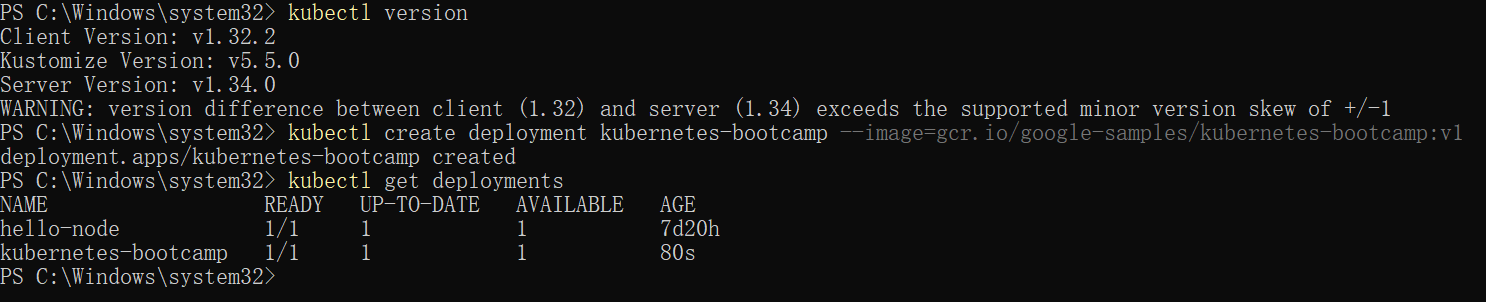
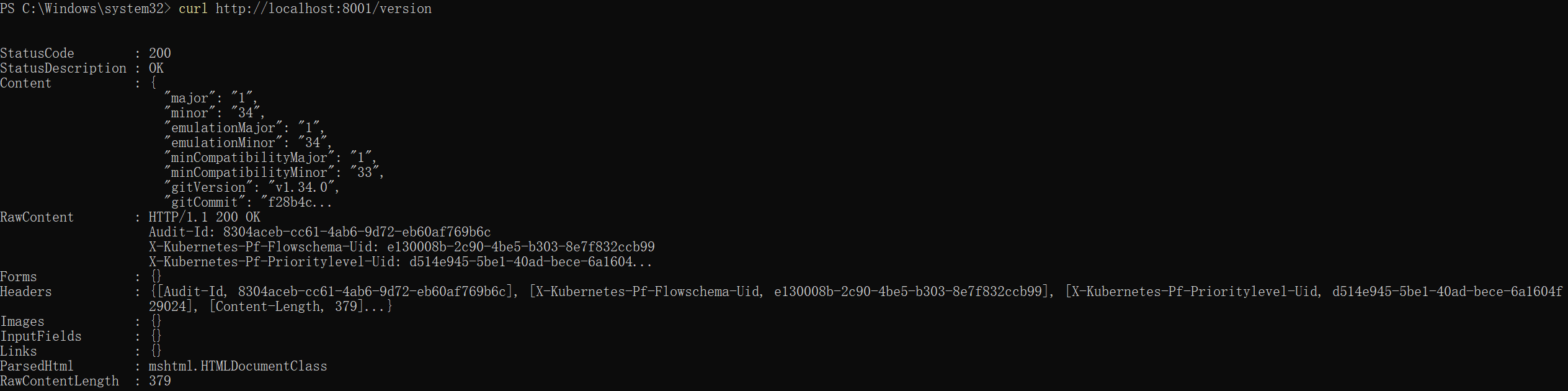
Using kubectl to Create a Deployment

1. deploy our first app on Kubernetes with the kubectl create deployment command.

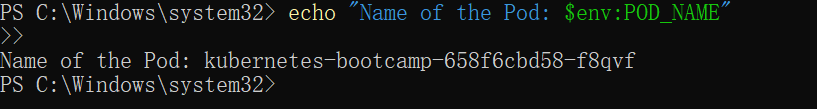


1. we can query the version directly through the API using the curl command:



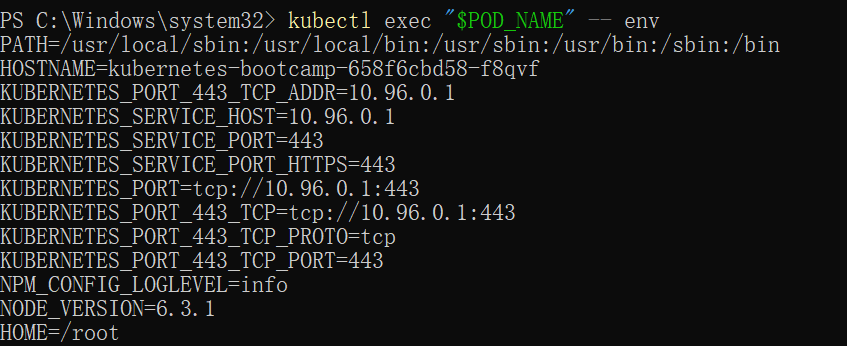
Viewing Pods and Nodes

1. First we need to get the Pod name, and we'll store it in the environment variable





1. We can execute commands directly on the container once the Pod is up and running. For this, we use the exec subcommand and use the name of the Pod as a parameter. Let’s list the environment variables:



1. Next let’s start a bash session in the Pod’s container:

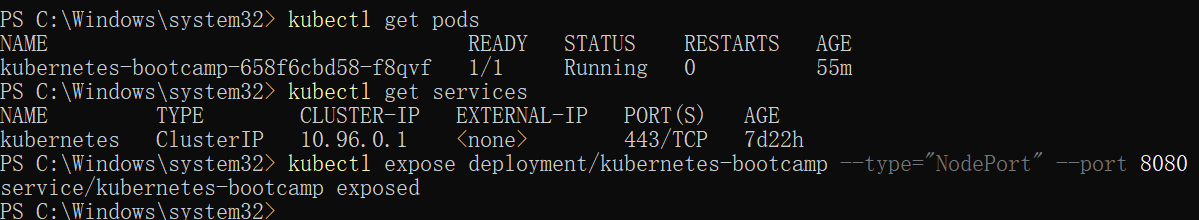


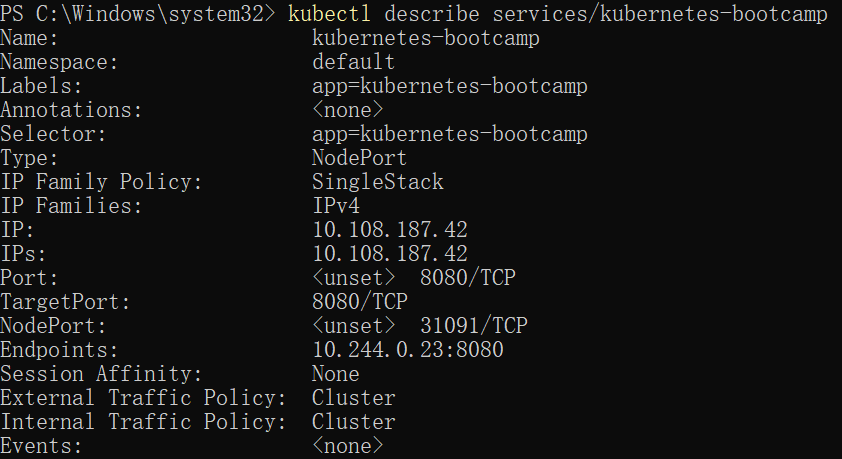


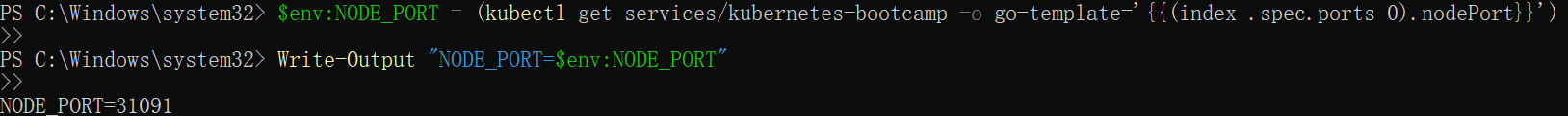
Using a Service to Expose Your App

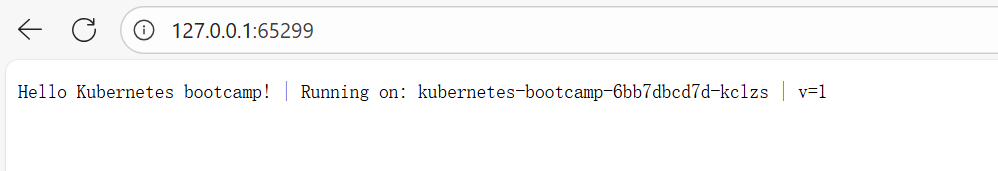
1. Creating a new Service

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



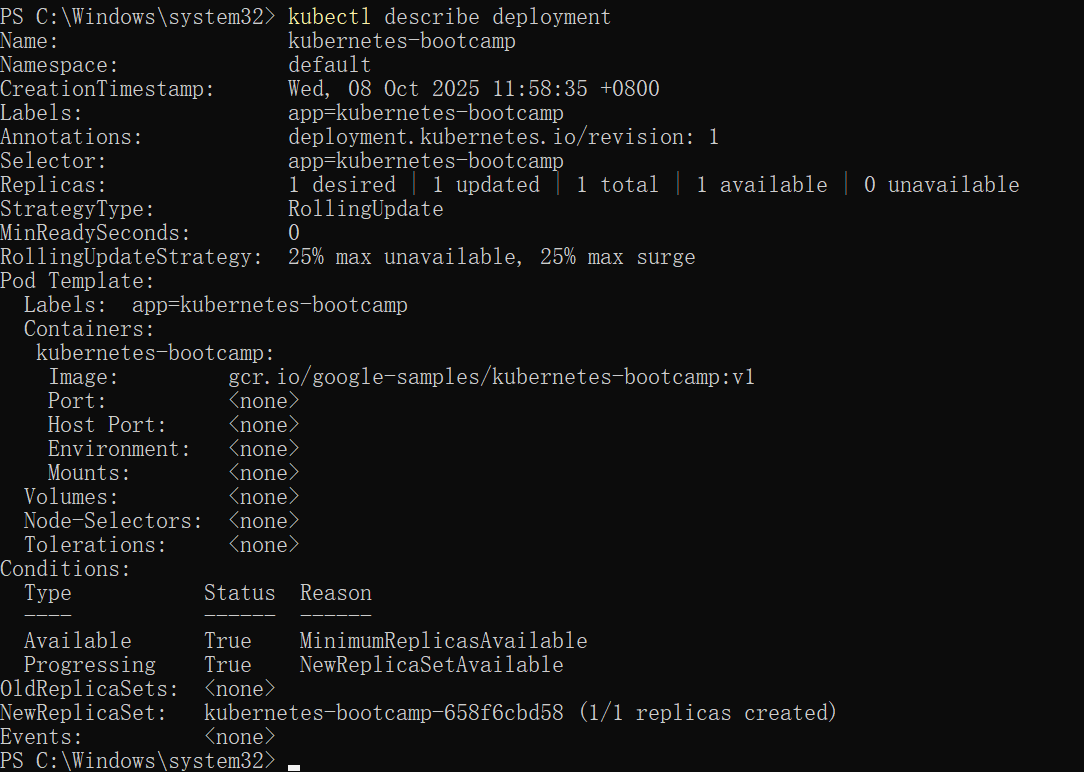
1. To find out what port was opened externally (for the type: NodePort Service) we’ll run the describe service subcommand:
2. 
3. Create an environment variable called NODE\_PORT that has the value of the Node port assigned:





1. Using labels

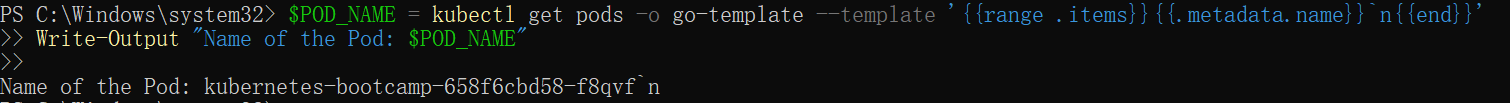
The Deployment created automatically a label for our Pod. With the describe deployment subcommand you can see the name (the key) of that label:



1. We’ll use the kubectl get pods command with -l as a parameter, followed by the label values:

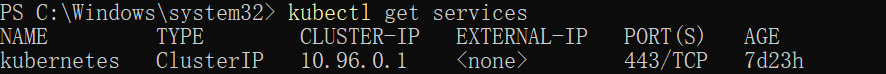


Get the name of the Pod and store it in the POD\_NAME environment variable:



1. To apply a new label we use the label subcommand followed by the object type, object name and the new label:
2. 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:
3. 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:
4. Deleting a service



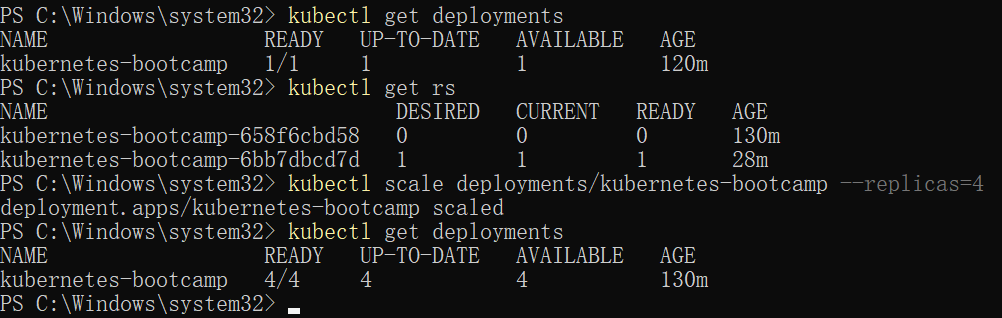


Running Multiple Instances of Your App

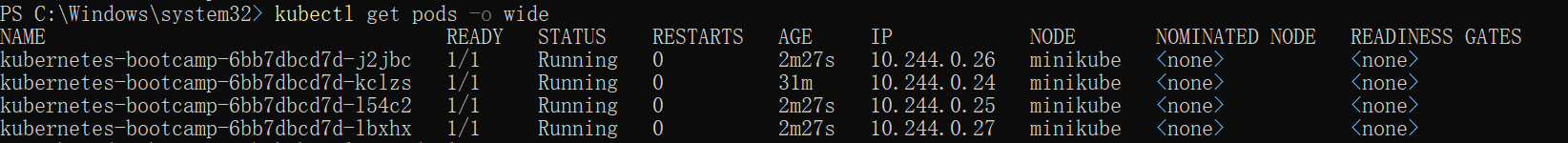
1. To list your Deployments, use the get deployments subcommand:

To see the ReplicaSet created by the Deployment, run:

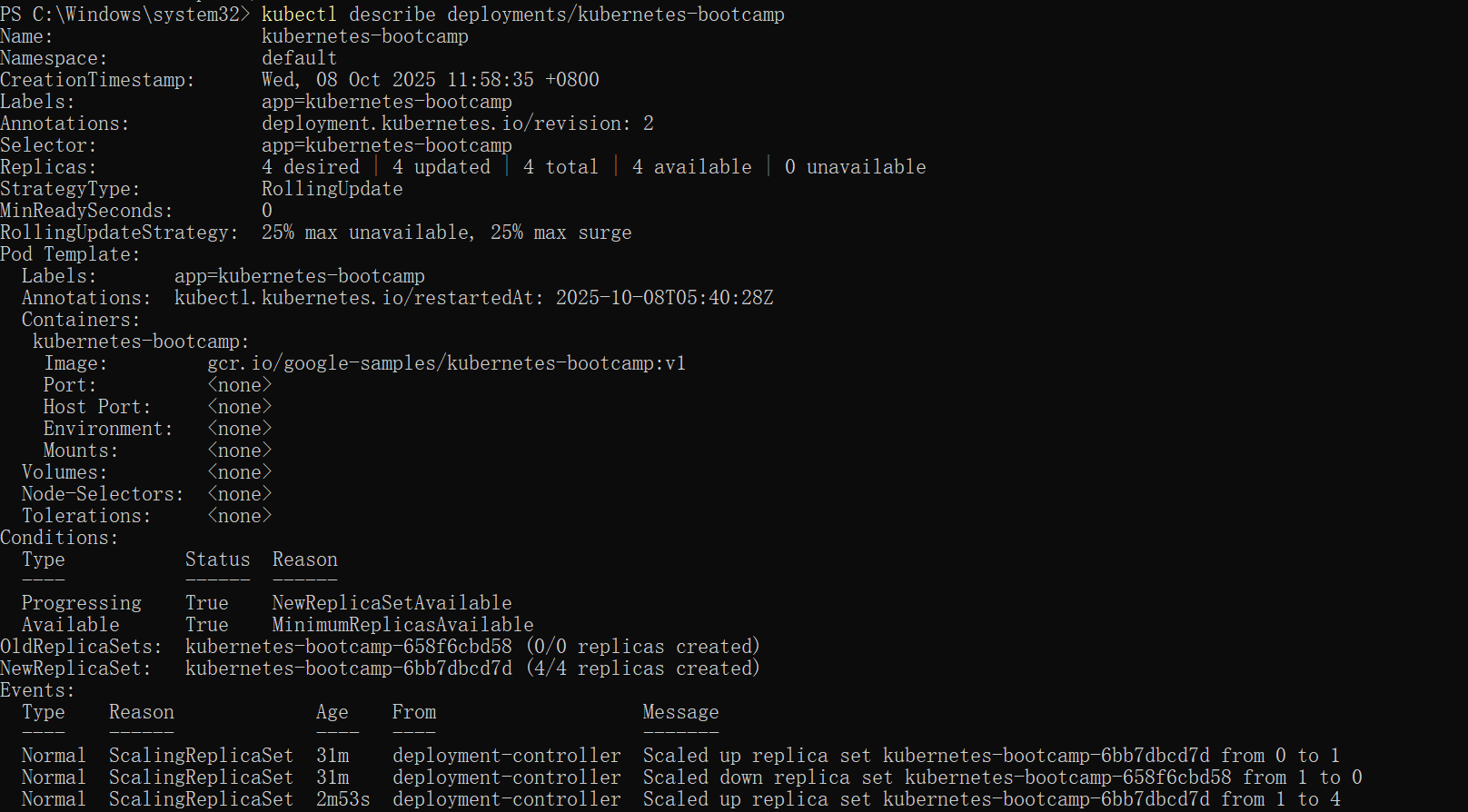
Next, let’s scale the Deployment to 4 replicas. We’ll use the kubectl scale command, followed by the Deployment type, name and desired number of instances:



1. The change was applied, and we have 4 instances of the application available. Next, let’s check if the number of Pods changed:



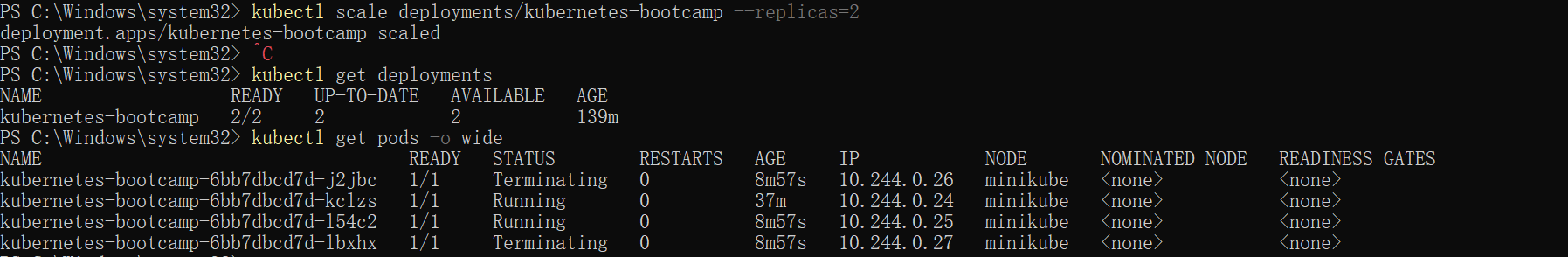
1. There are 4 Pods now, with different IP addresses. The change was registered in the Deployment events log. To check that, use the describe subcommand:



1. To scale down the Deployment to 2 replicas, run again the scale subcommand:

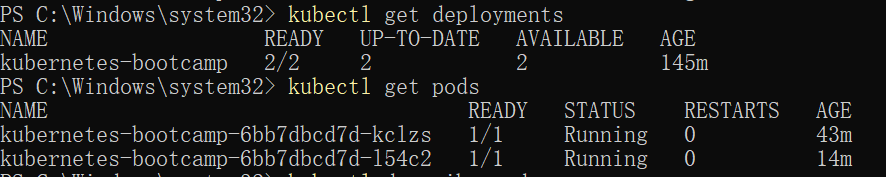
List the Deployments to check if the change was applied with the get deployments subcommand:

The number of replicas decreased to 2. List the number of Pods, with get pods:



Performing a Rolling Update

1. To list your Deployments, run the get deployments subcommand:
2. To list the running Pods, run the get pods subcommand:



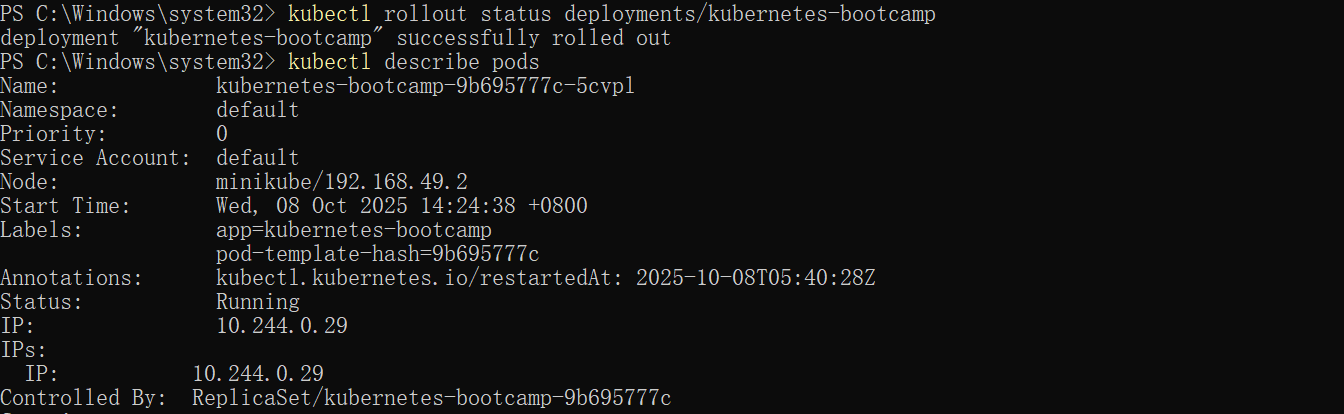
1. To update the image of the application to version 2, use the set image subcommand, followed by the deployment name and the new image version:



1. Verify an update，First, check that the service is running, as you might have deleted it in previous tutorial step, run describe services/kubernetes-bootcamp. If it's missing, you can create it again with:



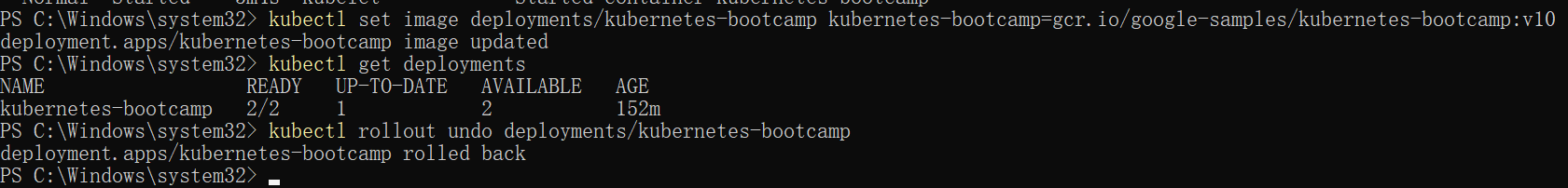
1. You can also confirm the update by running the rollout status subcommand:



1. Roll back an update

Let’s perform another update, and try to deploy an image tagged with v10:

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



The Deployment is once again using a stable version of the app (v2). The rollback was successful.

Remember to clean up your local cluster.

