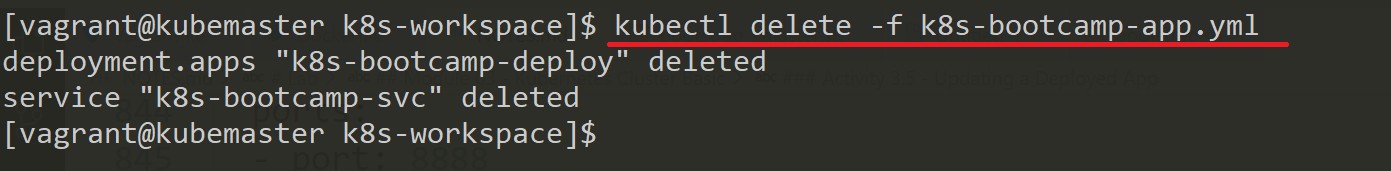
protocol: TCP

1. To create both the resource execute the command -



kubectl create -f k8s-bootcamp-app.yml

1. To delete both the resource execute the command -



kubectl delete -f k8s-bootcamp-app.yml