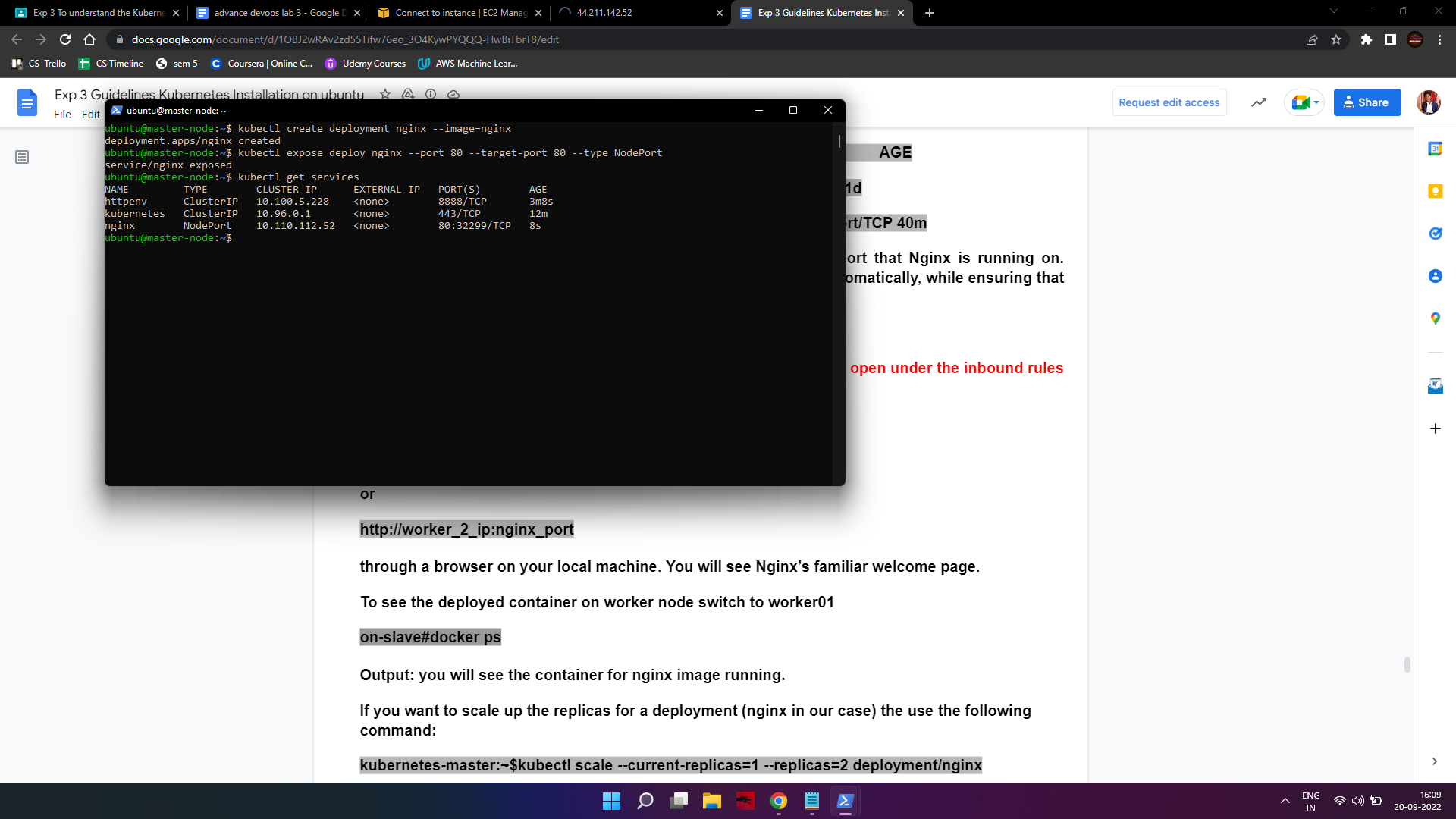
You can now deploy any containerized application to your cluster. To keep things familiar, let’s deploy Nginx using *Deployments* and *Services* to see how this application can be deployed to the cluster. You can use the commands below for other containerized applications as well, provided you change the Docker image name and any relevant flags (such as ports and volumes).

Still within the master node, execute the following command to create a deployment named nginx:

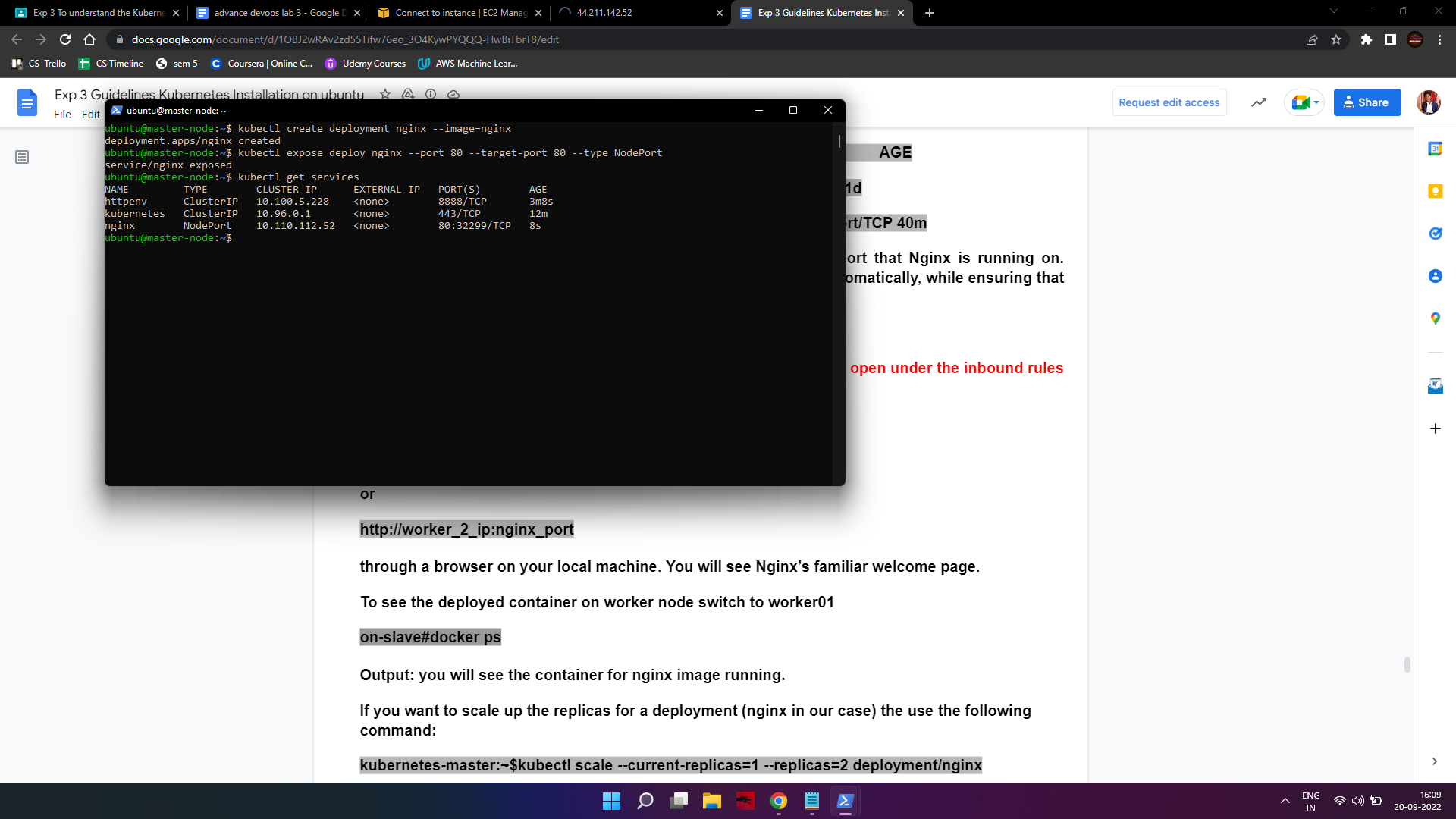
kubernetes-master:~$kubectl create deployment nginx --image=nginx



A deployment is a type of Kubernetes object that ensures there’s always a specified number of pods running based on a defined template, even if the pod crashes during the cluster’s lifetime. The above deployment will create a pod with one container from the Docker registry’s [Nginx Docker](https://hub.docker.com/_/nginx/) [Image](https://hub.docker.com/_/nginx/).

Next, run the following command to create a service named nginx that will expose the app publicly. It will do so through a *NodePort*, a scheme that will make the pod accessible through an arbitrary port opened on each node of the cluster:

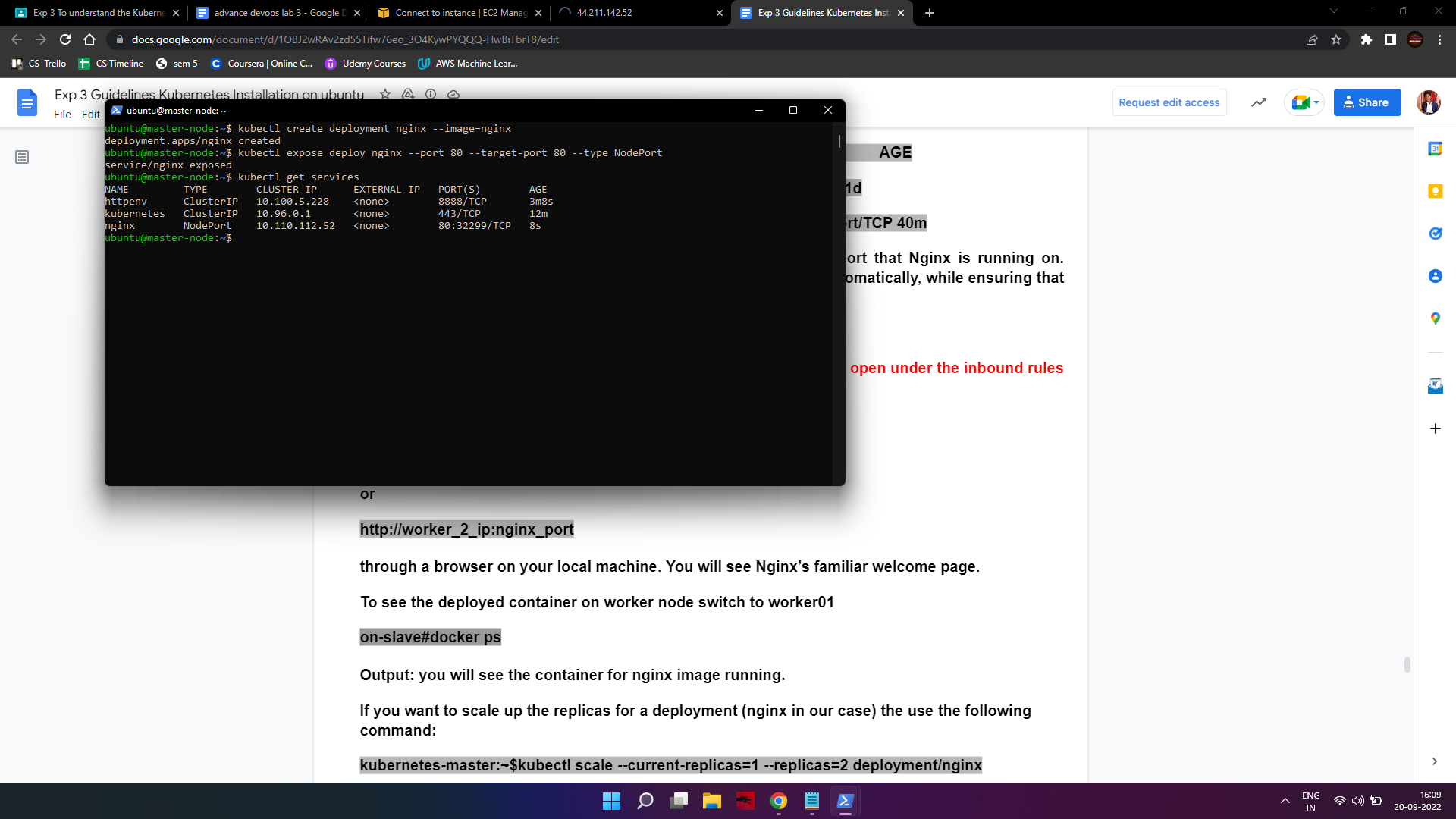
kubernetes-master:~$kubectl expose deploy nginx --port 80 --target-port 80 --type NodePort



Services are another type of Kubernetes object that expose cluster internal services to clients, both internal and external. They are also capable of load balancing requests to multiple pods, and are an integral component in Kubernetes, frequently interacting with other components.

Run the following command:

kubernetes-master:~$kubctl get services



This will output text similar to the following:

Output

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 1d

nginx NodePort 10.109.228.209 <none> 80:nginx\_port/TCP 40m

From the third line of the above output, you can retrieve the port that Nginx is running on. Kubernetes will assign a random port that is greater than 30000 automatically, while ensuring that the port is not already bound by another service.

Note: if you're running your setup on ec2 ensure the nginx\_port is open under the inbound rules in the security groups.

To test that everything is working, visit

http://worker\_1\_ip:nginx\_port or

http://worker\_2\_ip:nginx\_port

through a browser on your local machine. You will see Nginx’s familiar welcome page. To see the deployed container on the worker node switch to worker01

on-slave#docker ps

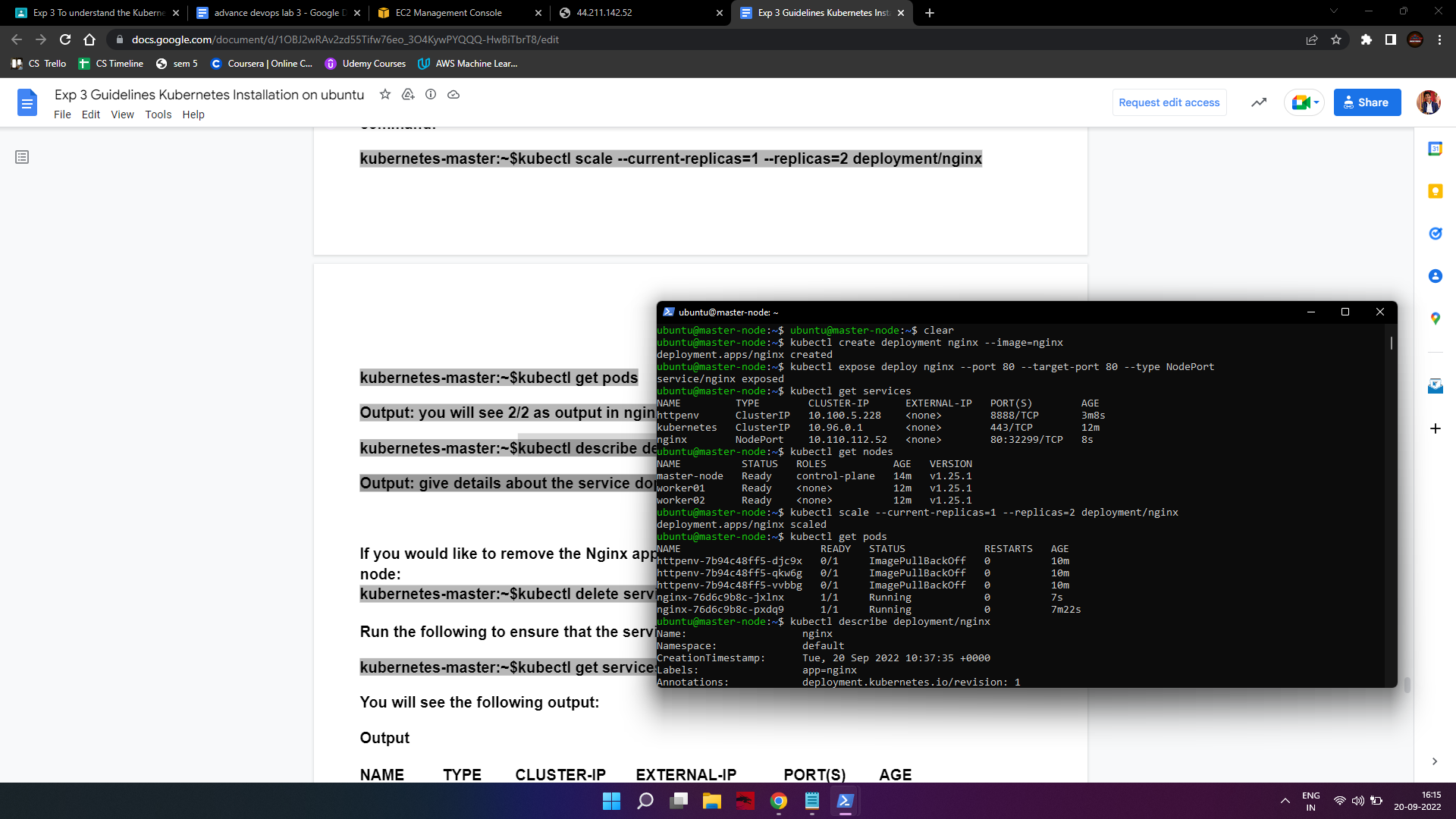
Output: you will see the container for nginx image running.

If you want to scale up the replicas for a deployment (nginx in our case) the use the following command:

kubernetes-master:~$kubectl scale --current-replicas=1 --replicas=2 deployment/nginx

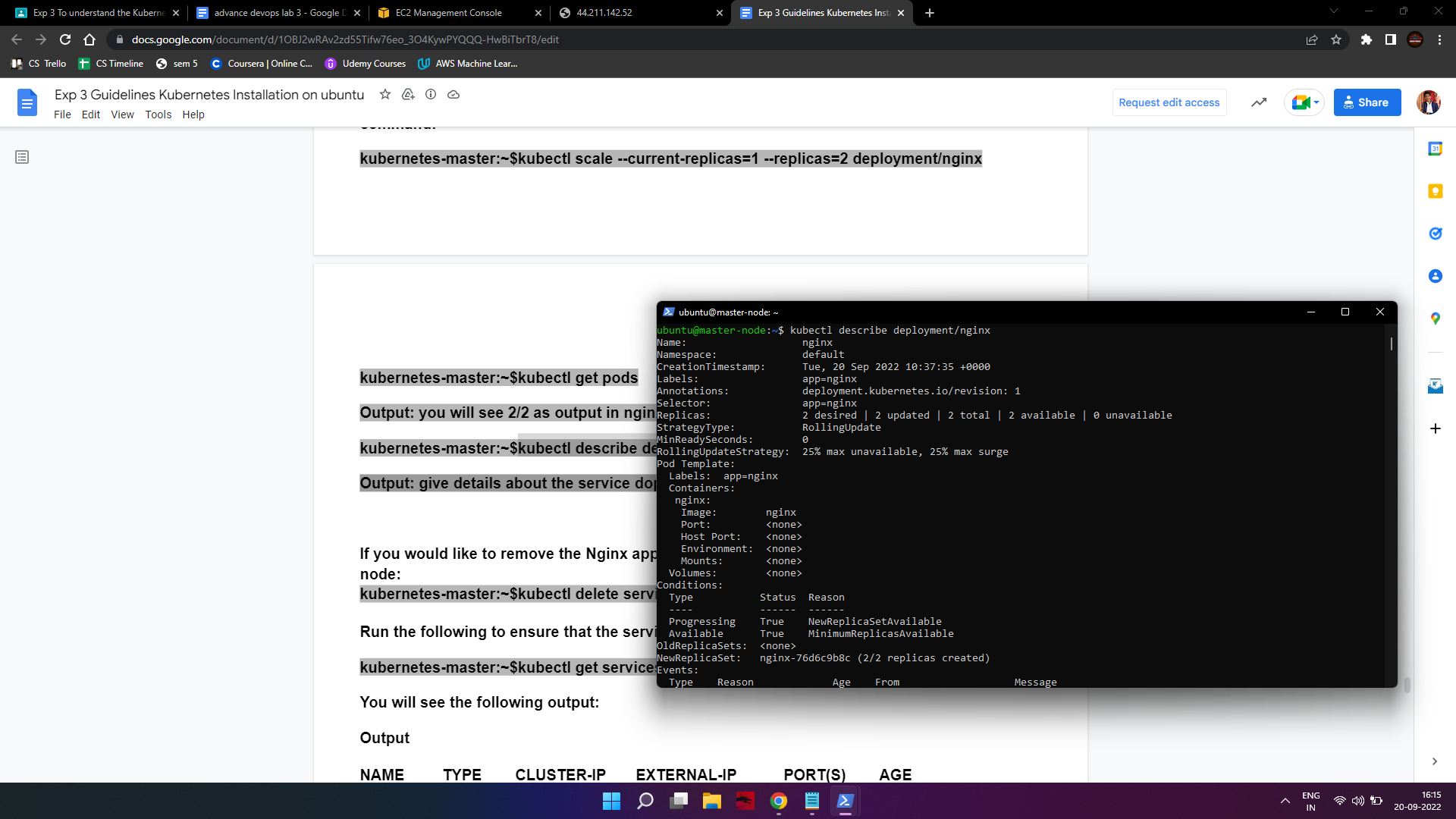
kubernetes-master:~$kubectl get pods

Output: you will see 2/2 as output in nginx deployment.



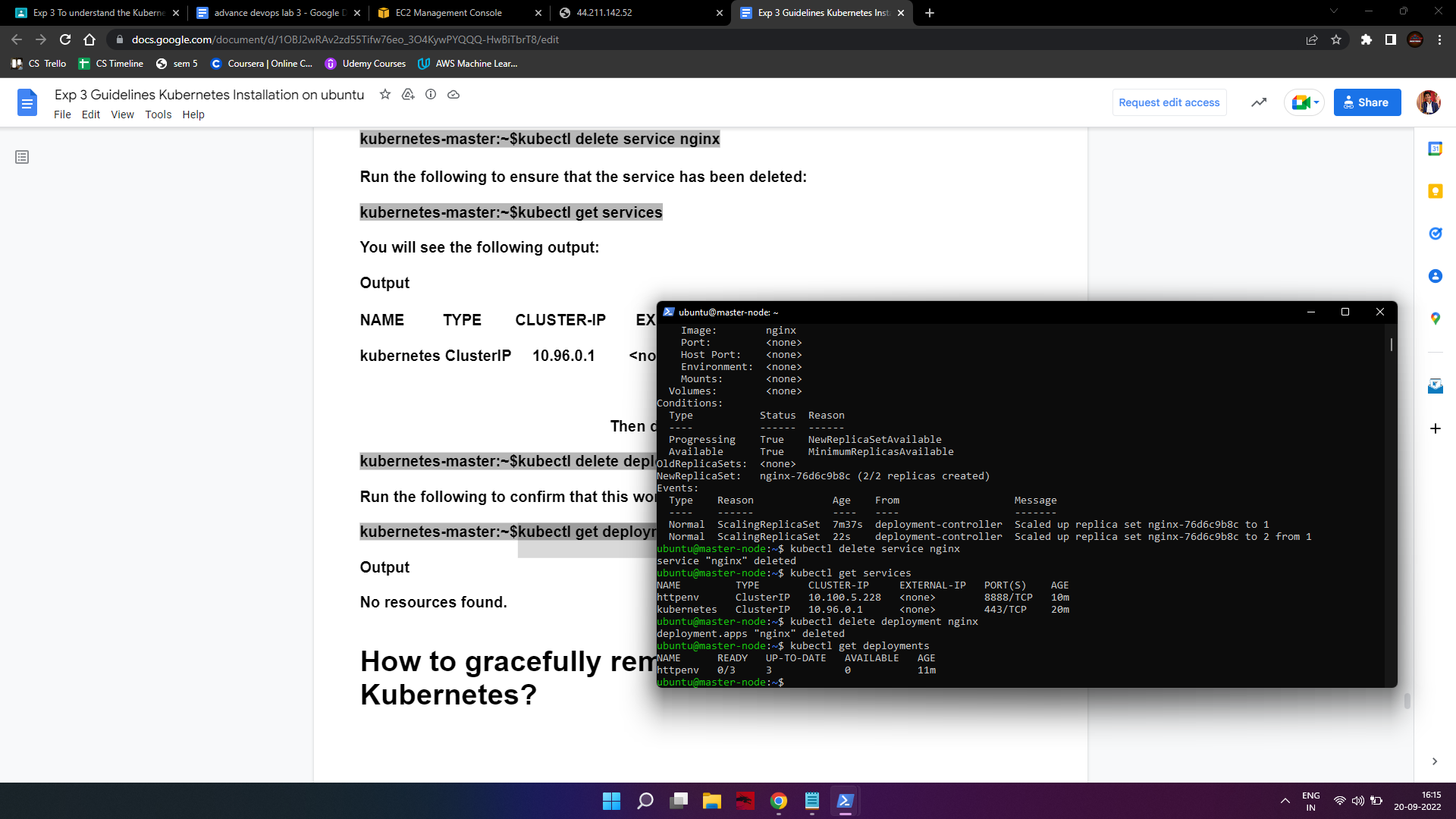
kubernetes-master:~$kubectl describe deployment/nginx

Output: give details about the service deployed



If you would like to remove the Nginx application, first delete the nginx service from the master node:

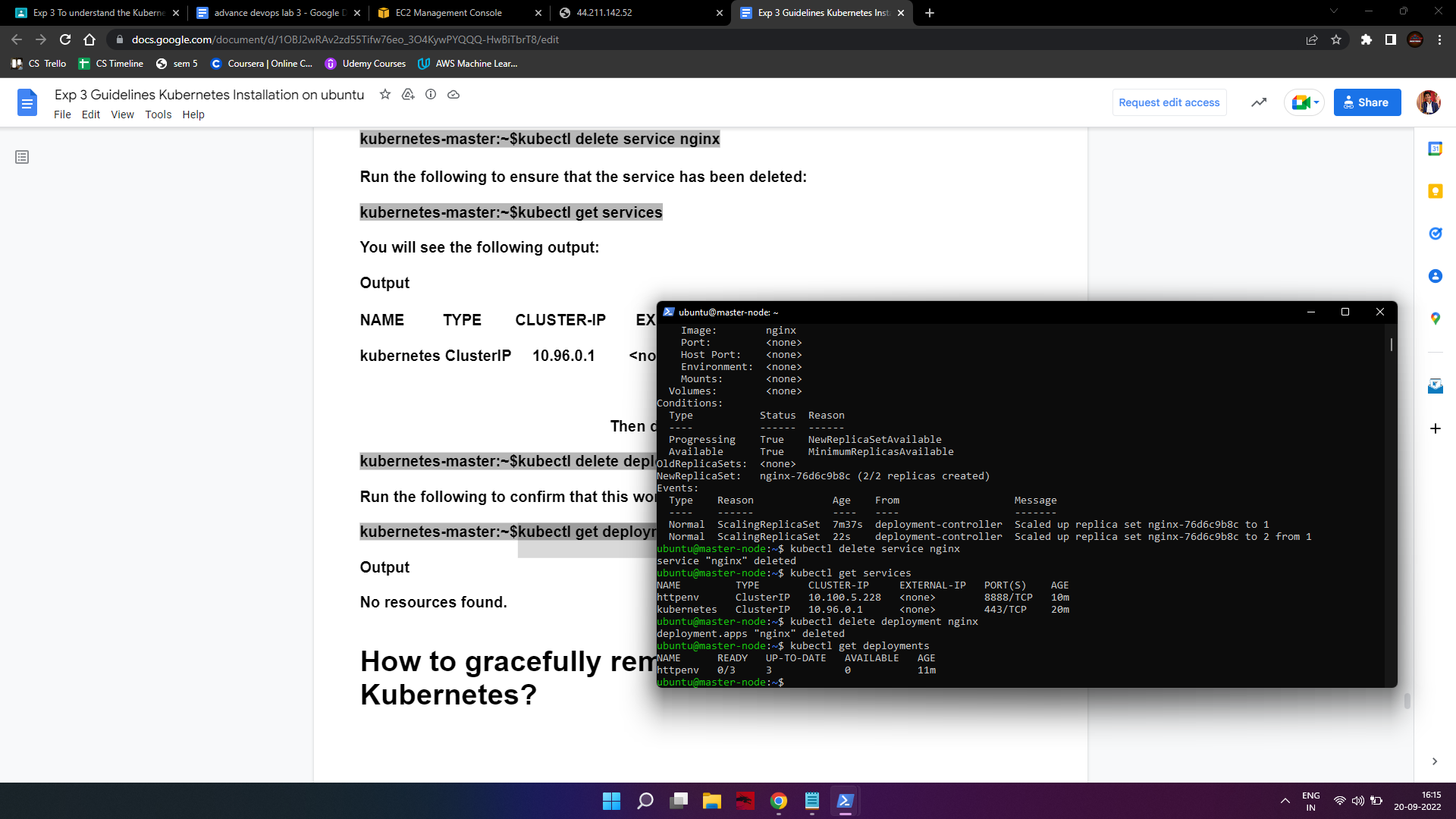
kubernetes-master:~$kubectl delete service nginx



Run the following to ensure that the service has been deleted:

kubernetes-master:~$kubectl get services

You will see the following output:



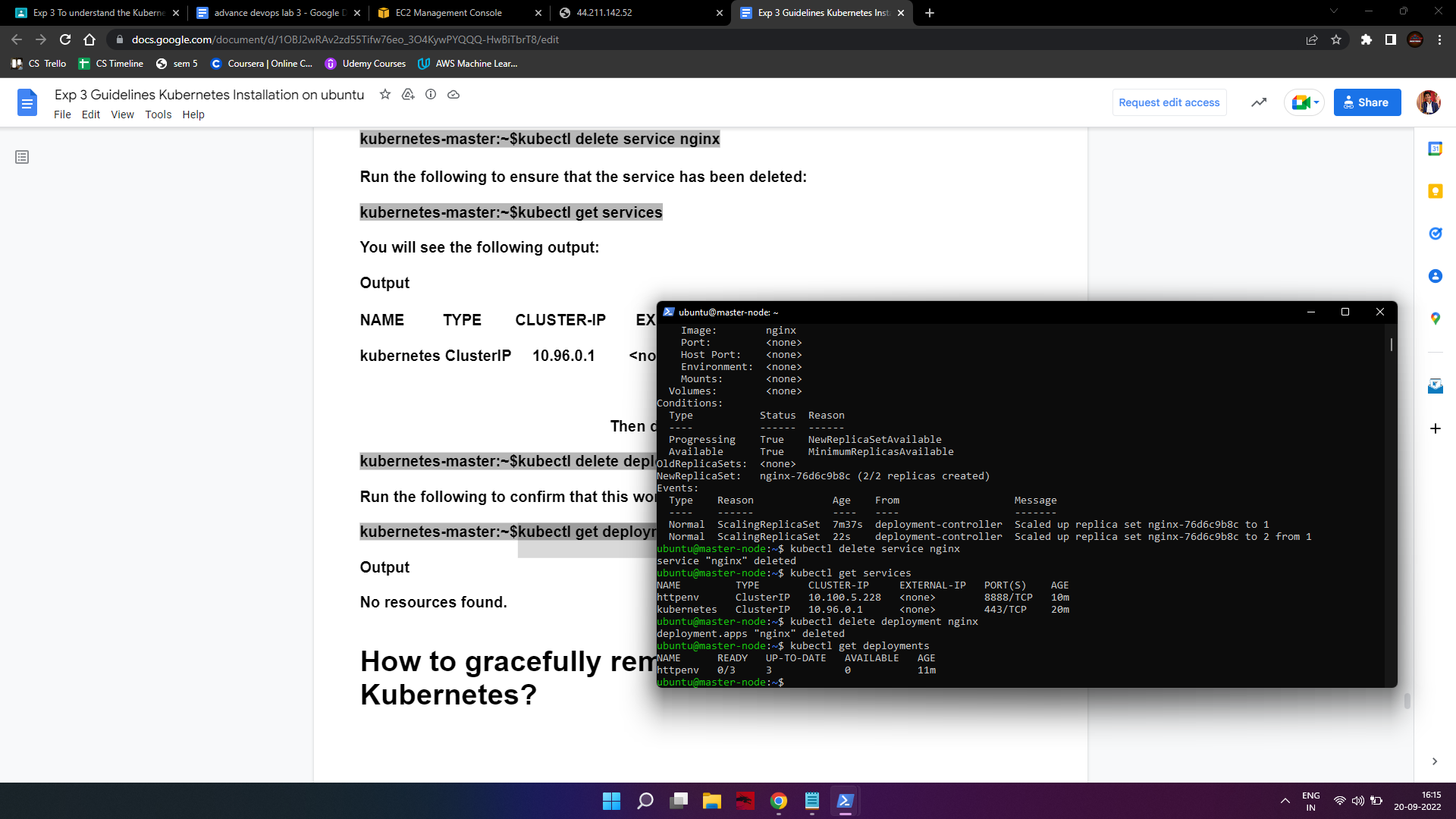
Output

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 1d

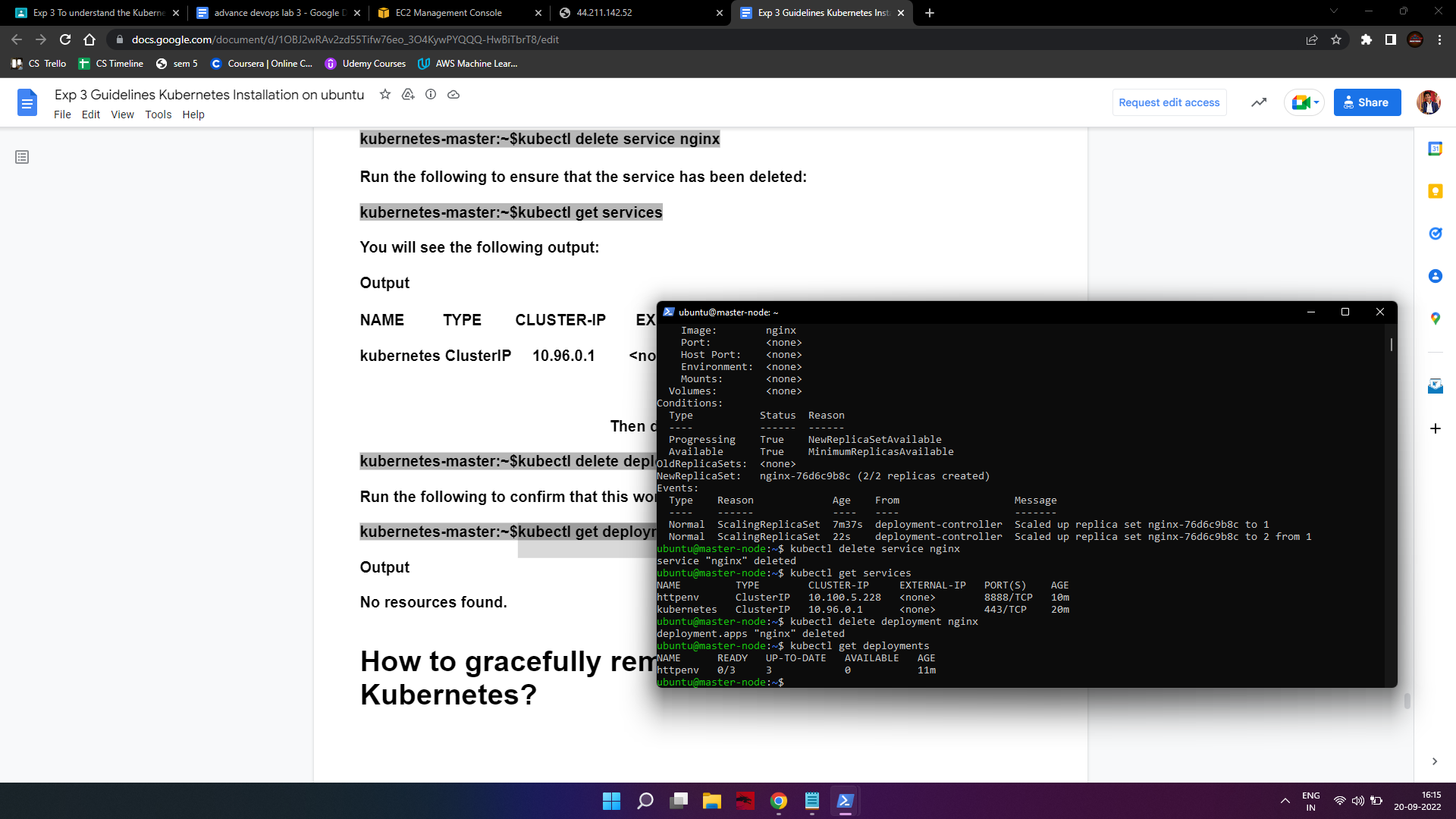
Then delete the deployment:

kubernetes-master:~$kubectl delete deployment nginx



Run the following to confirm that this worked:

kubernetes-master:~$kubectl get deployments



Output

No resources found.