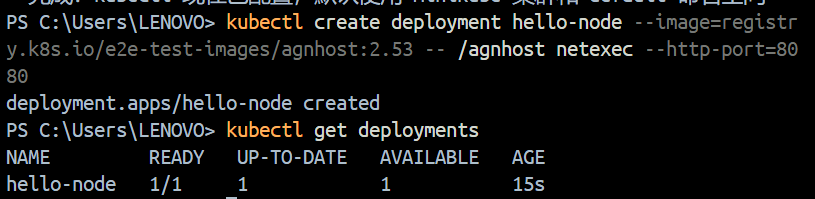
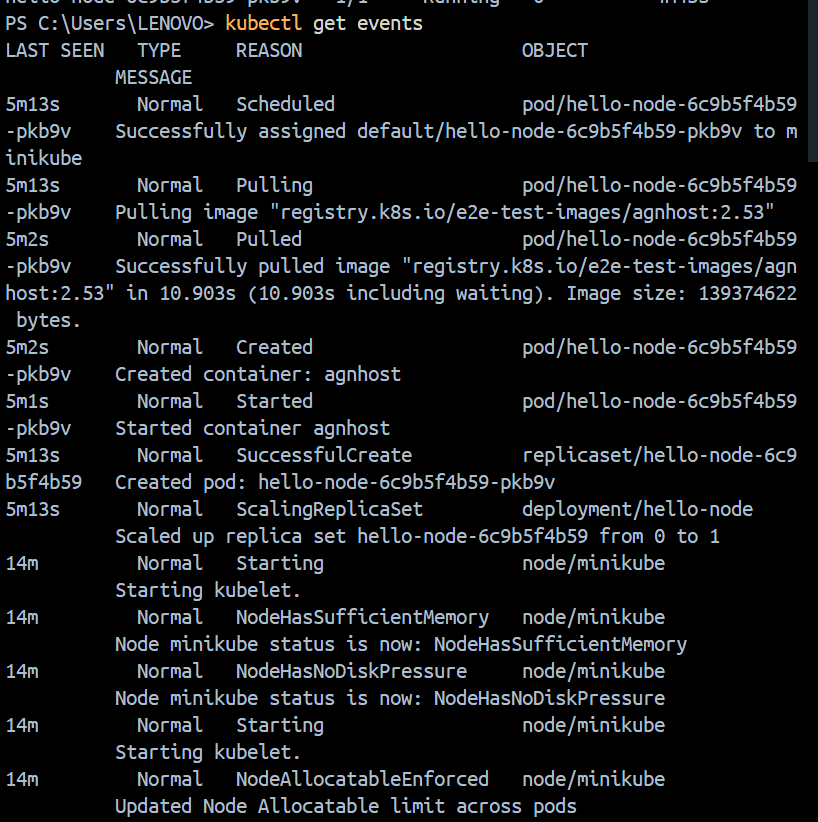
Use the kubectl create command to create a Deployment that manages a Pod. The Pod runs a Container based on the provided Docker image.

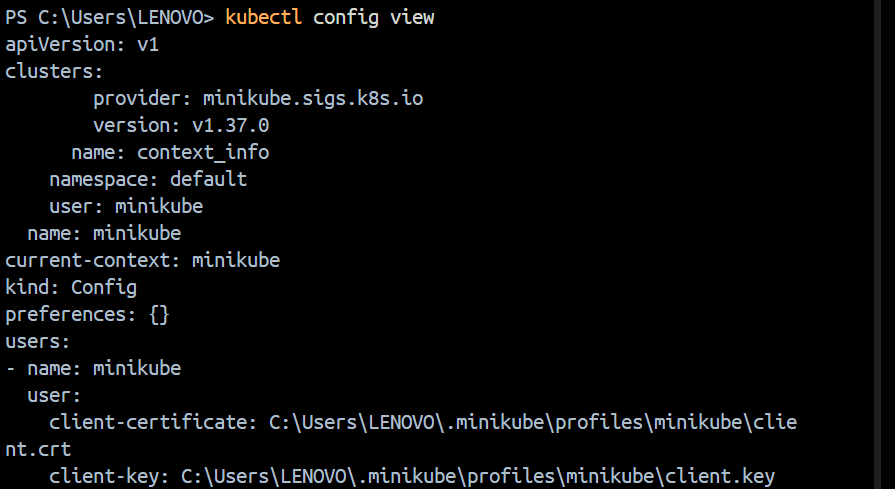
View the Deployment

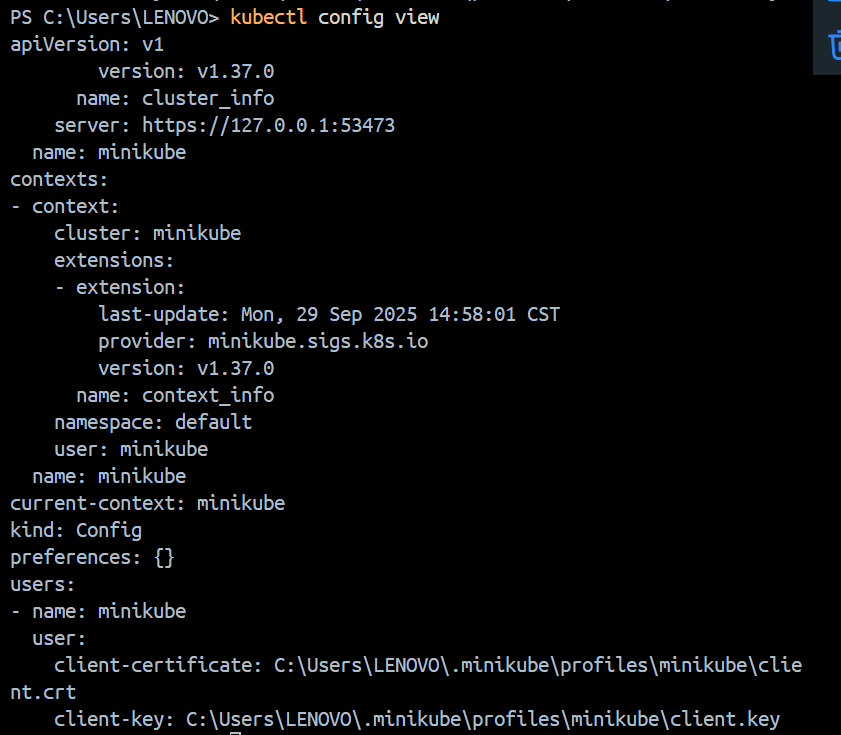


View cluster events



View the kubectl configuration:



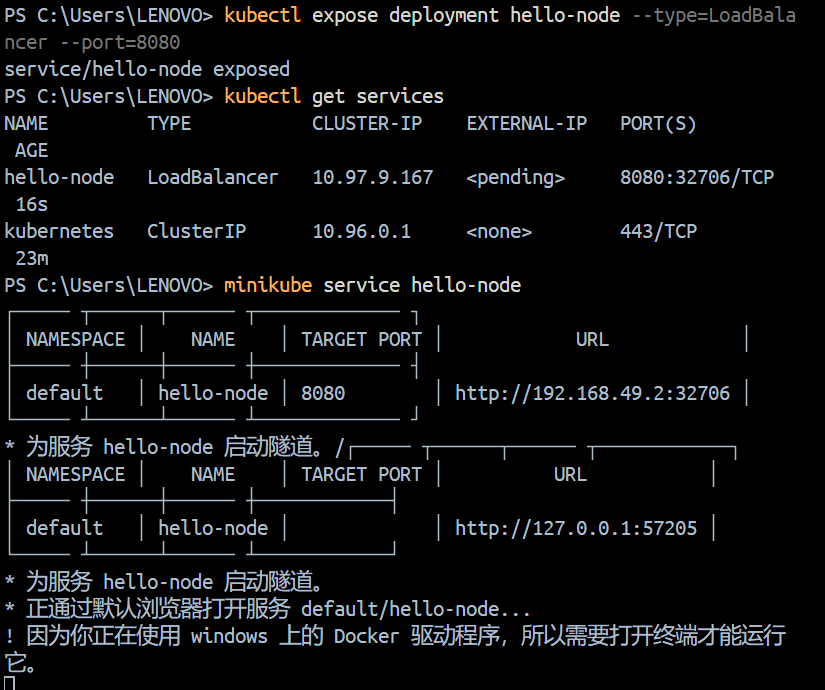


View application logs for a container in a pod (replace pod name with the one you got from kubectl get pods).

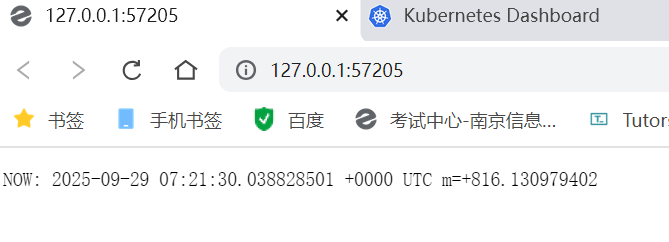
**Create a Service**

Expose the Pod to the public internet using the kubectl expose command:

View the Service you created:

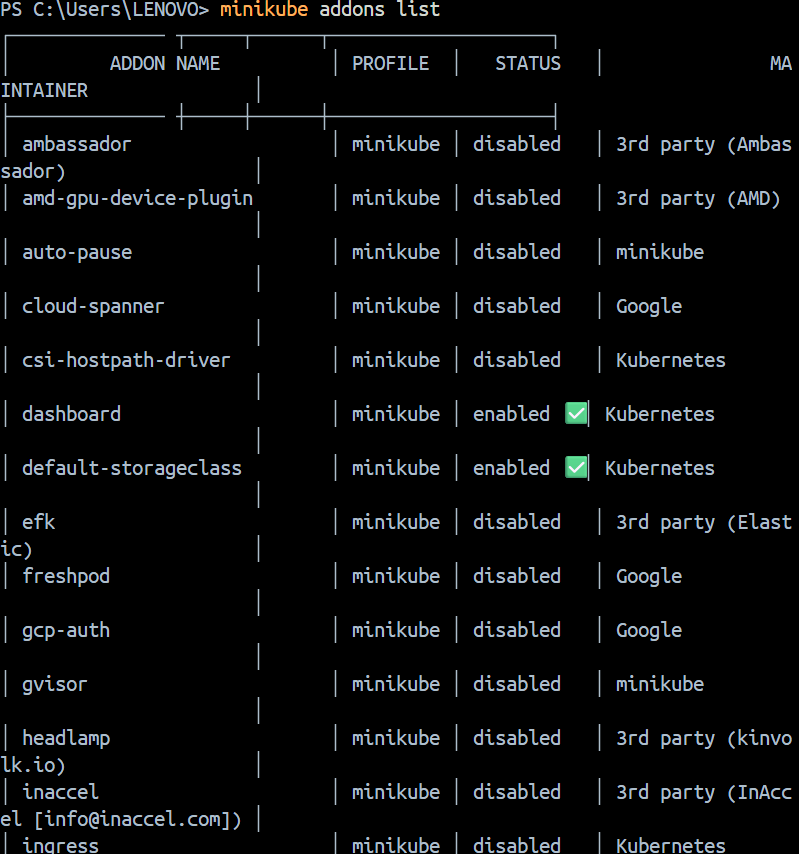


Run the command to opens up a browser windoe

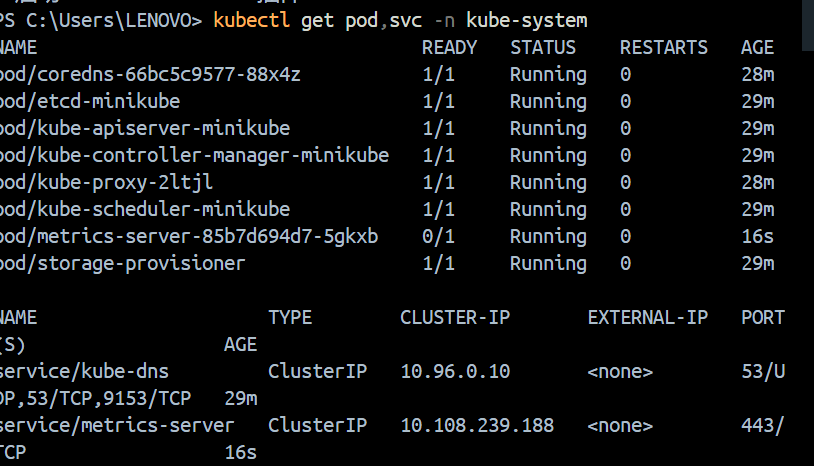


**Enable addons**

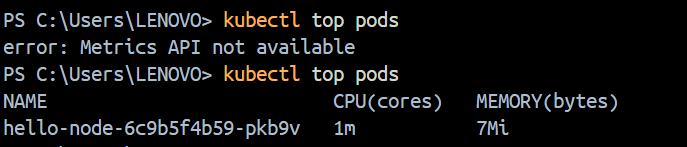
List the currently supported addons:



View the Pod and Service you created by installing that addon



Check the output from metrics-server



Disable metrics-server

Now you can clean up the resources you created in your cluster:

Stop the Minikube cluster

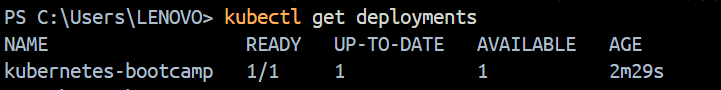


**Deploy an app**

Let’s deploy our first app on Kubernetes with the kubectl create deployment command. We need to provide the deployment name and app image location (include the full repository url for images hosted outside Docker Hub).

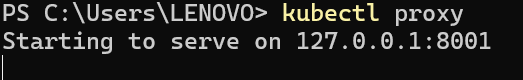


list the deployments

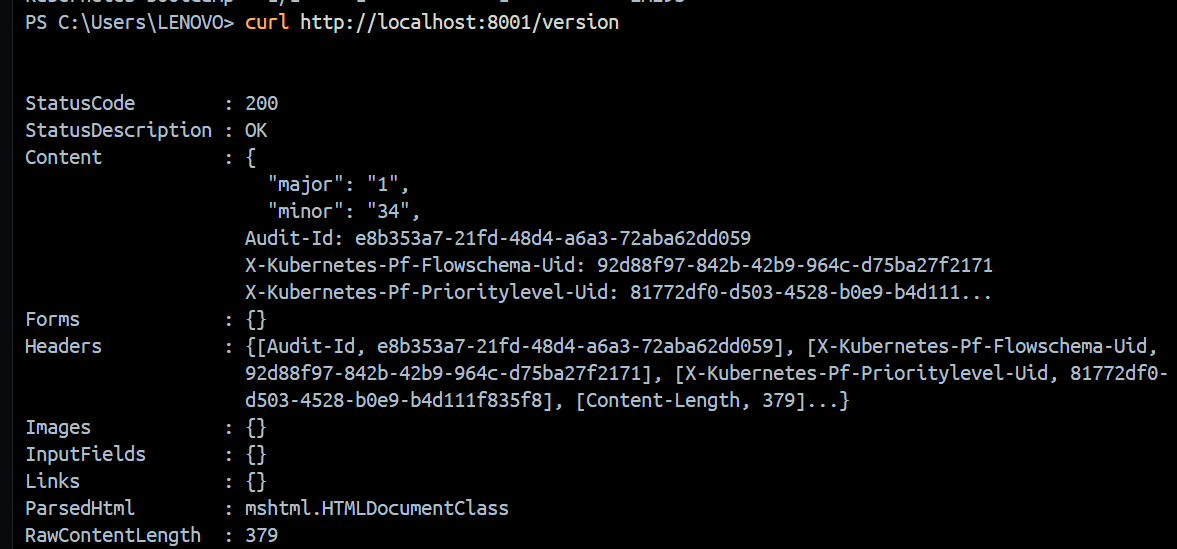


**View the app**

The kubectl proxy command can create a proxy that will forward communications into the cluster-wide, private network. The proxy can be terminated by pressing control-C and won't show any output while it's running.

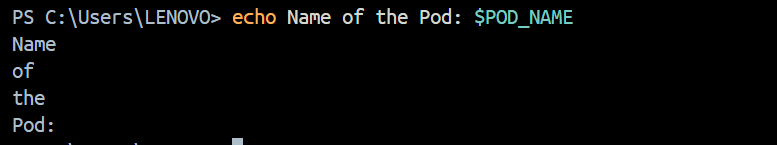


You can see all those APIs hosted through the proxy endpoint. For example, we can query the version directly through the API using the curl command:

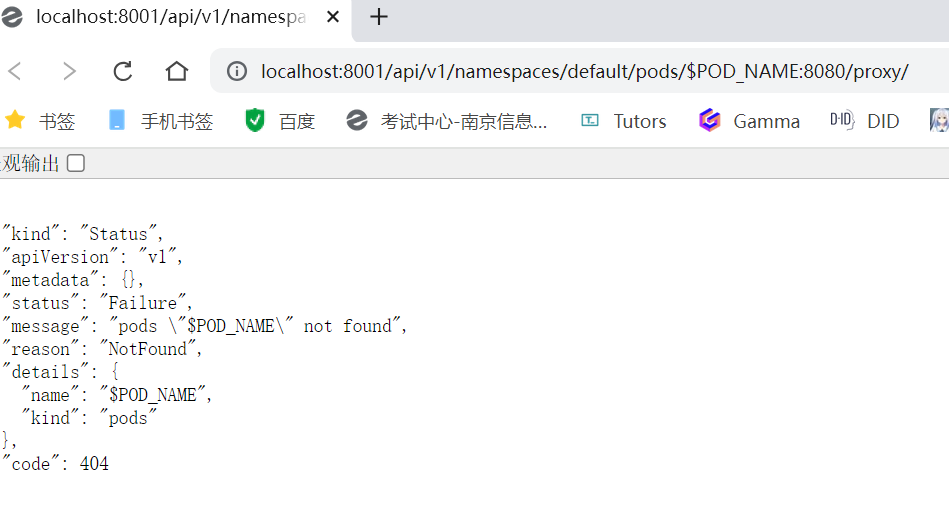


The API server will automatically create an endpoint for each pod, based on the pod name, that is also accessible through the proxy.

First get the Pod name, and store it in the environment variable POD\_NAME.

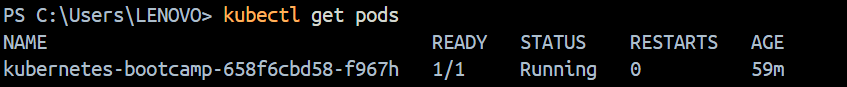


access the Pod through the proxied API

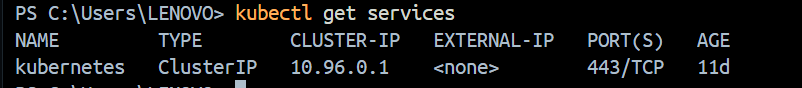


**Services and Labels**

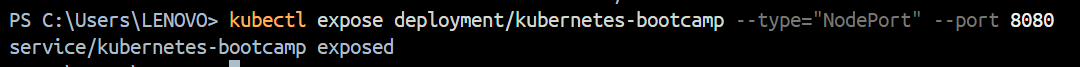
use the kubectl get command and look for existing Pods:

****

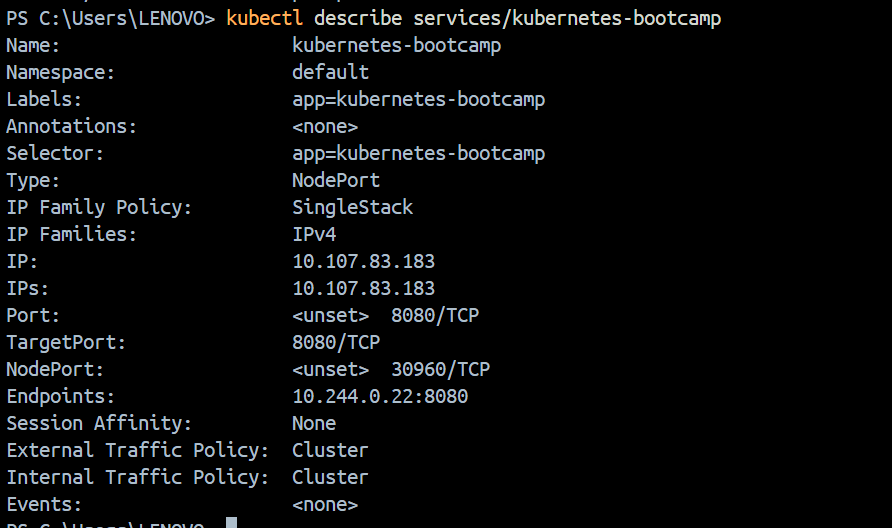
list the current Services from our cluster:

****

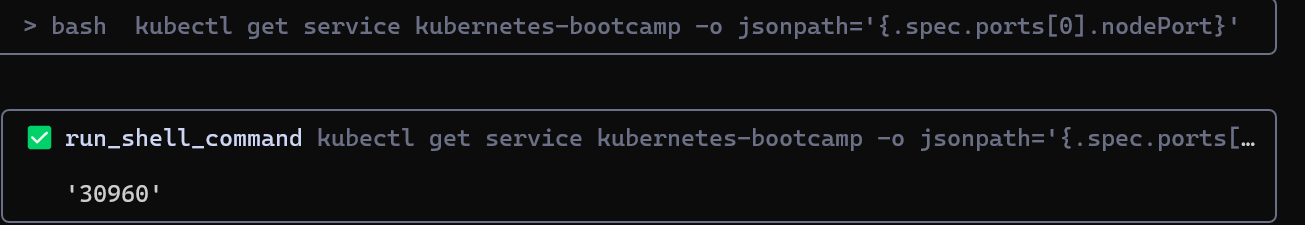
To expose the deployment to external traffic, we'll use the kubectl expose command with the --type=NodePort option:



To find out what port was opened externally (for the type: NodePort Service) we’ll run the describe service subcommand:

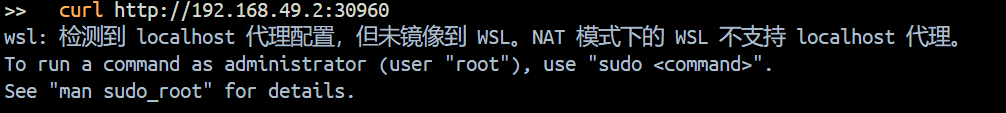


Create an environment variable called NODE\_PORT that has the value of the Node port assigned:



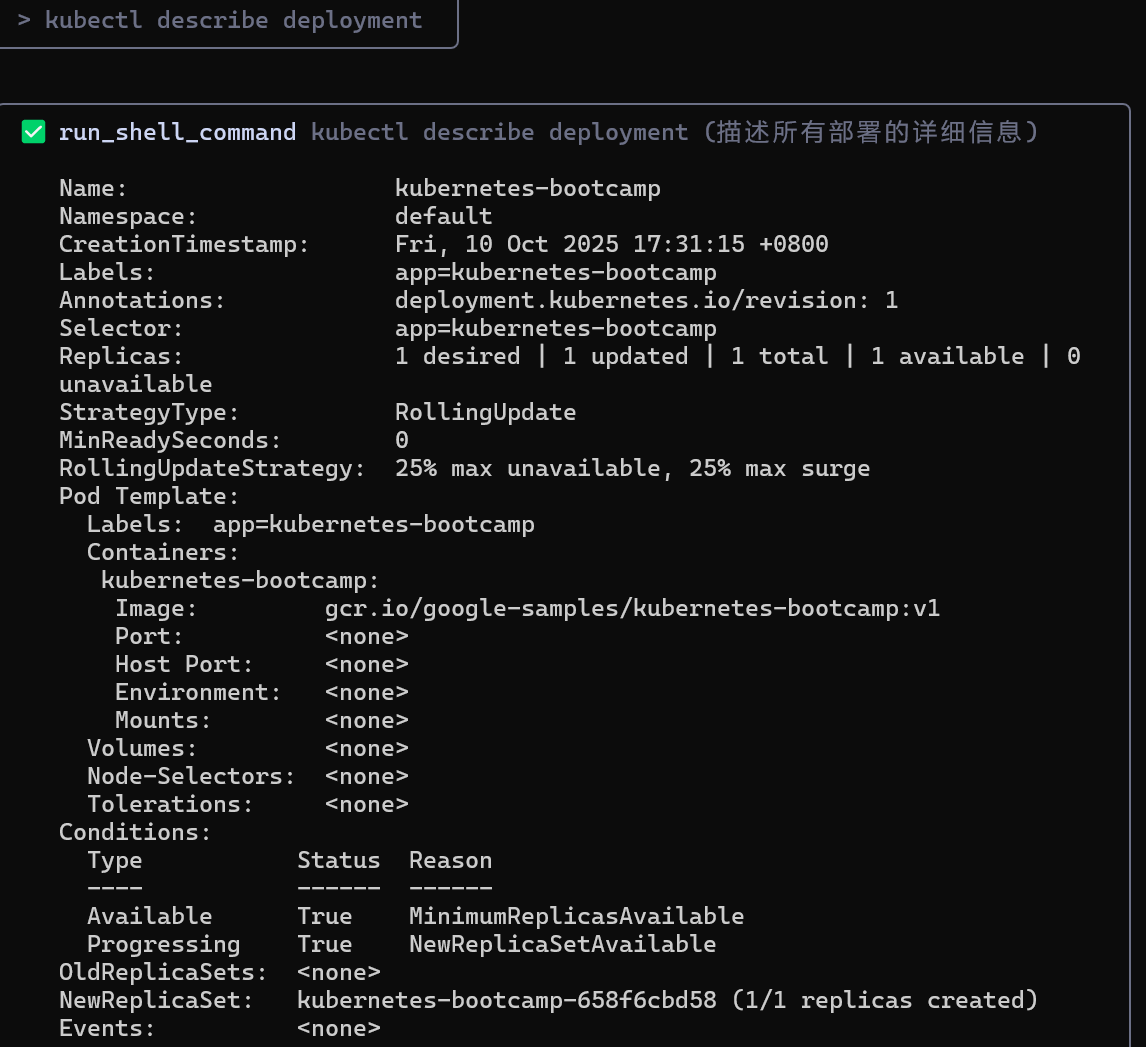


Now we can test that the app is exposed outside of the cluster using curl, the IP address of the Node and the externally exposed port:

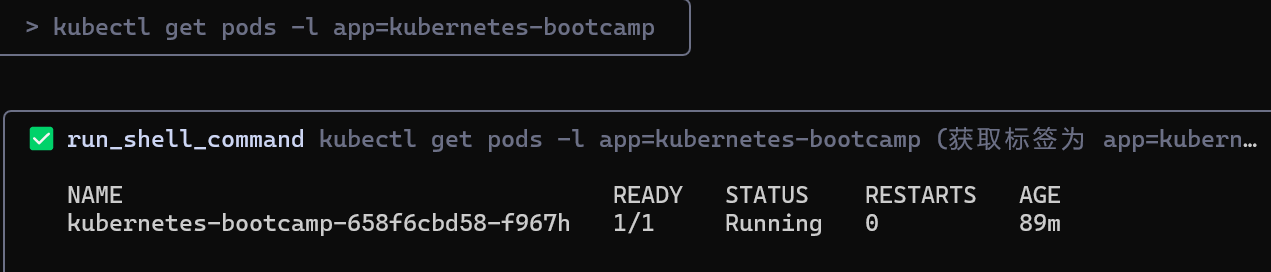


**Using labels**

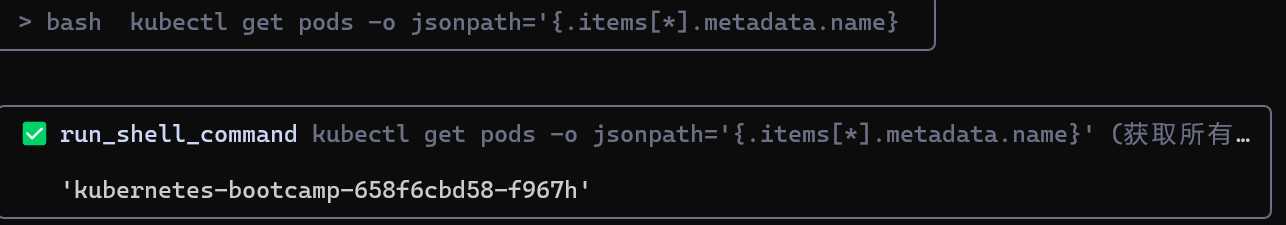
The Deployment created automatically a label for our Pod. With the describe deployment subcommand you can see the name (the *key*) of that label:



use the kubectl get pods command with -l as a parameter, followed by the label values:

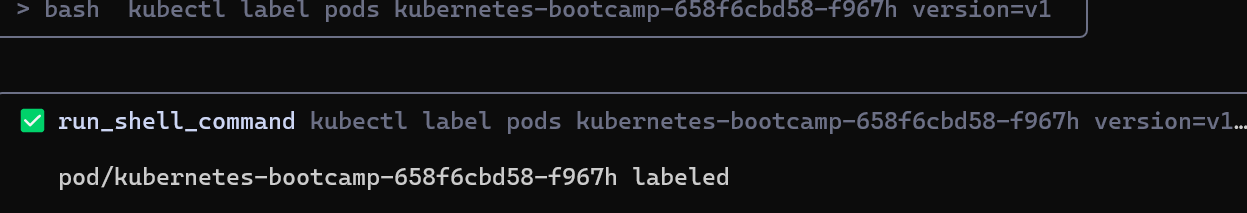


Get the name of the Pod and store it in the POD\_NAME environment variable:

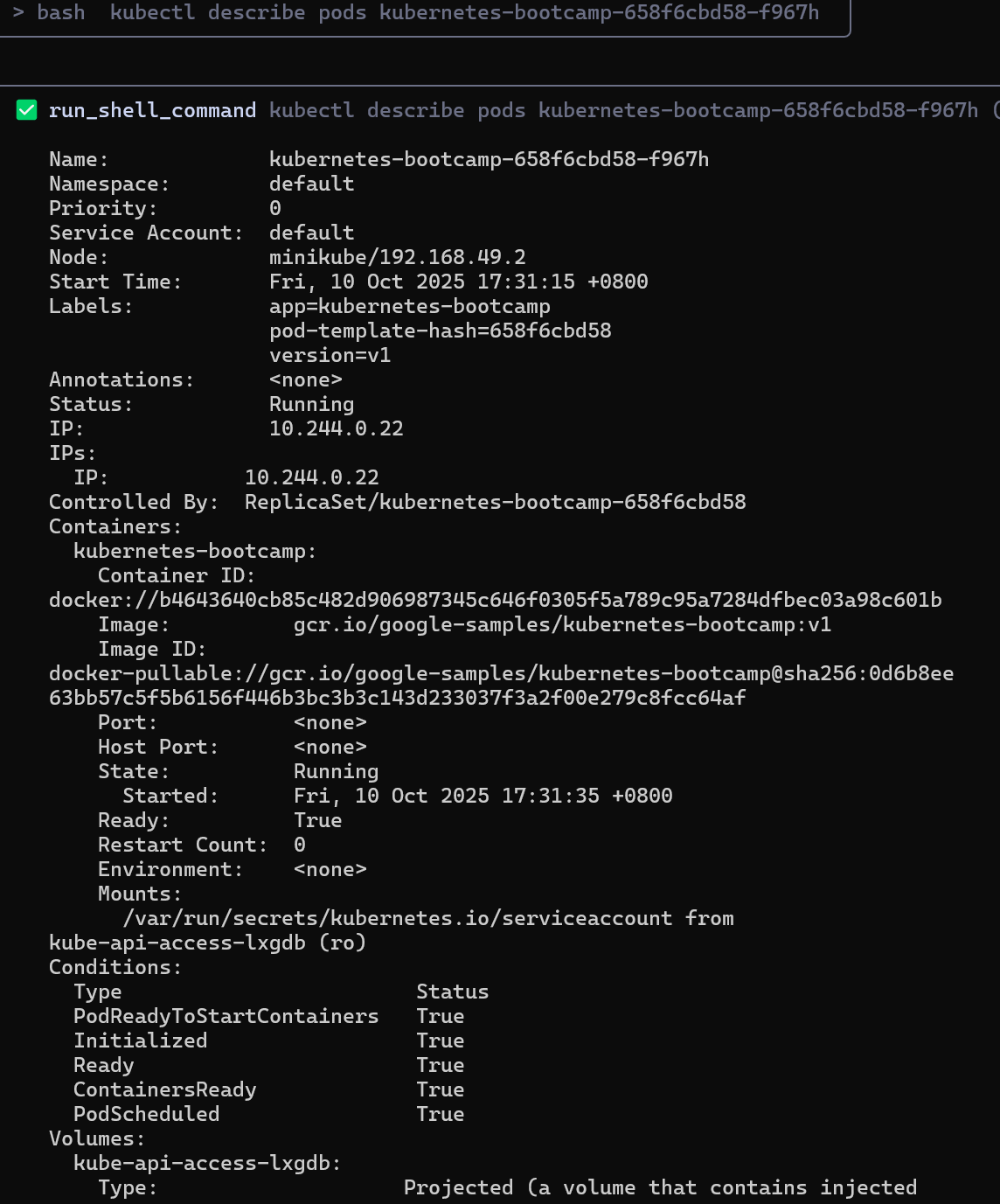


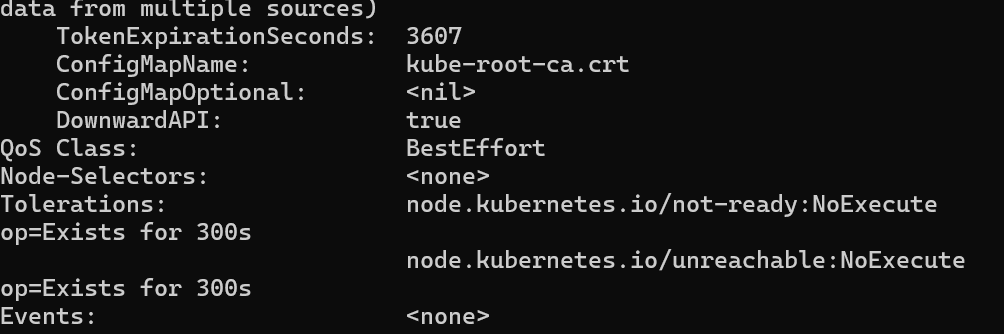


To apply a new label we use the label subcommand followed by the object type, object name and the new label:

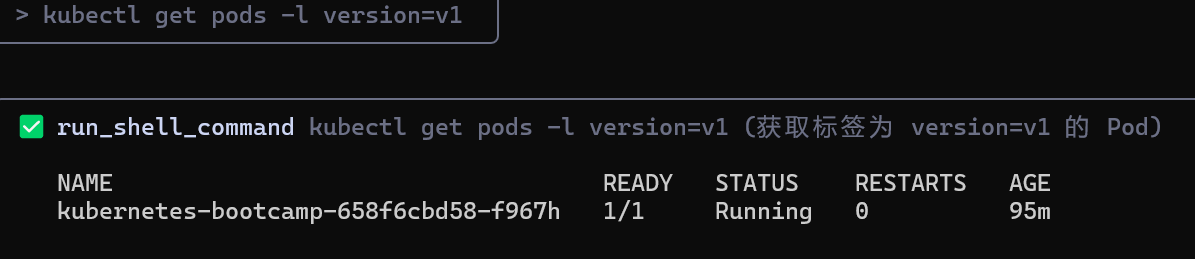


This will apply a new label to our Pod (we pinned the application version to the Pod), and we can check it with the describe pod command

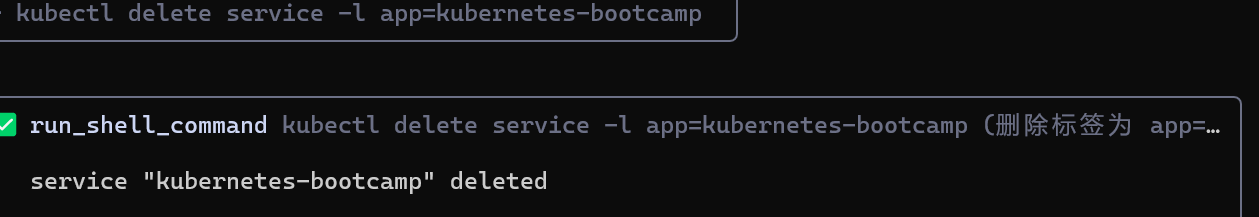




We see here that the label is attached now to our Pod. And we can query now the list of pods using the new label:

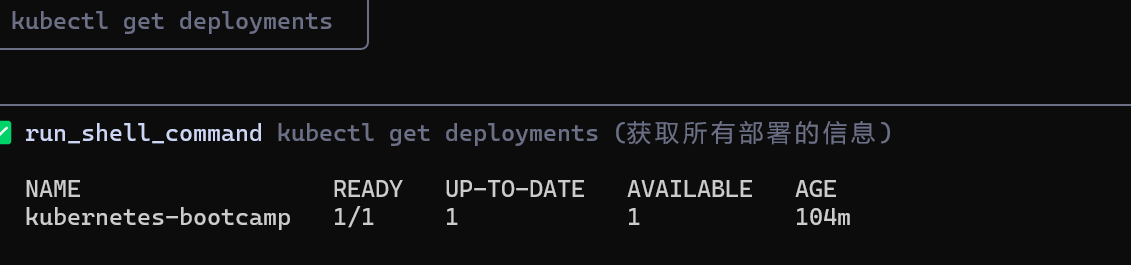


**Deleting a service**

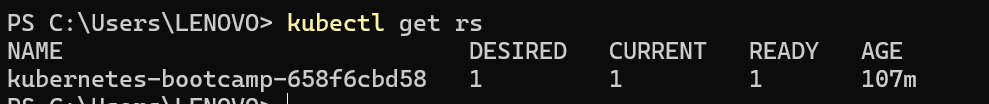


**Scaling a Deployment**

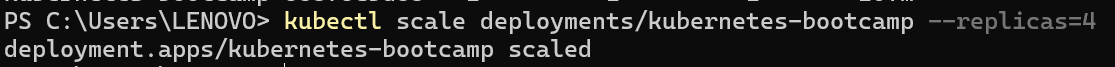
list Deployments

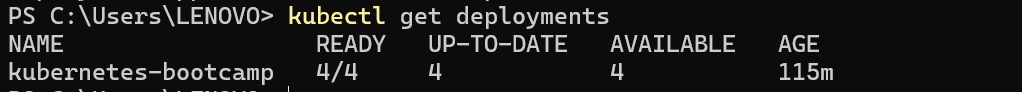


see the ReplicaSet created by the Deployment

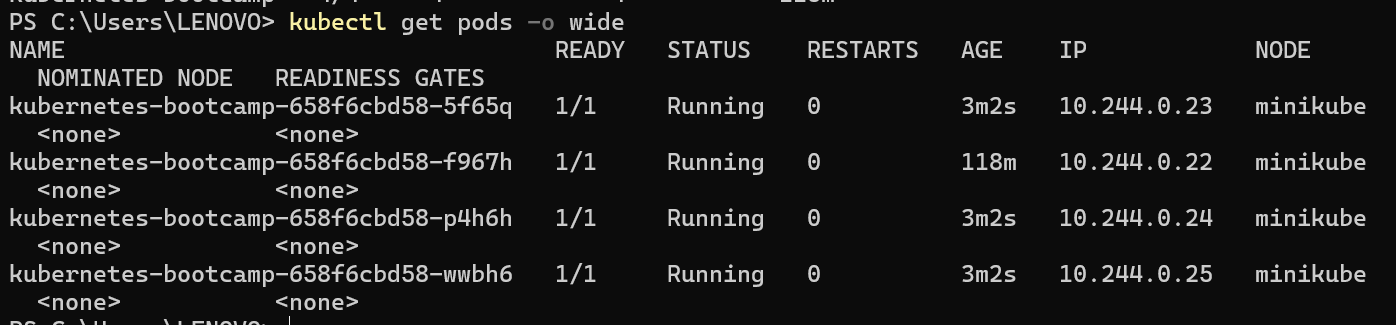


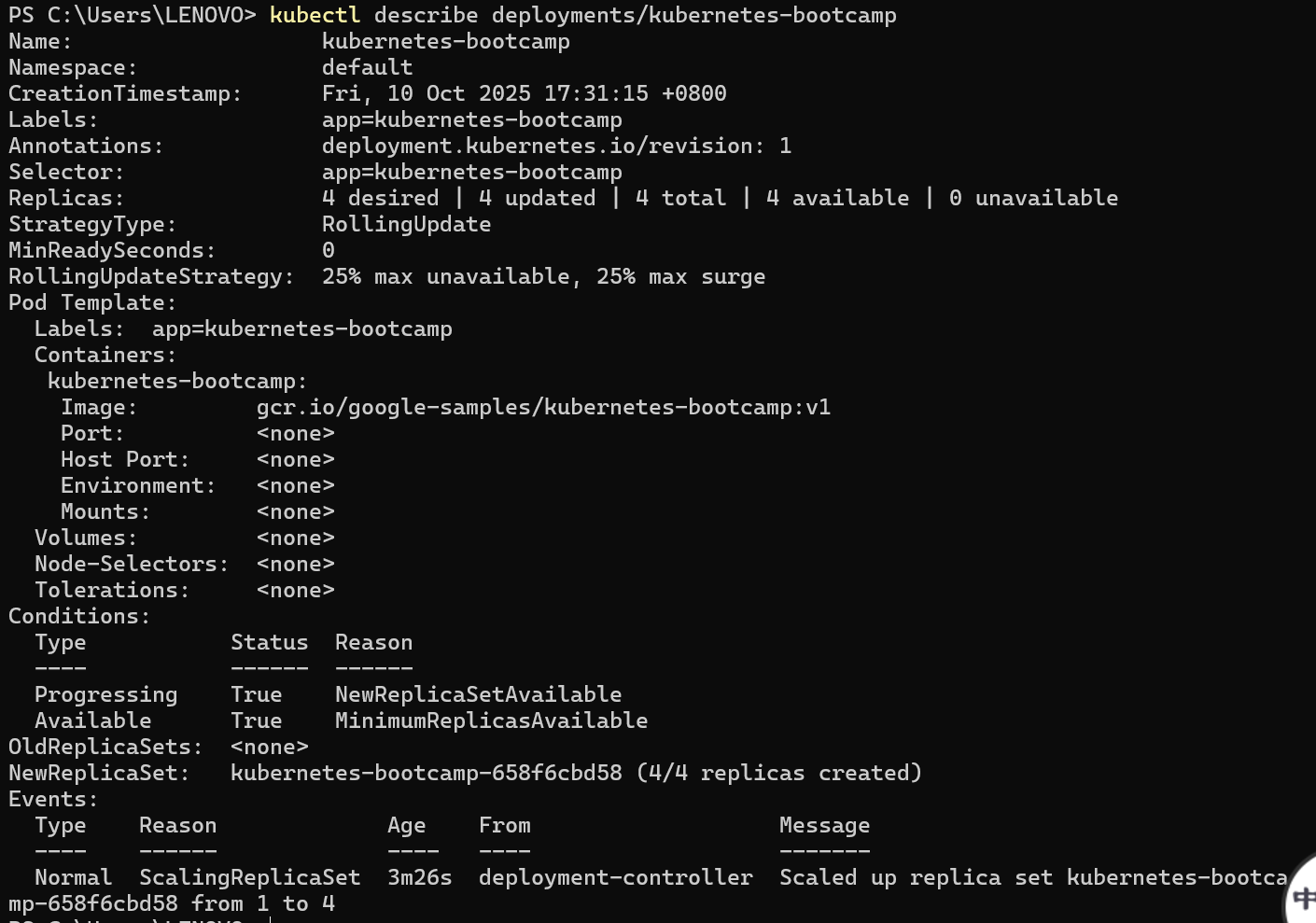
scale the Deployment to 4 replicas.



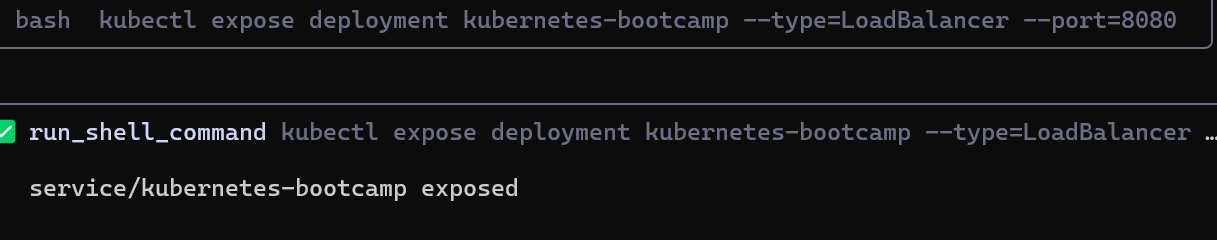


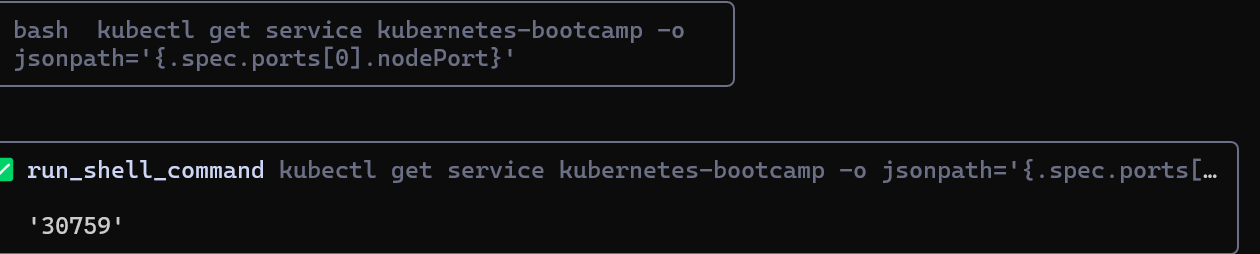
 check if the number of Pods changed

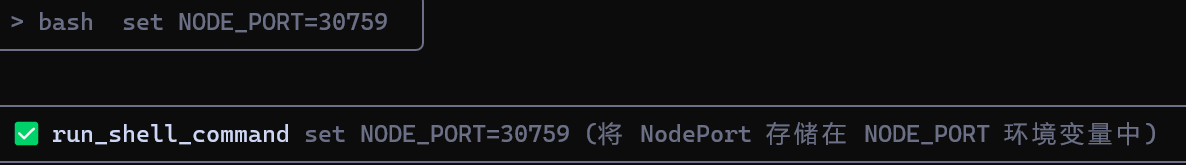




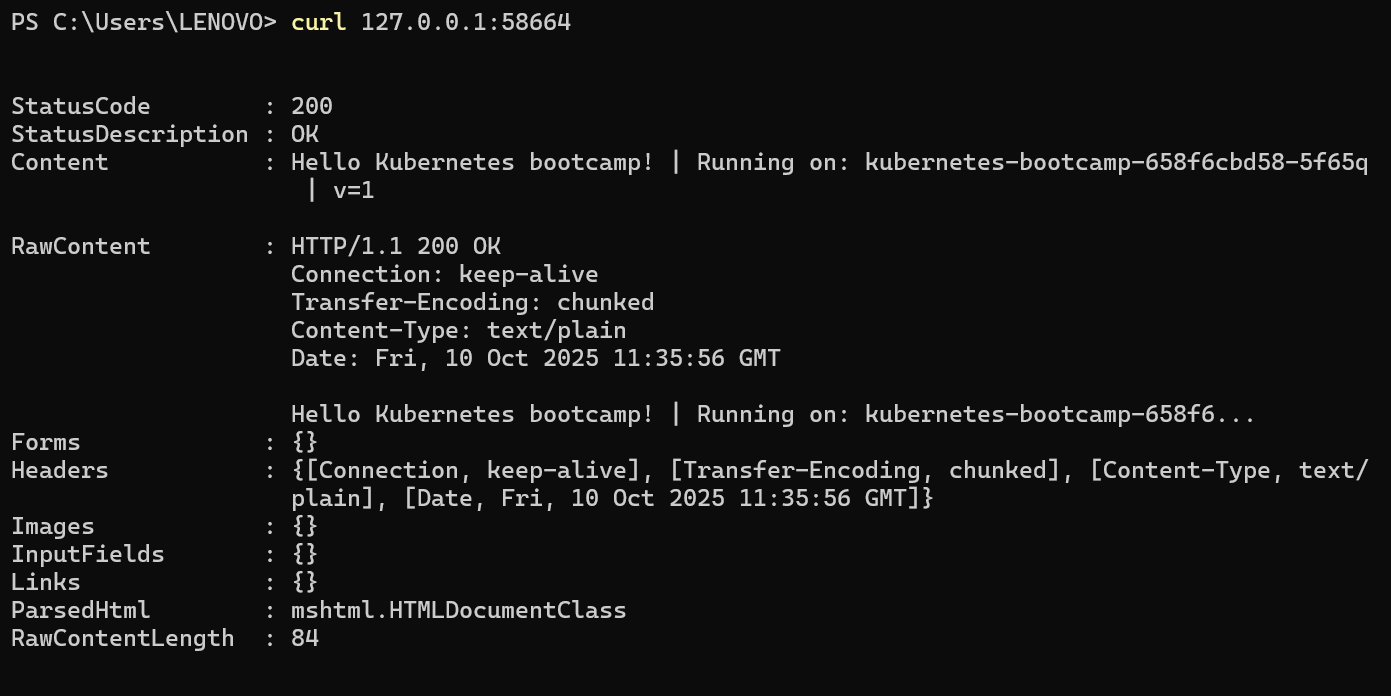
check that the Service is load-balancing the traffic.







do a curl to the exposed IP address and port.

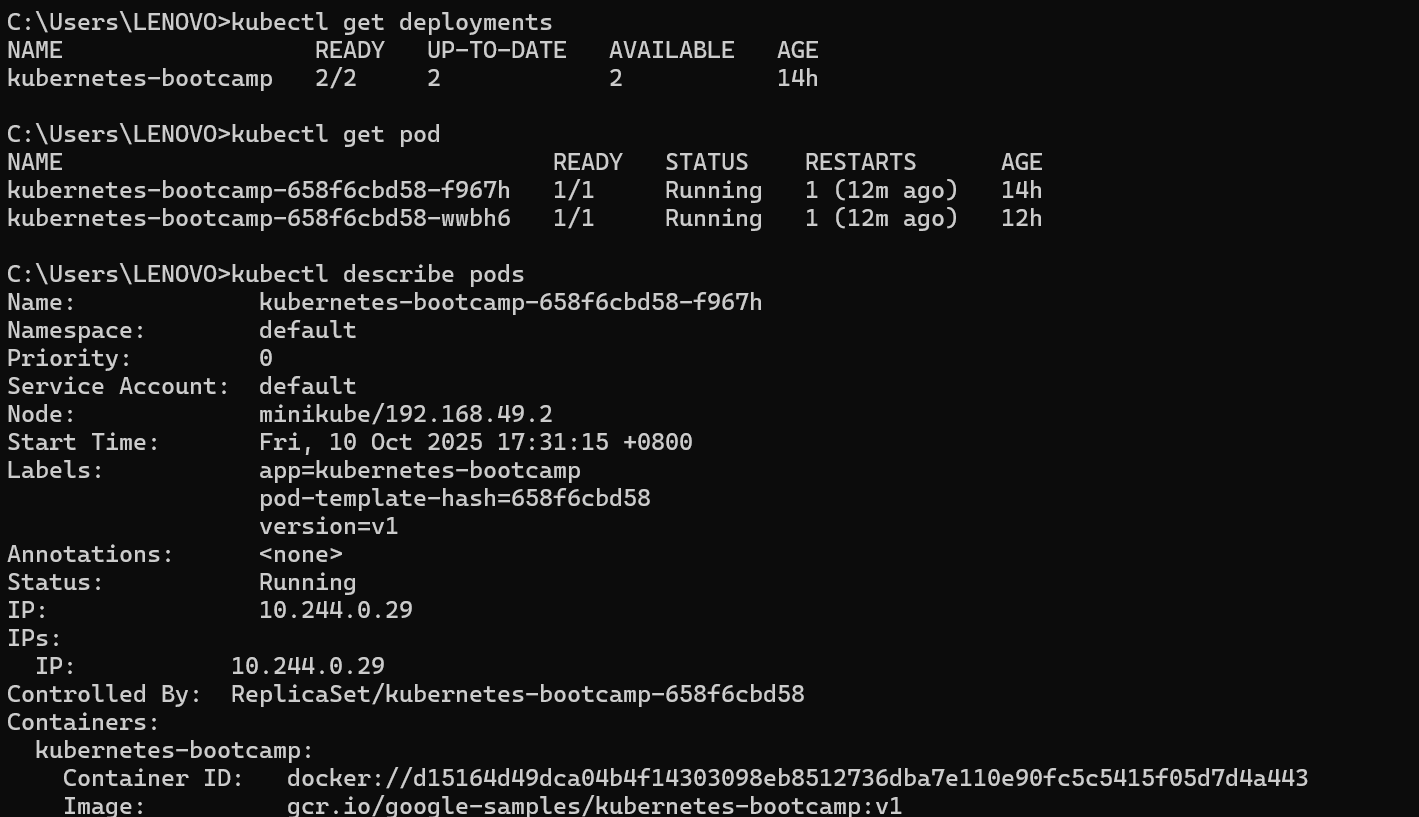


**Scale Down**

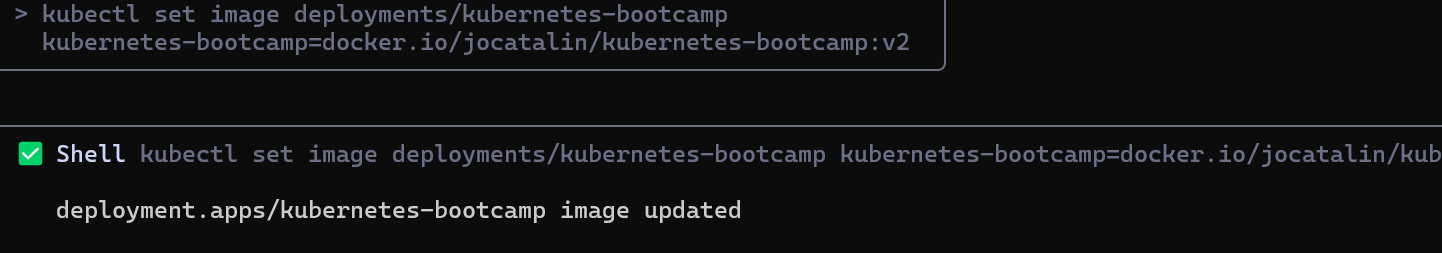


The number of replicas decreased to 2. 

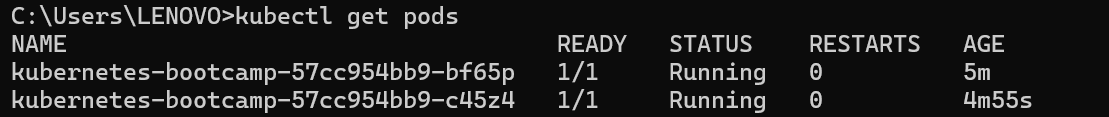
**Update the version of the app**

****

update the image of the application to version 2

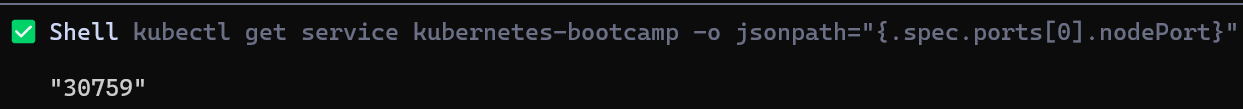


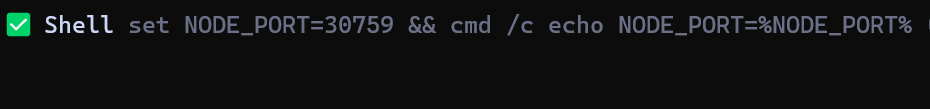
Check the status of the new Pods, and view the old one terminating with the get pods subcommand:

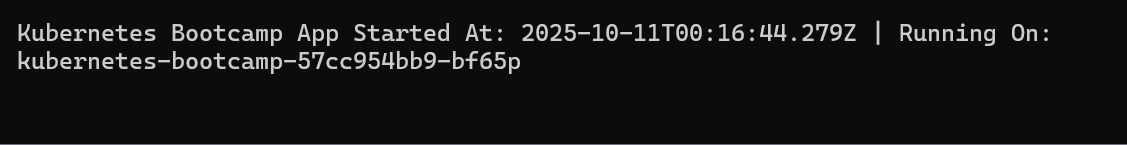


**Verify an update**

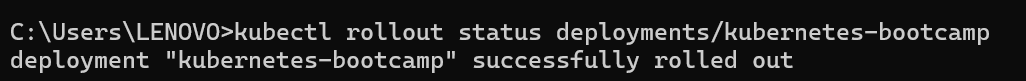
Create an environment variable called NODE\_PORT that has the value of the Node port assigned:



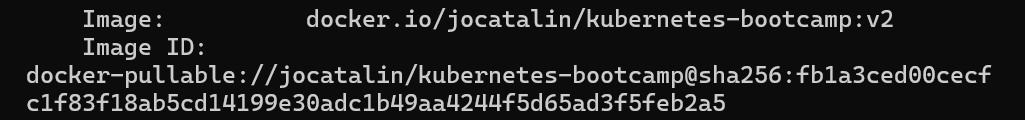




You can also confirm the update by running the rollout status subcommand:

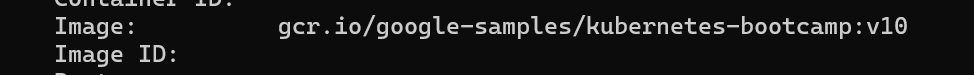


To view the current image version of the app, run the describe pods subcommand:

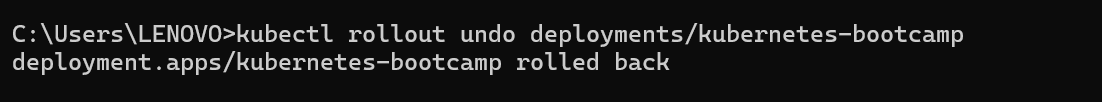


**Roll back an update**

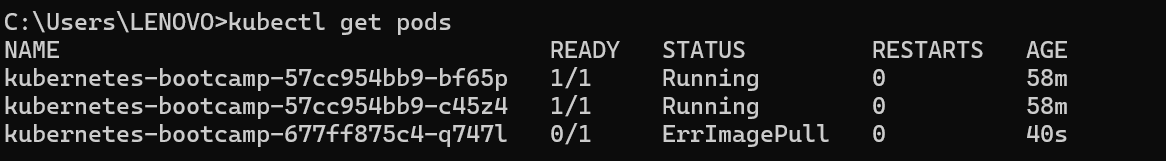
 deploy an image tagged with v10:



To roll back the deployment to your last working version, use the rollout undo subcommand:



Use the get pods subcommand to list the Pods again:



 check the image deployed on the running Pods

