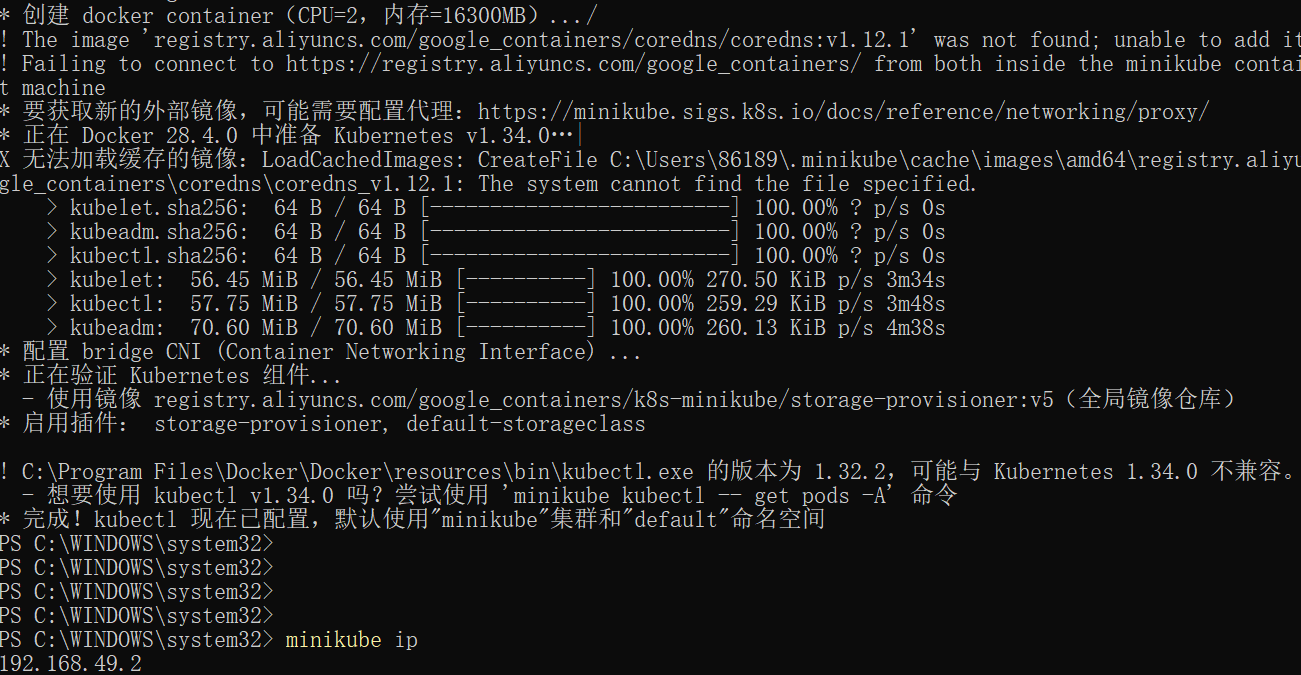
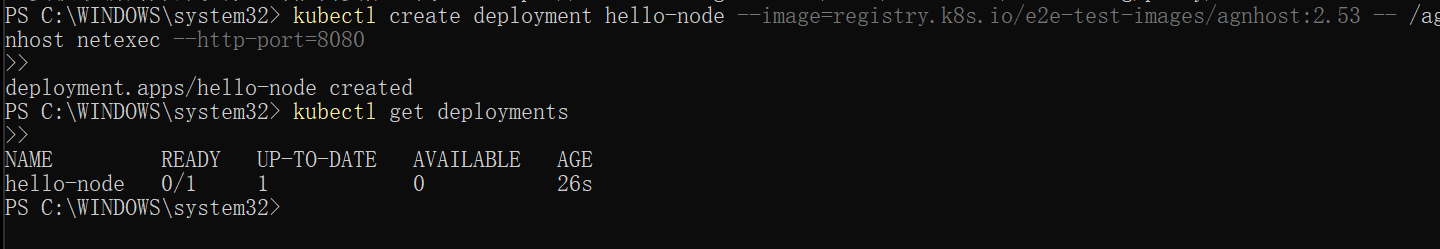
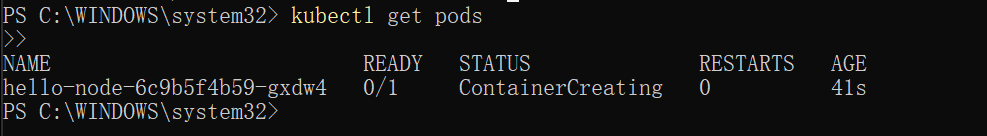
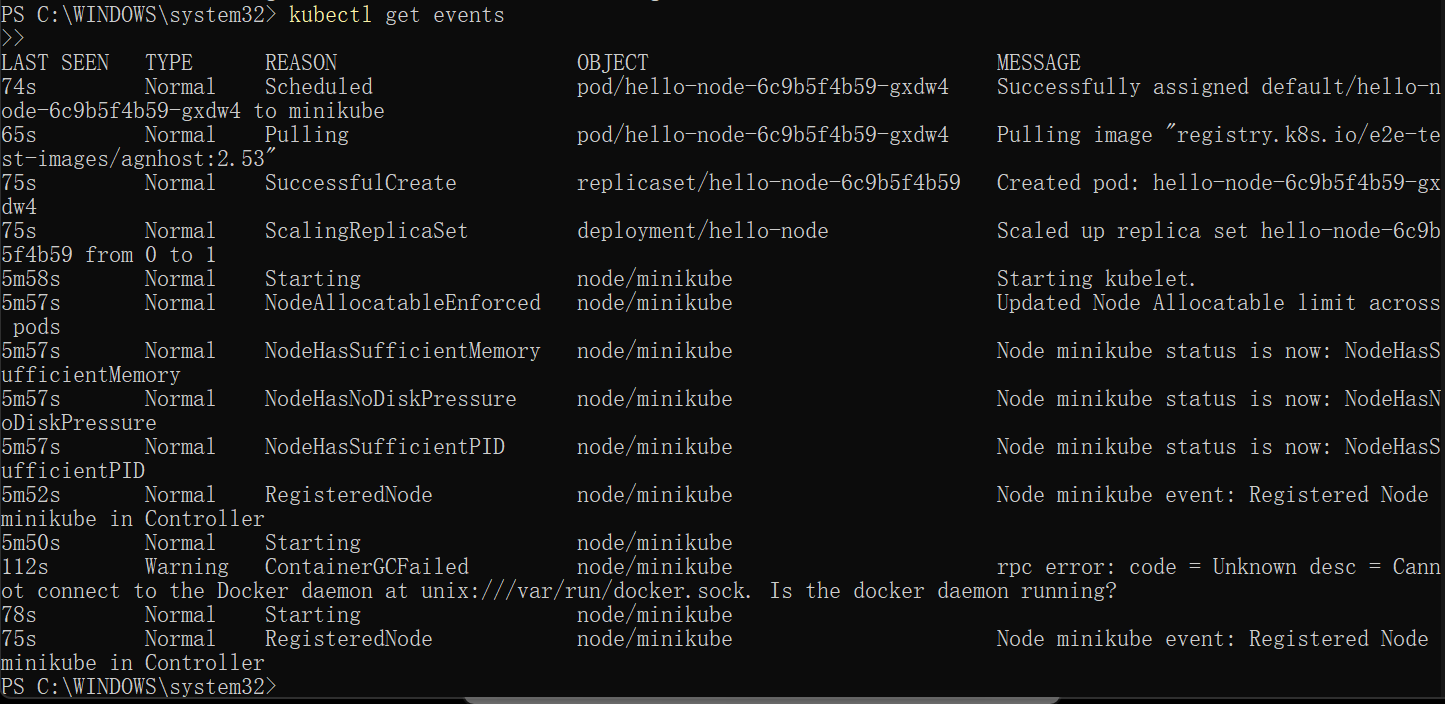
1：

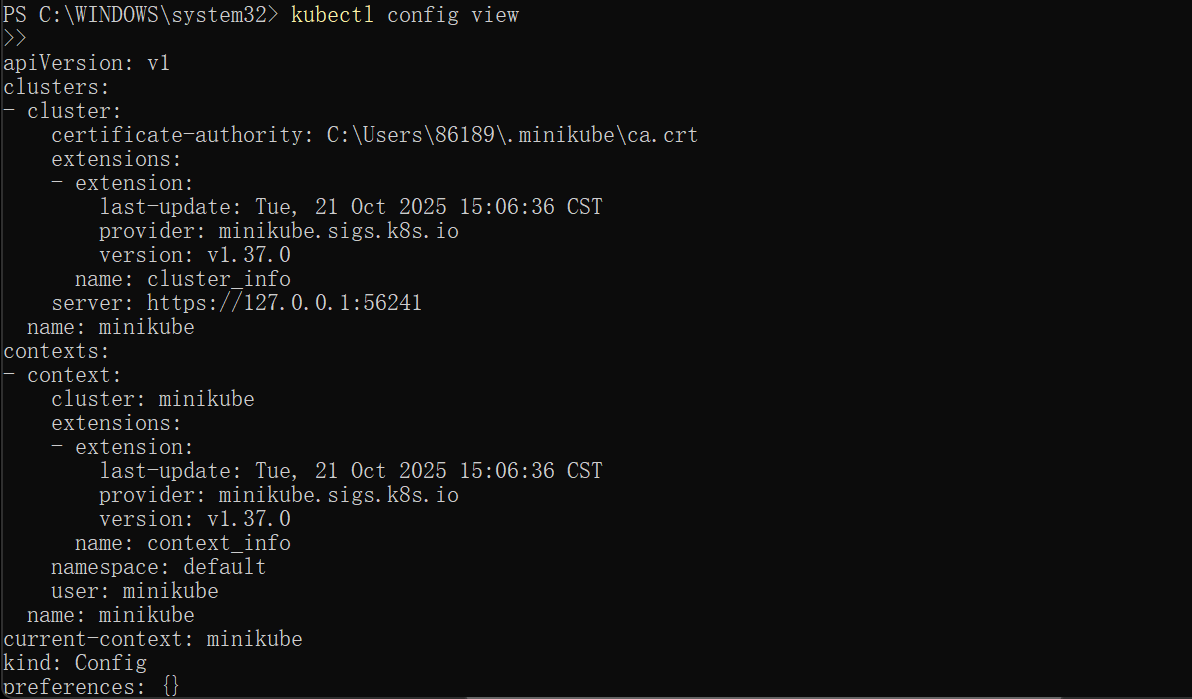
This tutorial shows you how to run a sample app on Kubernetes using minikube. The tutorial provides a container image that uses NGINX to echo back all the requests





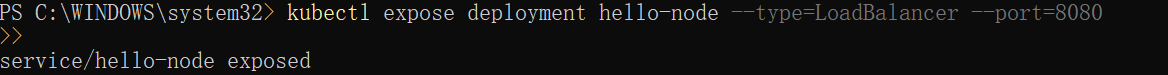


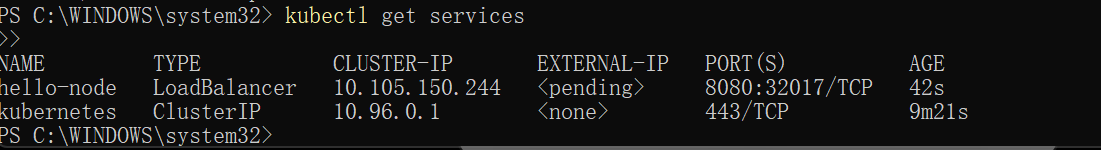


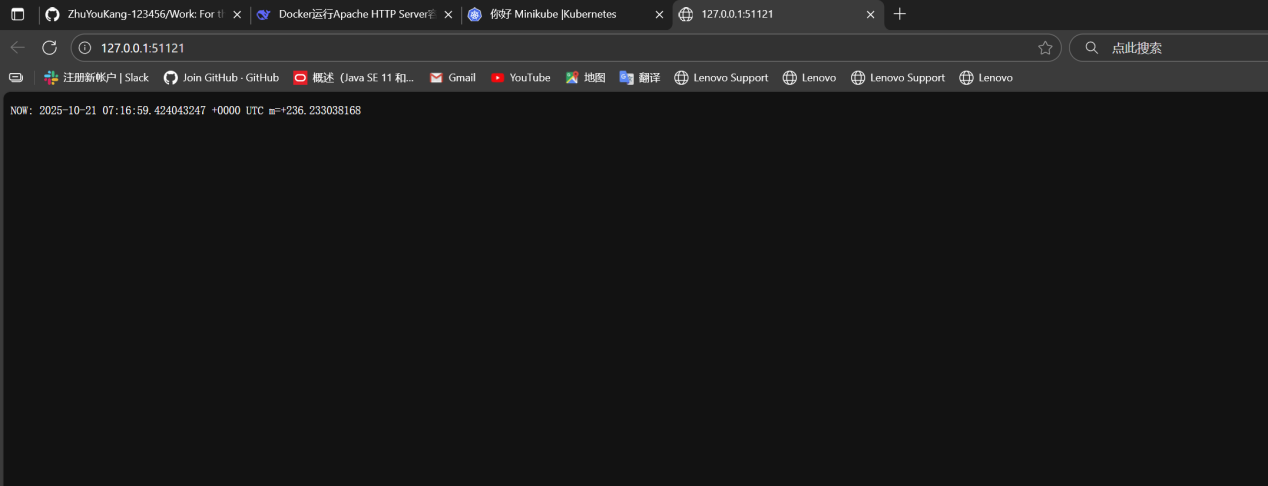


Open the Kubernetes dashboard. You can do this two different ways:



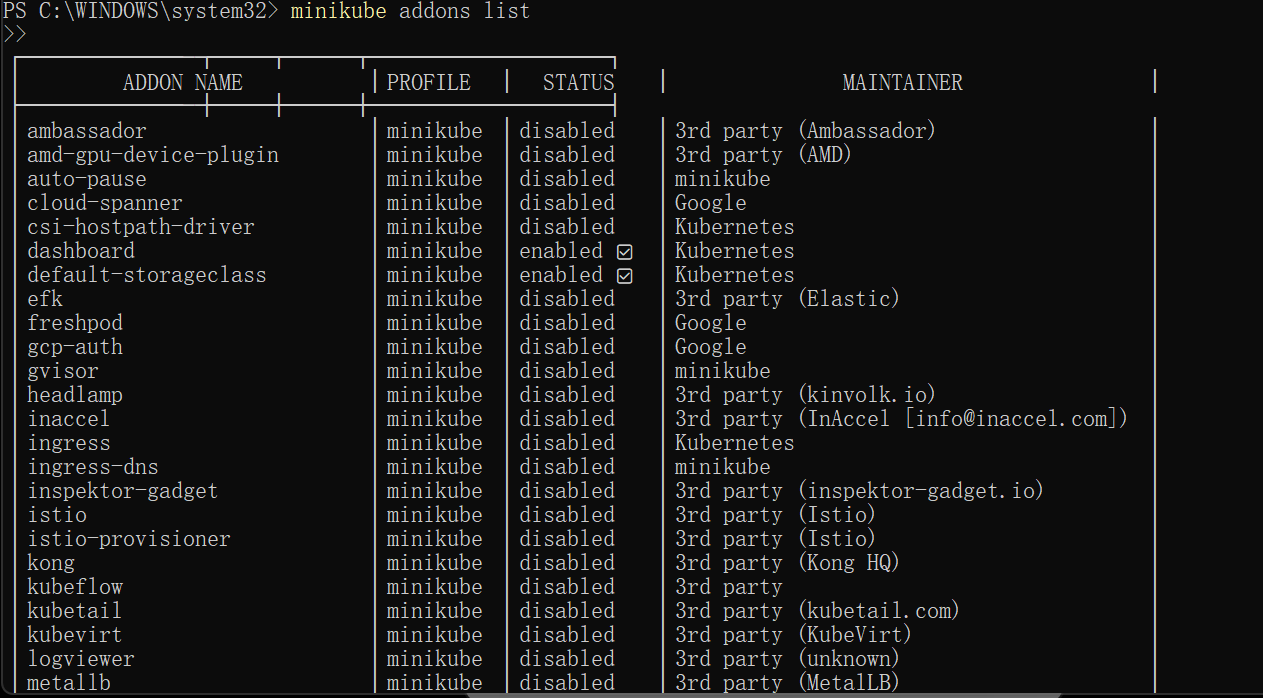




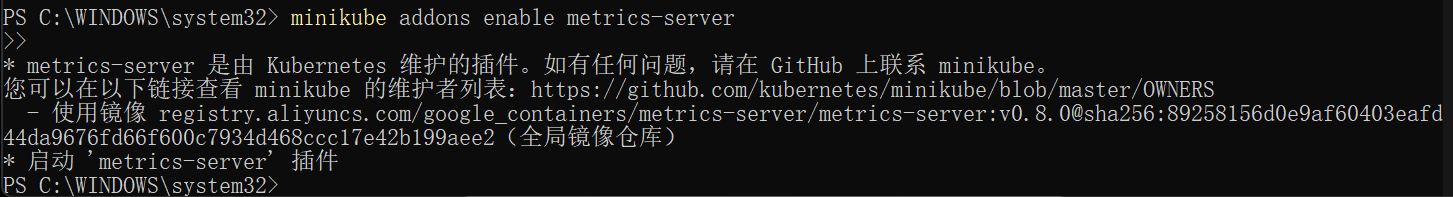


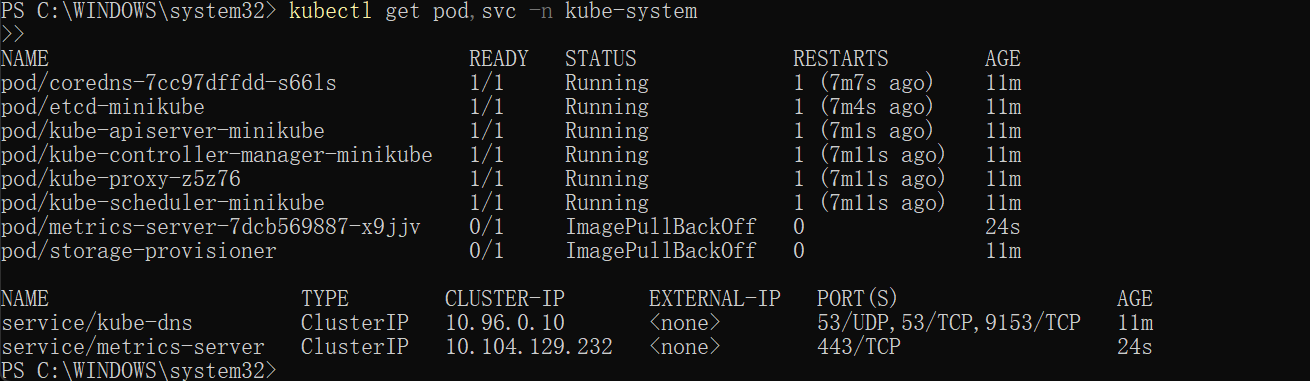


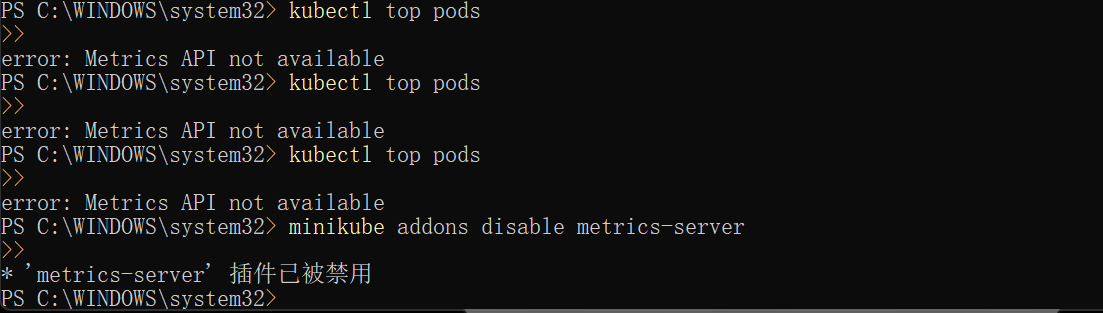
A Kubernetes [Pod](https://kubernetes.io/docs/concepts/workloads/pods/) is a group of one or more Containers, tied together for the purposes of administration and networking. The Pod in this tutorial has only one Container. A Kubernetes [Deployment](https://kubernetes.io/docs/concepts/workloads/controllers/deployment/) checks on the health of your Pod and restarts the Pod's Container if it terminates. Deployments are the recommended way to manage the creation and scaling of Pods.



By default, the Pod is only accessible by its internal IP address within the Kubernetes cluster. To make the hello-node Container accessible from outside the Kubernetes virtual network, you have to expose the Pod as a Kubernetes [Service](https://kubernetes.io/docs/concepts/services-networking/service/).

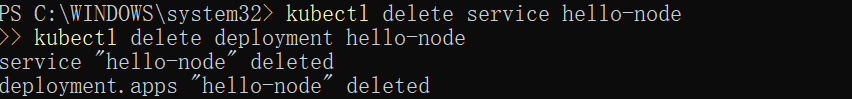


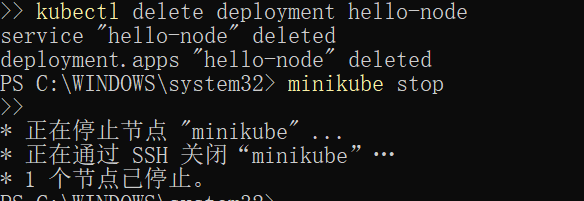




The minikube tool includes a set of built-in [addons](https://kubernetes.io/docs/concepts/cluster-administration/addons/" \o "" \t "https://kubernetes.io/docs/tutorials/hello-minikube/_blank) that can be enabled, disabled and opened in the local Kubernetes environment.

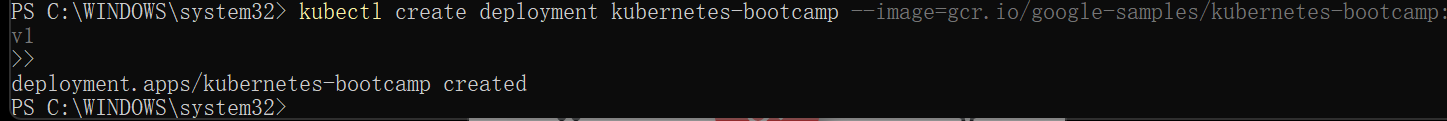
Now you can clean up the resources you created in your cluster:





2:

In a pre-orchestration world, installation scripts would often be used to start applications, but they did not allow recovery from machine failure. By both creating your application instances and keeping them running across Nodes, Kubernetes Deployments provide a fundamentally different approach to application management



The common format of a kubectl command is: kubectl action resource.

This performs the specified action (like create, describe or delete) on the specified resource (like node or deployment. You can use --help after the subcommand to get additional info about possible parameters (for example: kubectl get nodes --help).

Check that kubectl is configured to talk to your cluster, by running the kubectl version command.

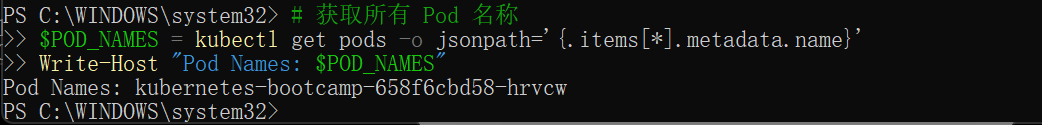
Check that kubectl is installed and that you can see both the client and the server versions.

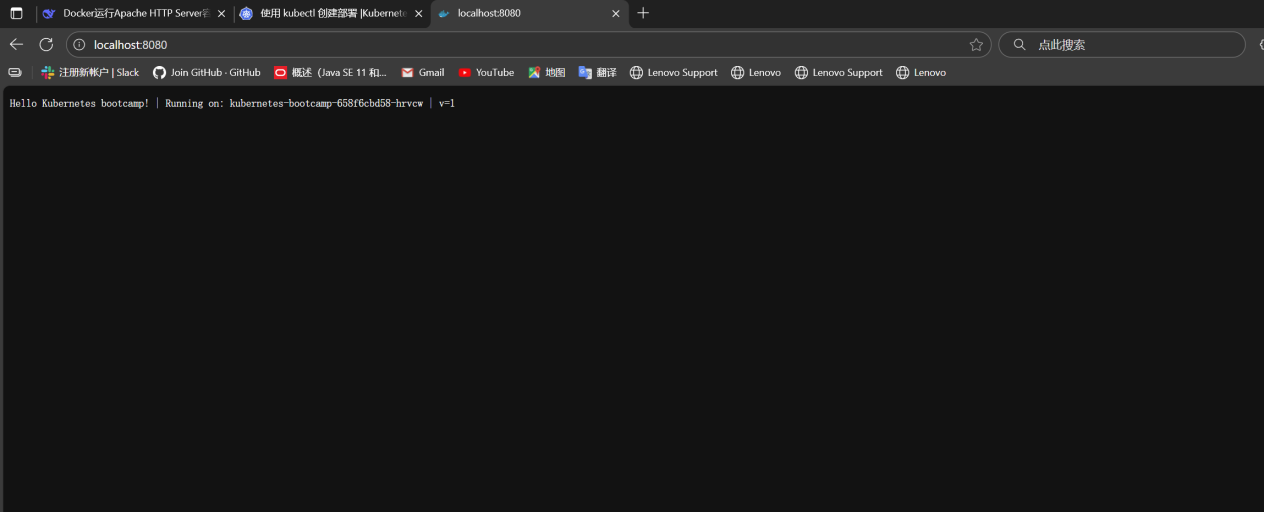
To view the nodes in the cluster, run the kubectl get nodes command.

You see the available nodes. Later, Kubernetes will choose where to deploy our application based on Node available resources.



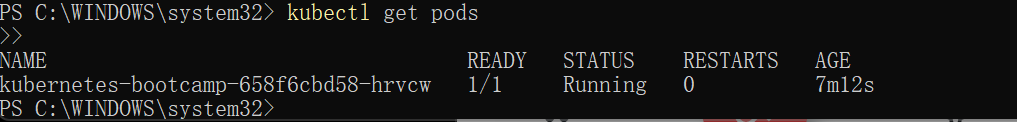
The kubectl proxy command can create a proxy that will forward communications into the cluster-wide, private network. The proxy can be terminated by pressing control-C and won't show any output while it's running.



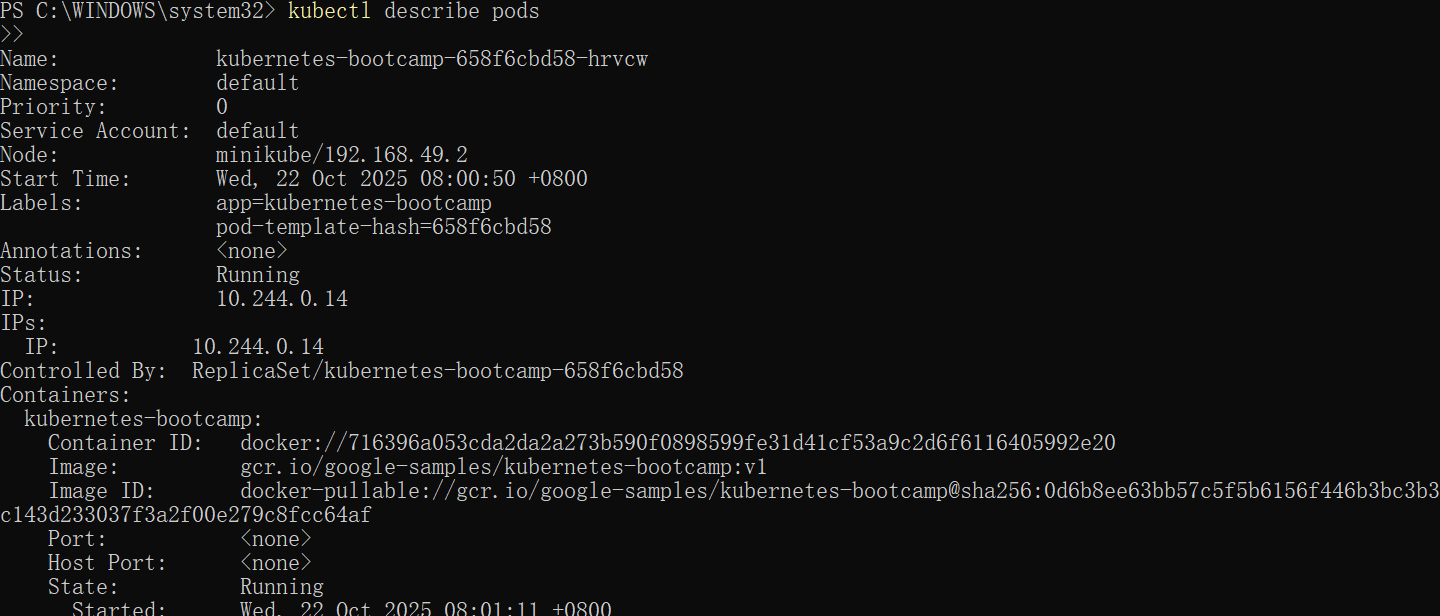


3:

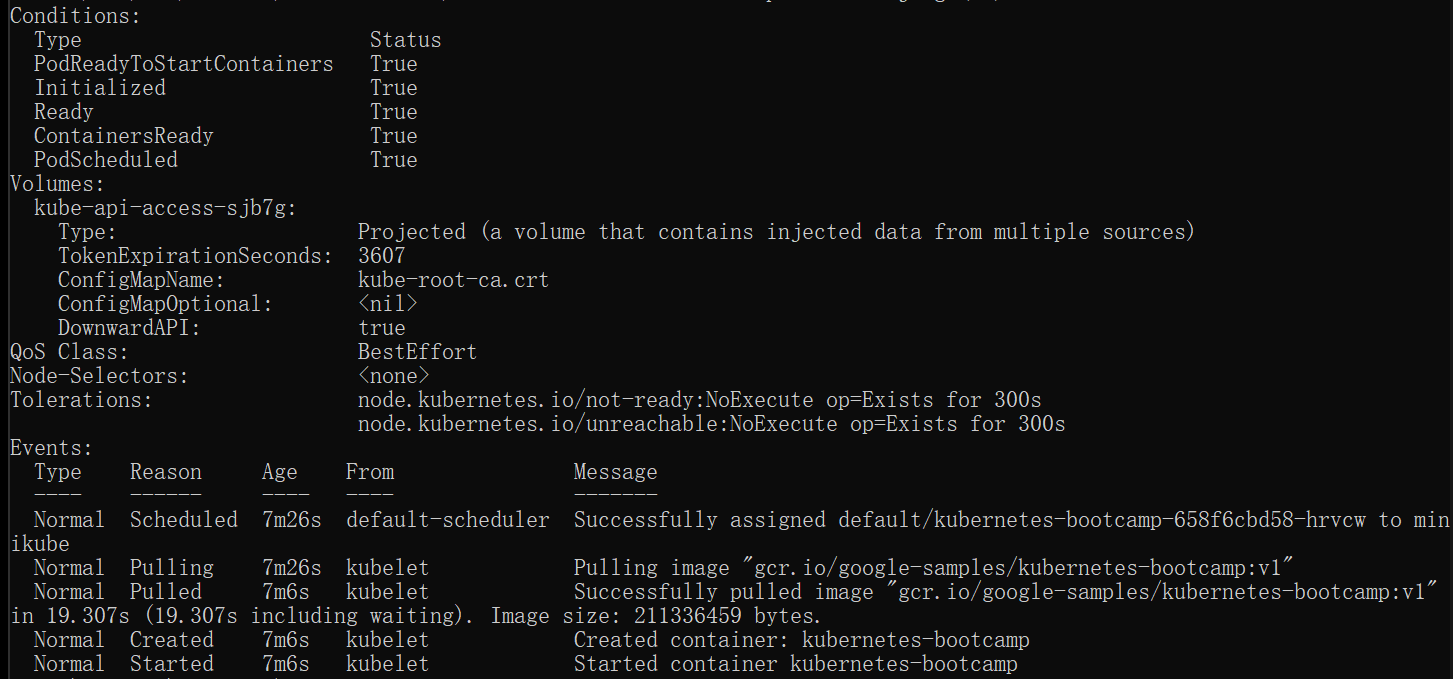
A Pod models an application-specific "logical host" and can contain different application containers which are relatively tightly coupled. For example, a Pod might include both the container with your Node.js app as well as a different container that feeds the data to be published by the Node.js webserver. The containers in a Pod share an IP Address and port space, are always co-located and co-scheduled, and run in a shared context on the same Node.

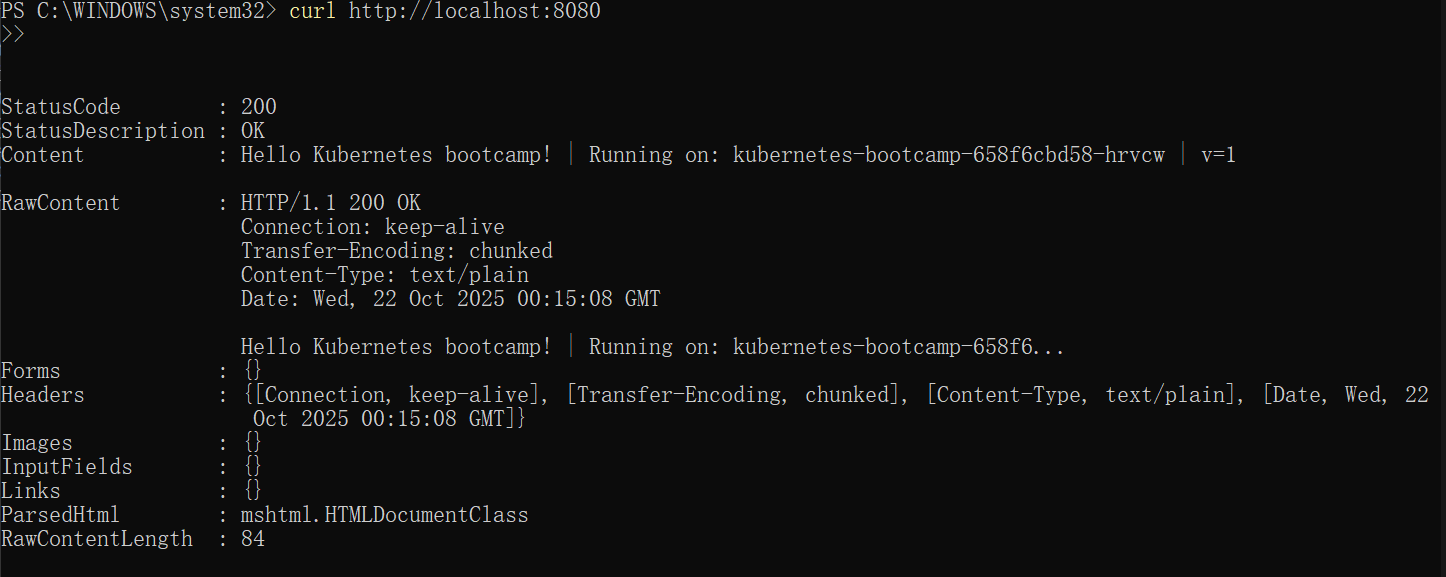


A Pod always runs on a Node. A Node is a worker machine in Kubernetes and may be either a virtual or a physical machine, depending on the cluster. Each Node is managed by the control plane. A Node can have multiple pods, and the Kubernetes control plane automatically handles scheduling the pods across the Nodes in the cluster. The control plane's automatic scheduling takes into account the available resources on each Node.

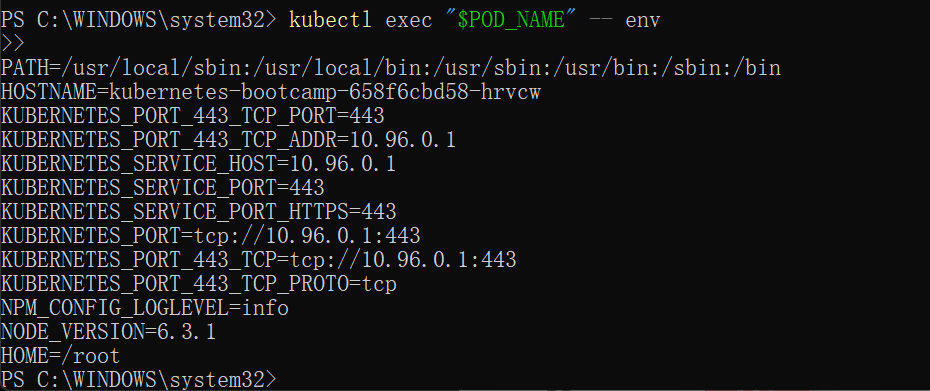


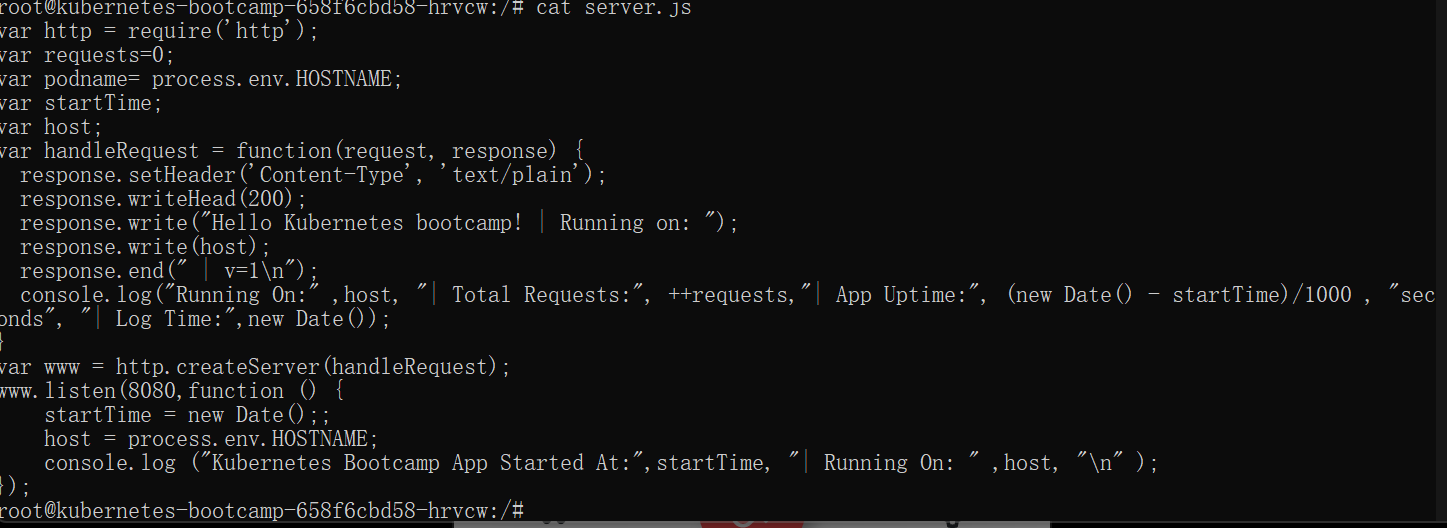
 verify that the application we deployed in the previous scenario is running. We'll use the kubectl get command and look for existing Pods:

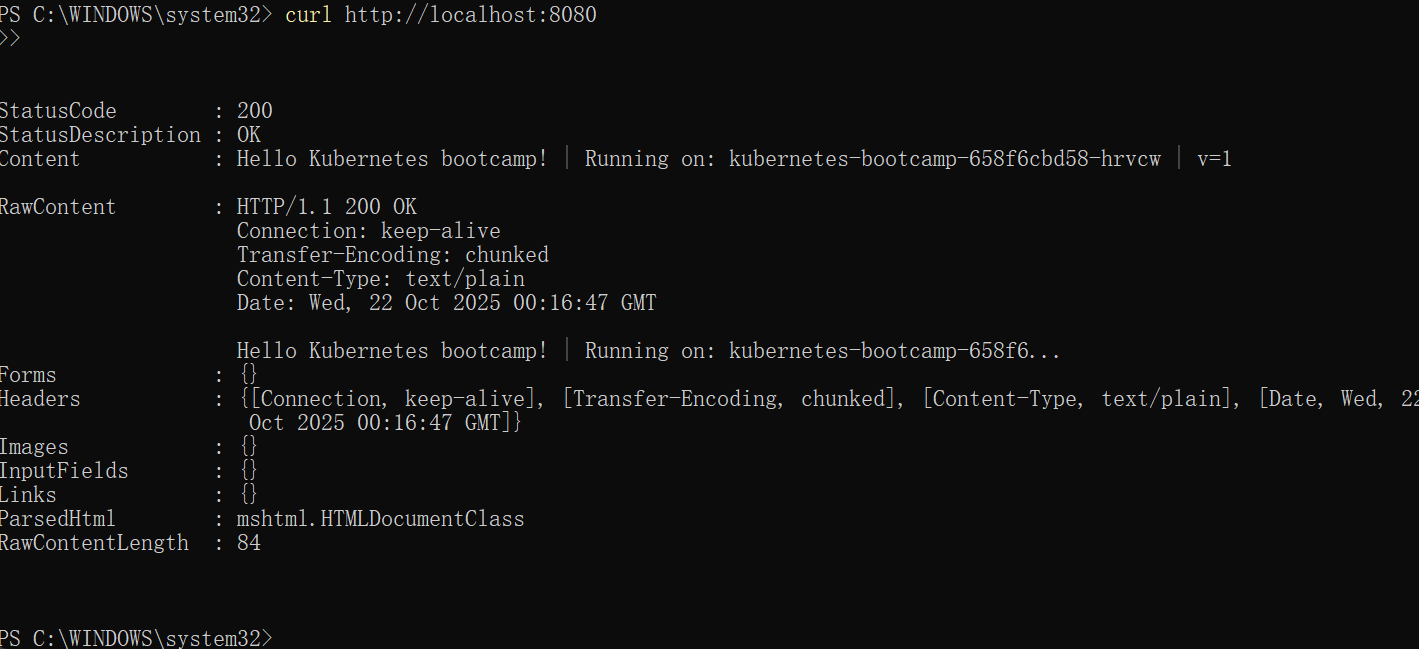




We can execute commands directly on the container once the Pod is up and running. For this, we use the exec subcommand and use the name of the Pod as a parameter. Let’s list the environment variables:

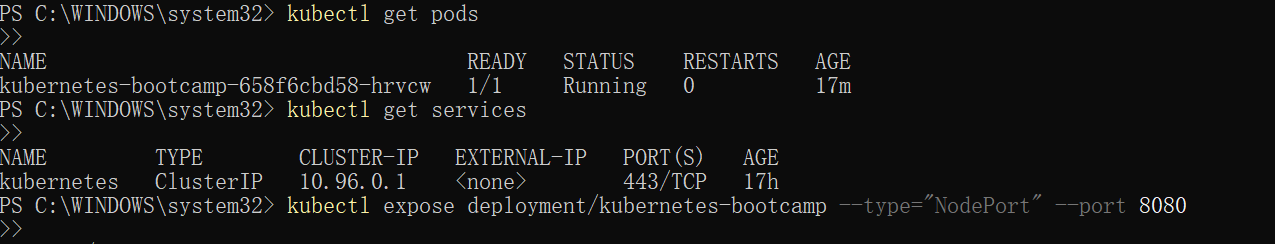




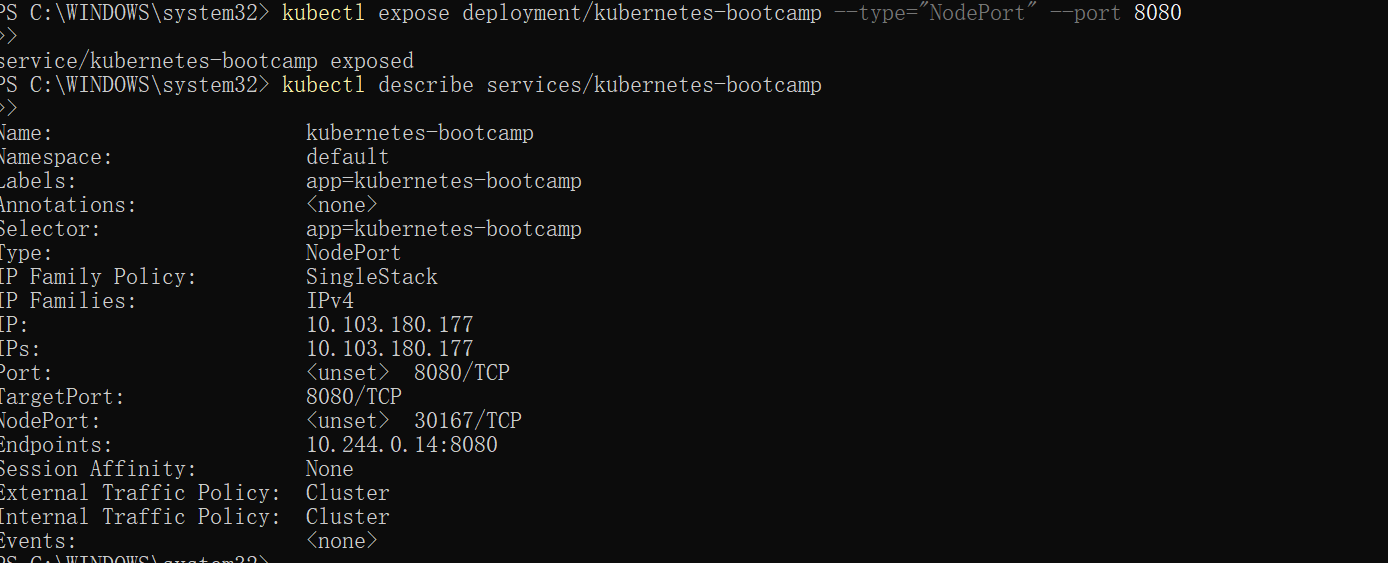


4:

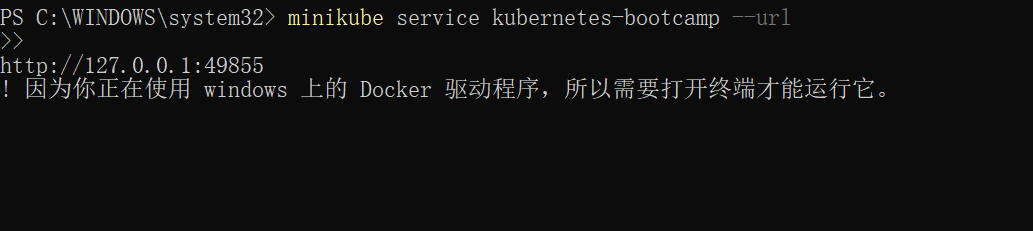
Kubernetes [Pods](https://kubernetes.io/docs/concepts/workloads/pods/) are mortal. Pods have a [lifecycle](https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle/). When a worker node dies, the Pods running on the Node are also lost. A [Replicaset](https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/) might then dynamically drive the cluster back to the desired state via the creation of new Pods to keep your application running. As another example, consider an image-processing backend with 3 replicas. Those replicas are exchangeable; the front-end system should not care about backend replicas or even if a Pod is lost and recreated. That said, each Pod in a Kubernetes cluster has a unique IP address, even Pods on the same Node, so there needs to be a way of automatically reconciling changes among Pods so that your applications continue to function.

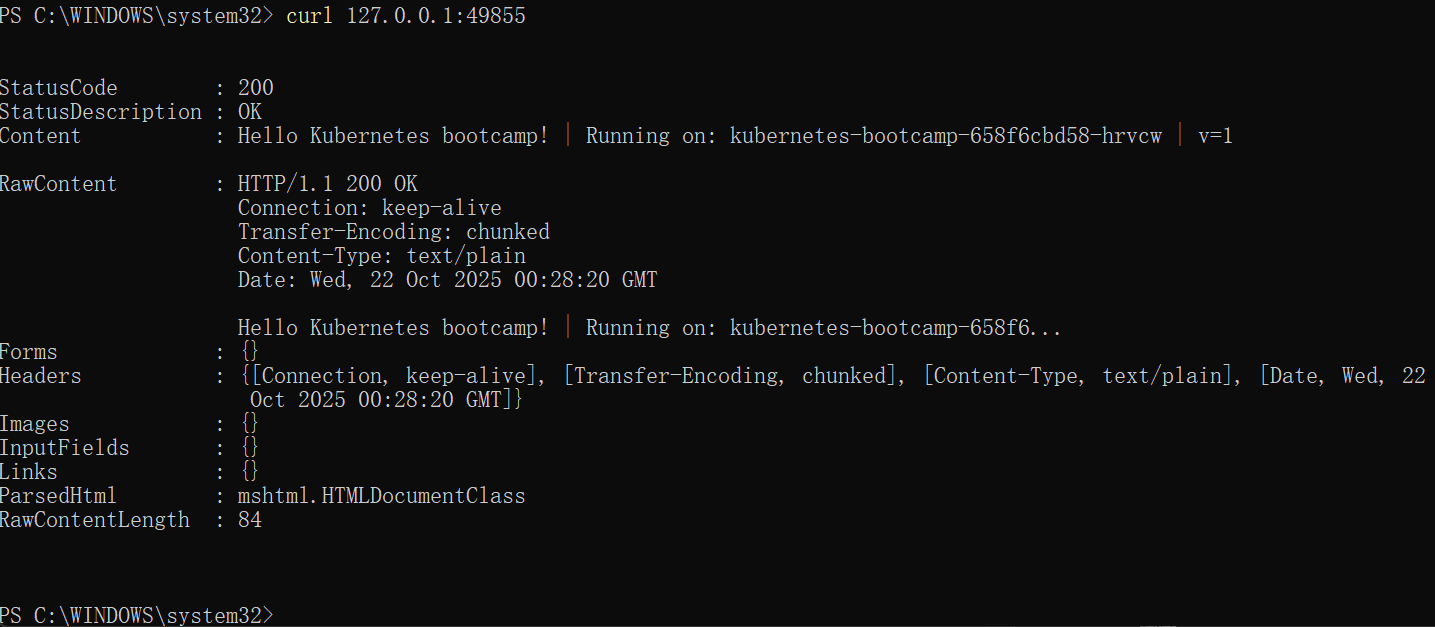


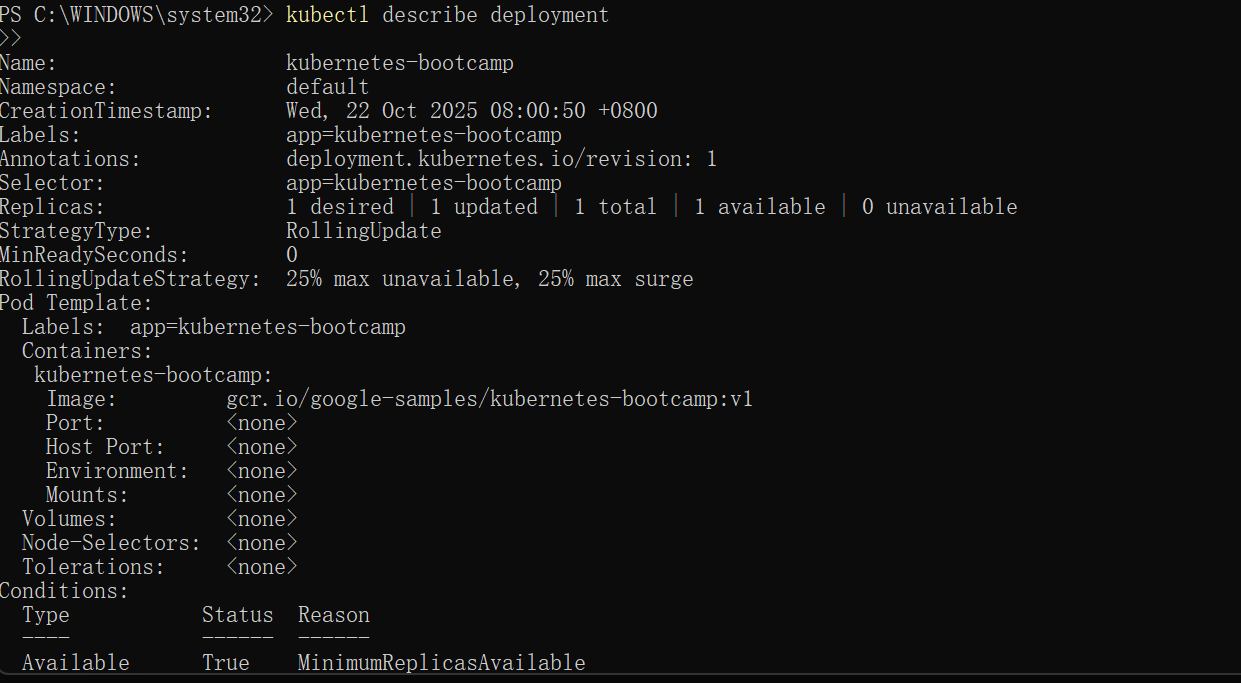
A Service routes traffic across a set of Pods. Services are the abstraction that allows pods to die and replicate in Kubernetes without impacting your application. Discovery and routing among dependent Pods (such as the frontend and backend components in an application) are handled by Kubernetes Services.



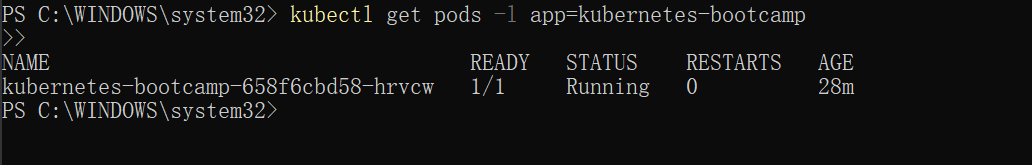
To find out what port was opened externally (for the type: NodePort Service) we’ll run the describe service subcommand:

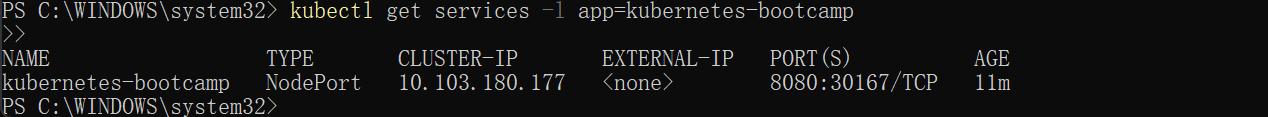


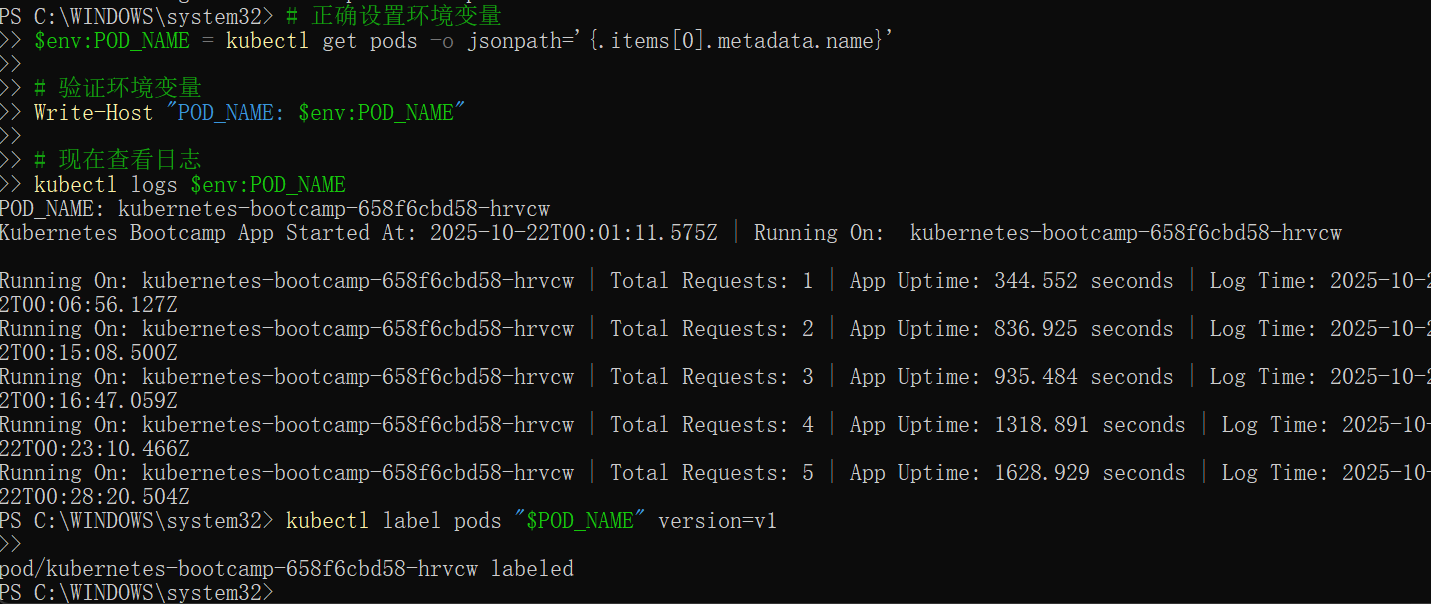


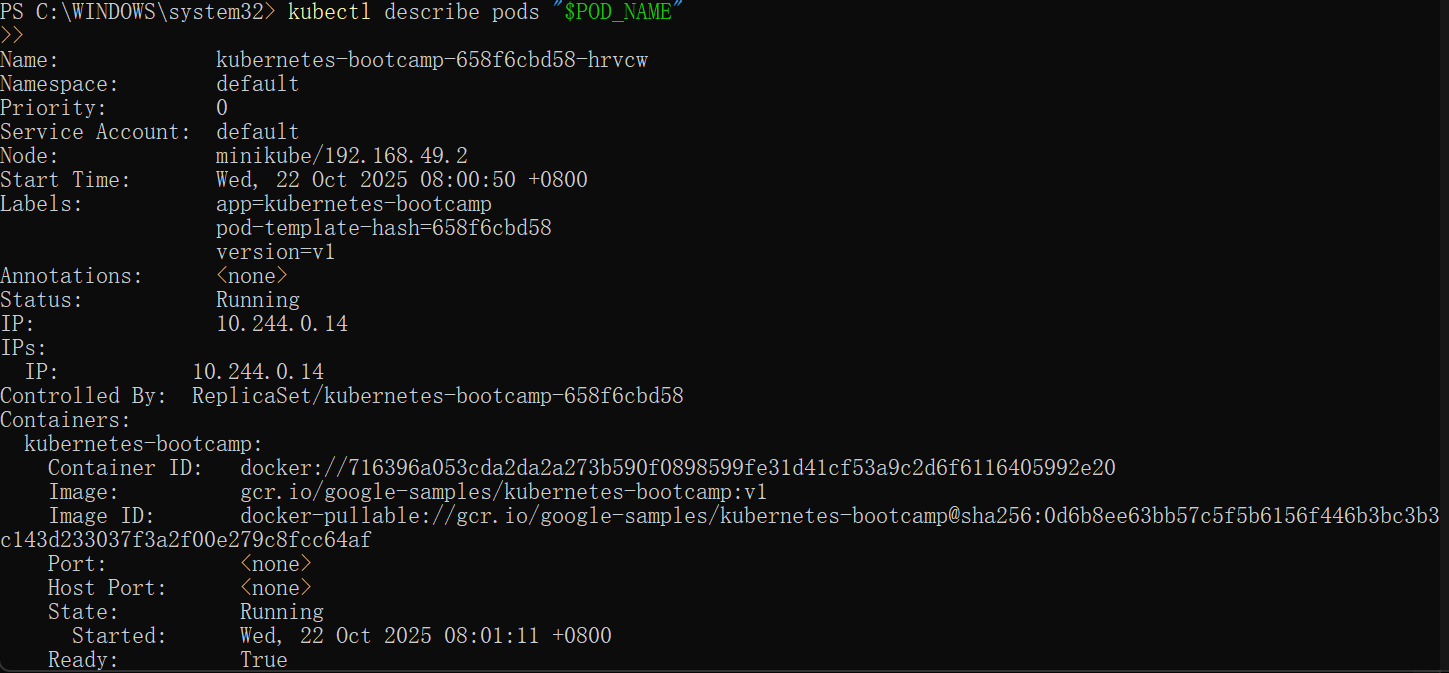


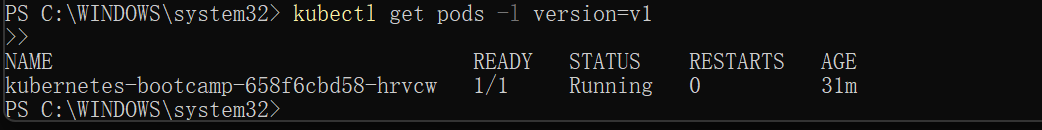
To apply a new label we use the label subcommand followed by the object type, object name and the new label:

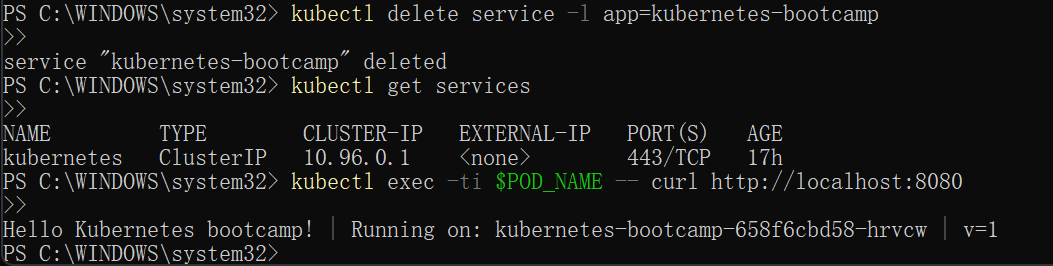






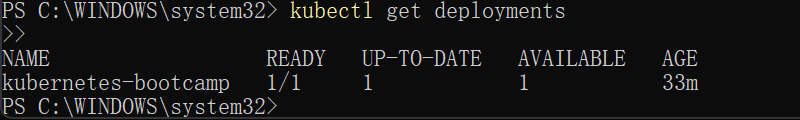


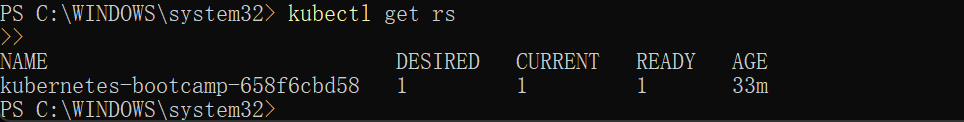


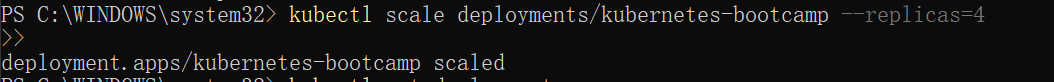
This proves that the application is not reachable anymore from outside of the cluster. You can confirm that the app is still running with a curl from inside the pod:

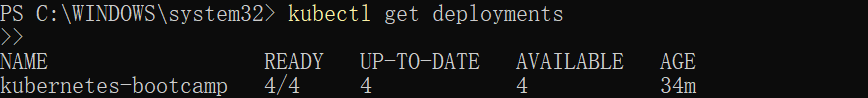
5:

Scaling out a Deployment will ensure new Pods are created and scheduled to Nodes with available resources. Scaling will increase the number of Pods to the new desired state. Kubernetes also supports [autoscaling](https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/) of Pods, but it is outside of the scope of this tutorial. Scaling to zero is also possible, and it will terminate all Pods of the specified Deployment.

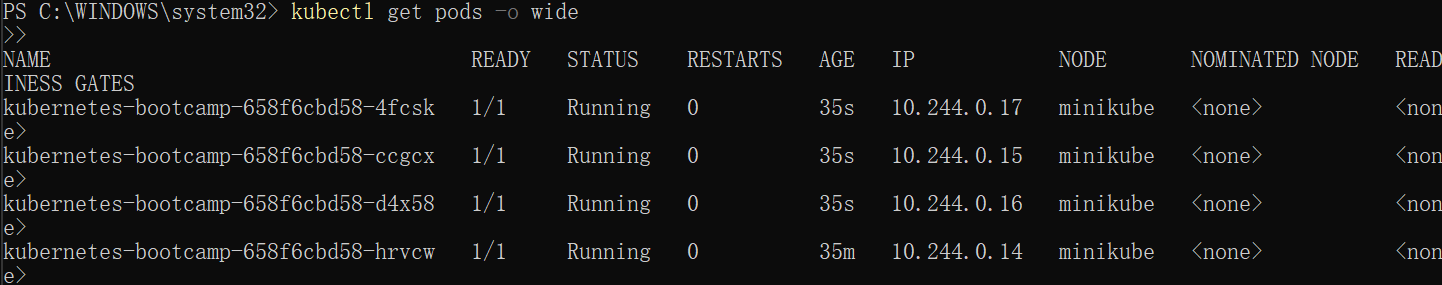


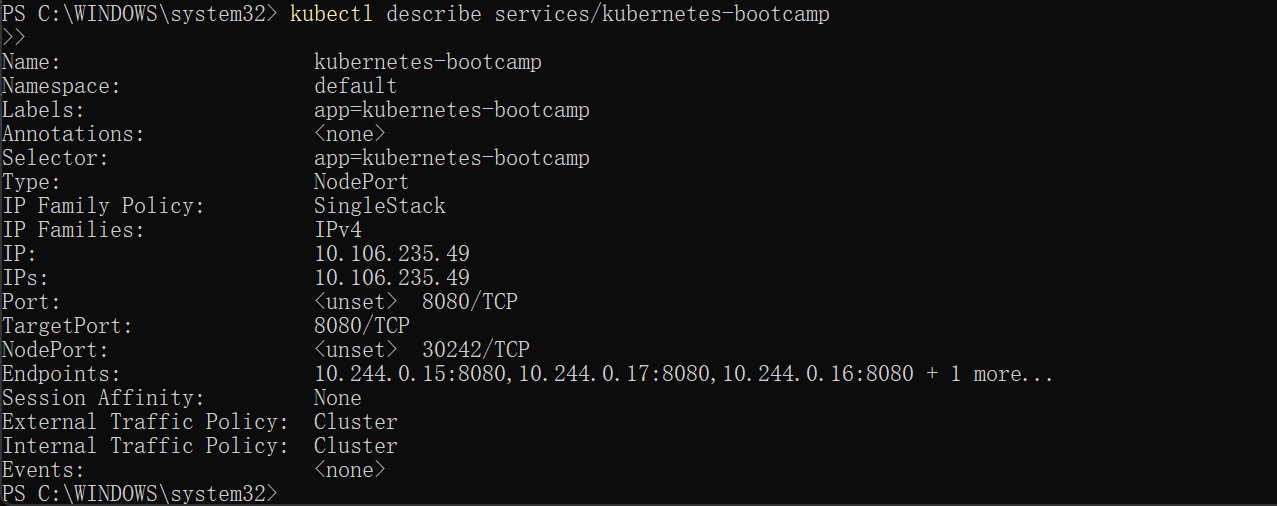




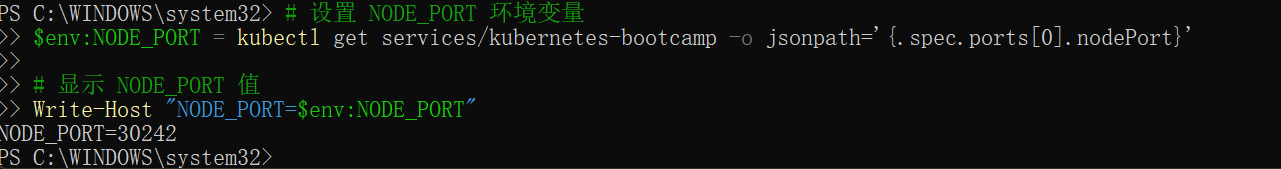


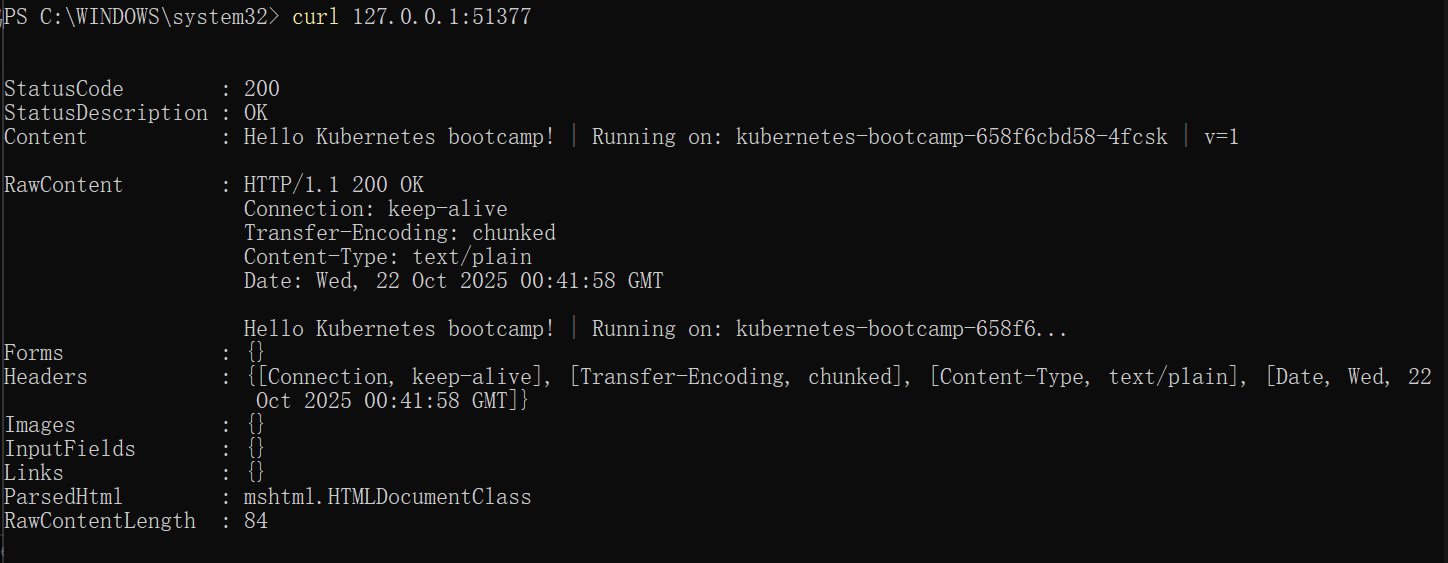
The change was applied, and we have 4 instances of the application available. Next, let’s check if the number of Pods changed:



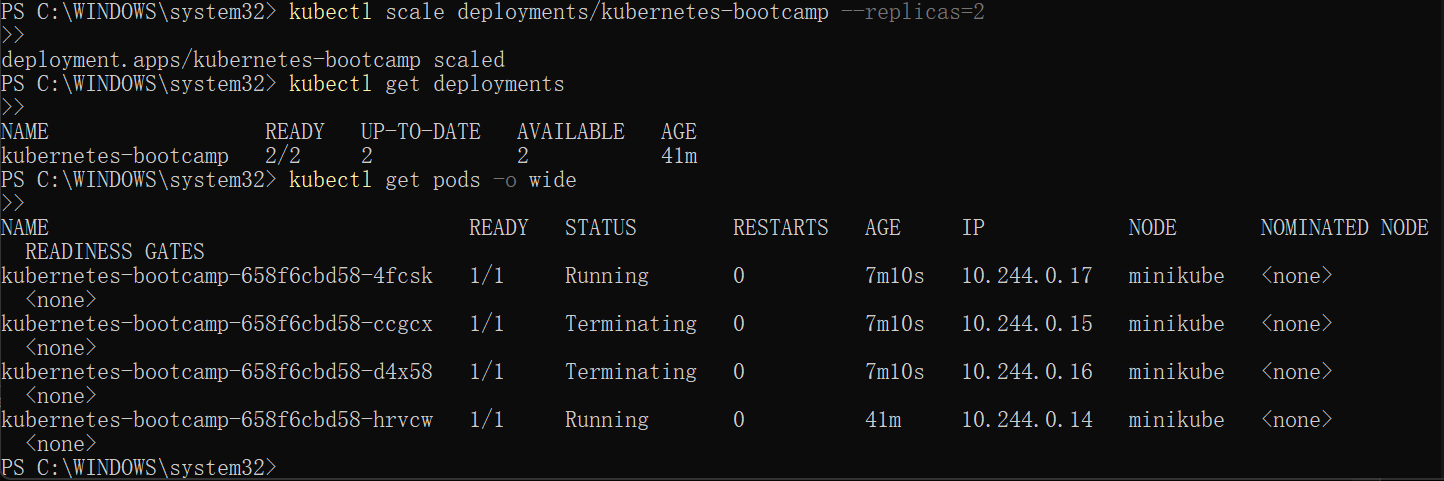


Create an environment variable called NODE\_PORT that has a value as the Node port:





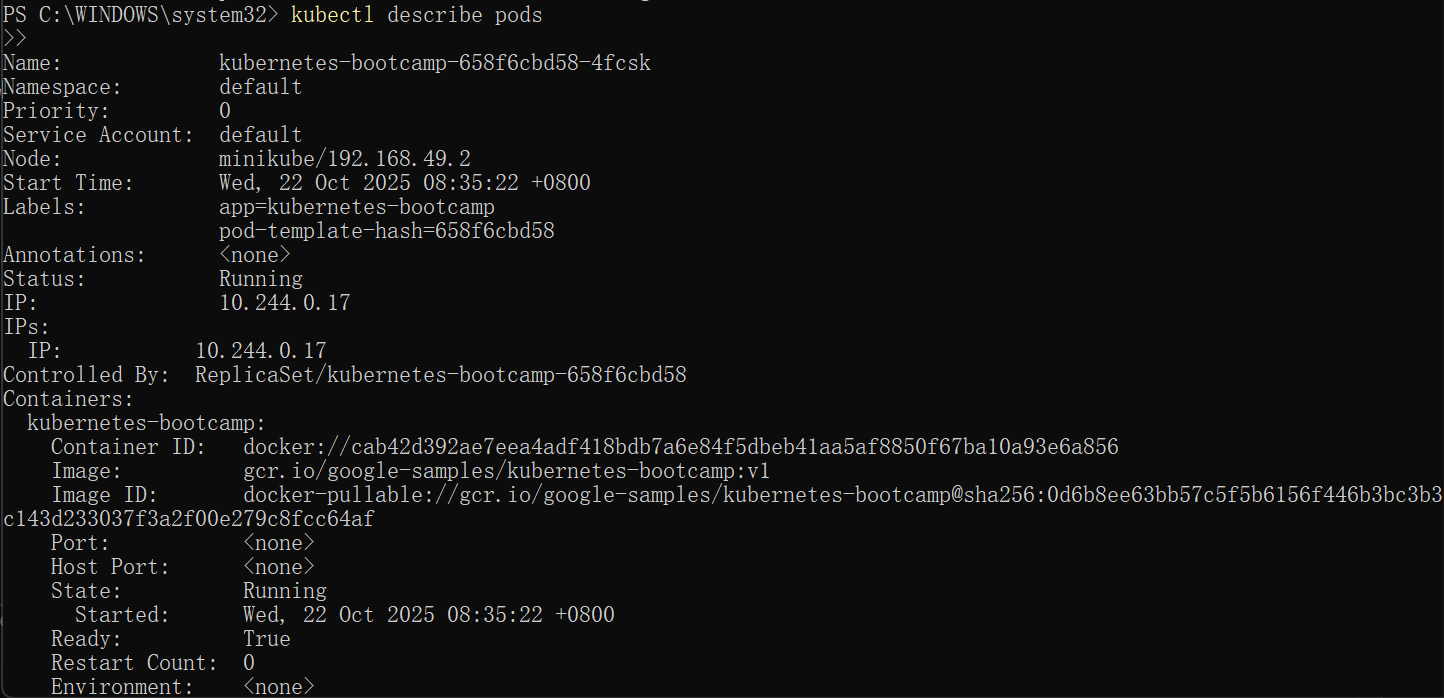
List the Deployments to check if the change was applied with the get deployments subcommand:

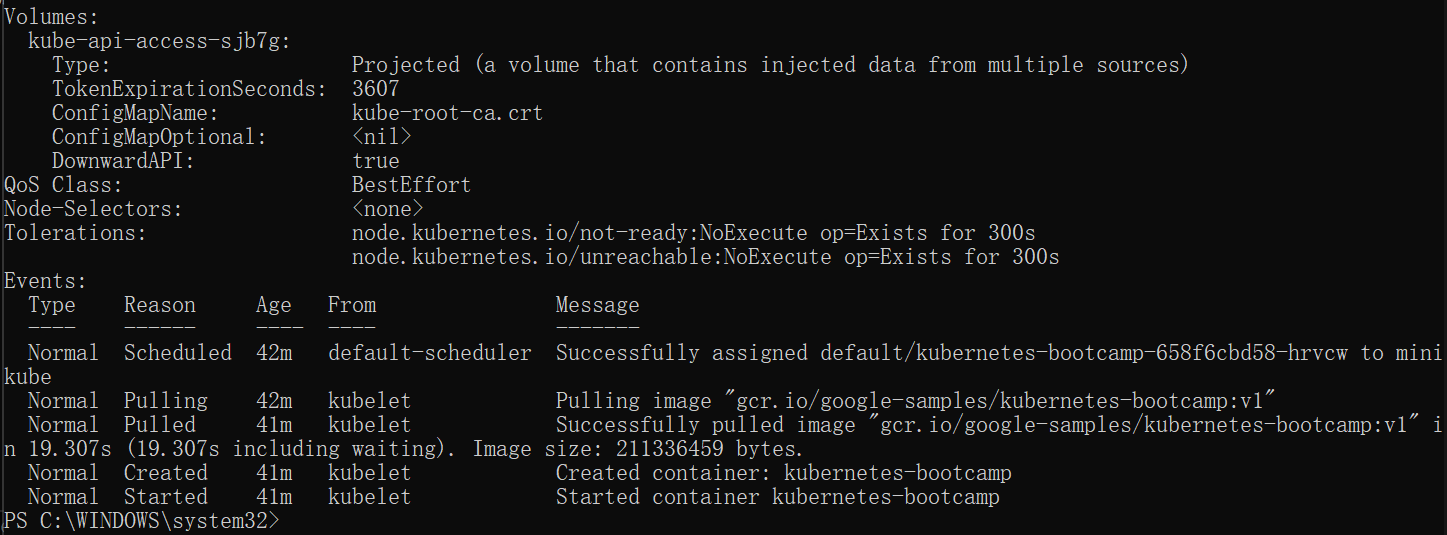


6:

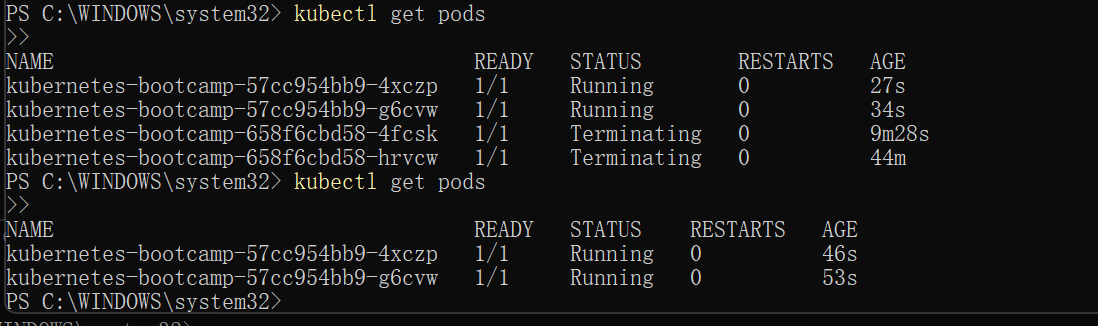
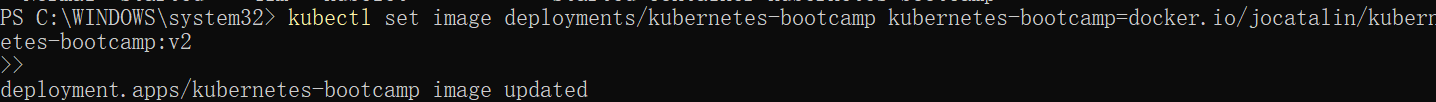
Users expect applications to be available all the time, and developers are expected to deploy new versions of them several times a day. In Kubernetes this is done with rolling updates. A rolling update allows a Deployment update to take place with zero downtime. It does this by incrementally replacing the current Pods with new ones. The new Pods are scheduled on Nodes with available resources, and Kubernetes waits for those new Pods to start before removing the old Pods.



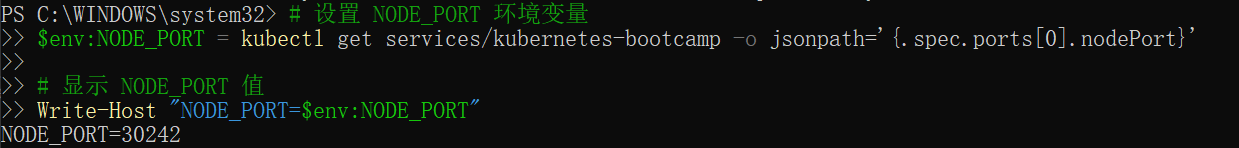


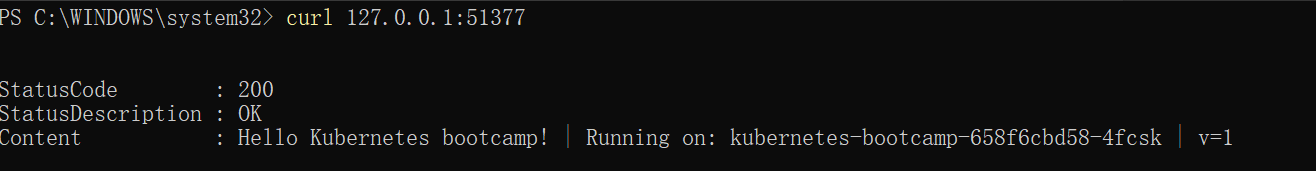


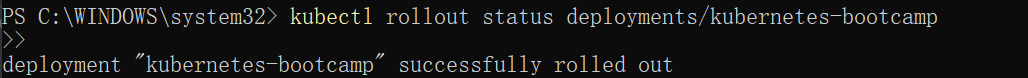
Similar to application Scaling, if a Deployment is exposed publicly, the Service will load-balance the traffic only to available Pods during the update. An available Pod is an instance that is available to the users of the application.



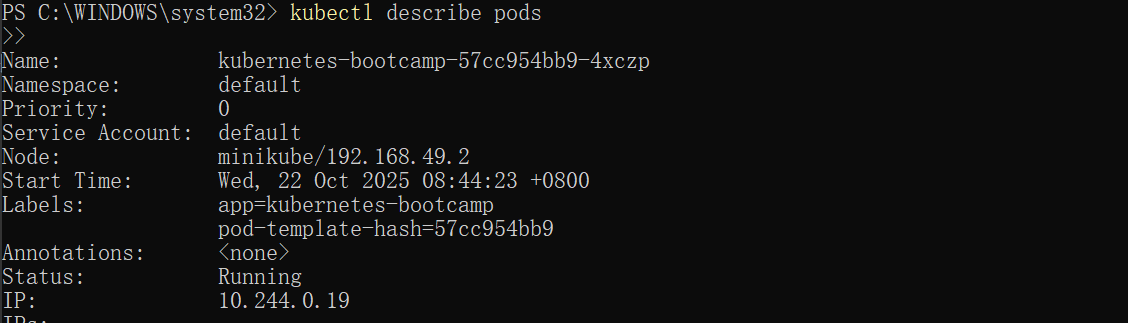
Create an environment variable called NODE\_PORT that has the value of the Node port assigned:

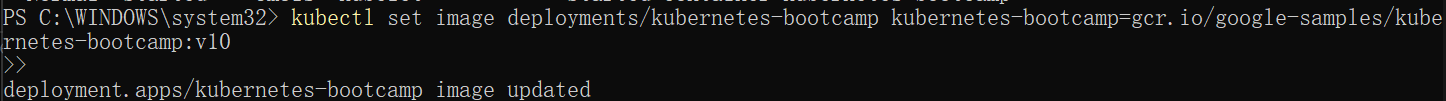


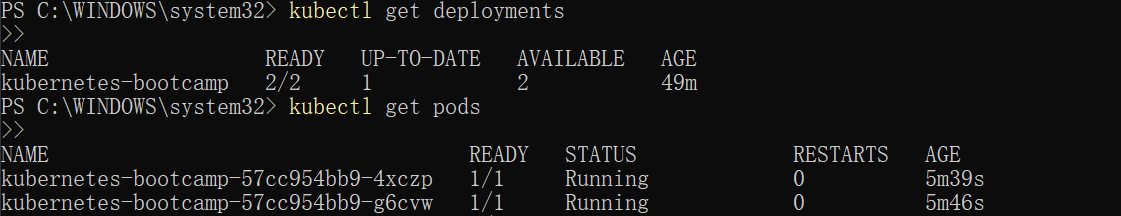




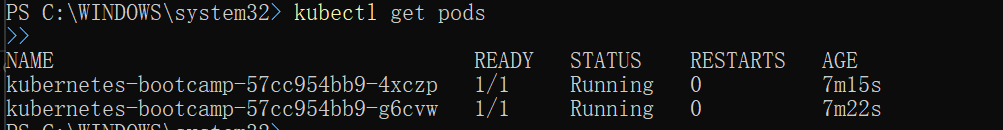
Notice that the output doesn't list the desired number of available Pods. Run the get pods subcommand to list all Pods:

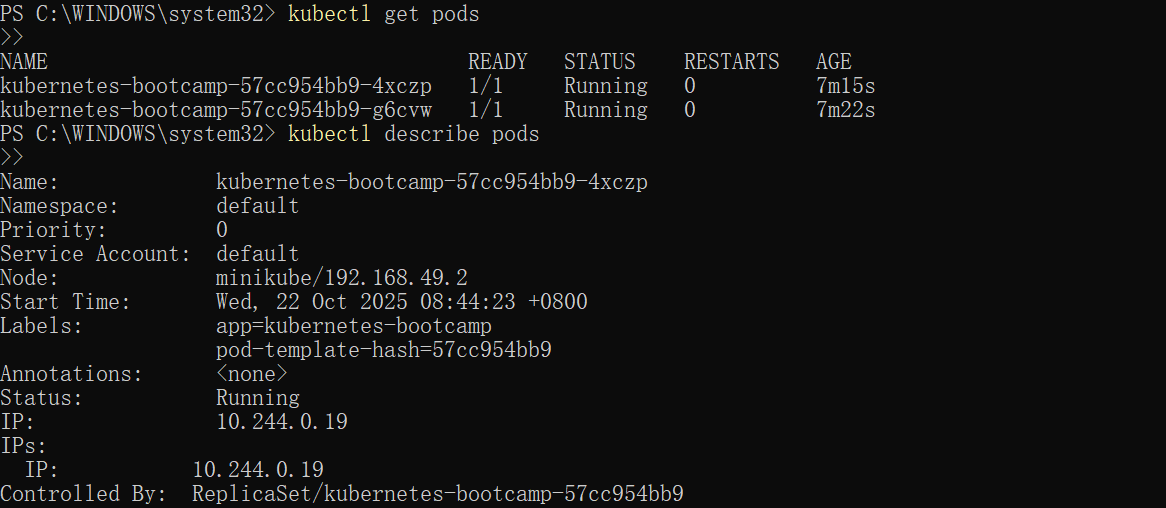






# 





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

Remember to clean up your local cluster.

