Kubernetes helps make sure those containerized applications run where and when you want, and helps them find the resources and tools they need to work.

Create a cluster

Using Minikube to create a cluster

A Kubernetes cluster consists of two types of resources:

1. Control Pane - manage the cluster
2. Nodes - a Node is a VM or physical computer that serves as a worker in a Kubernetes cluster

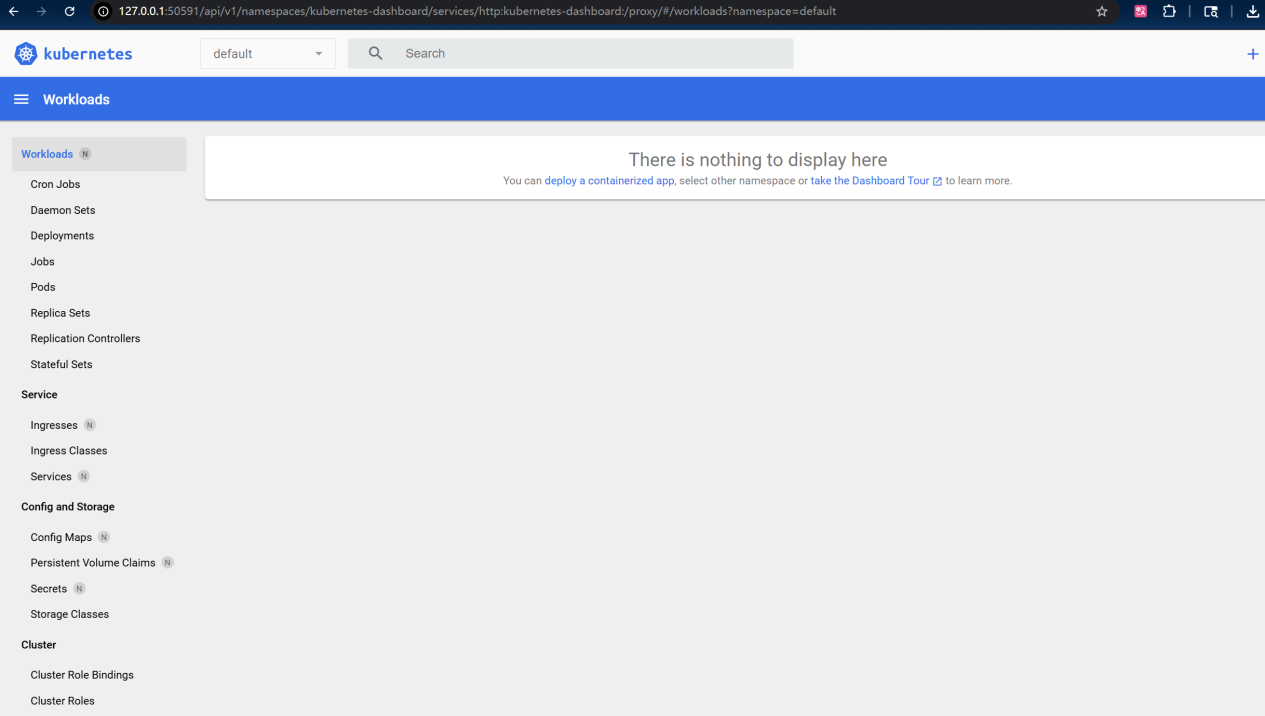
Create a minikube cluster

minikube start





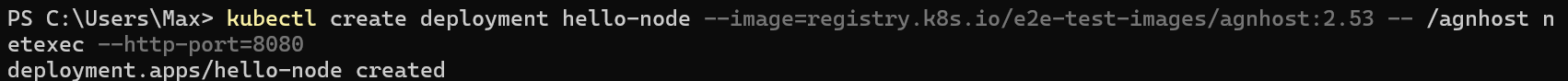
The dashboard command enables the dashboard add-on and opens the proxy in the default web browser. You can create Kubernetes resources on the dashboard such as Deployment and Service.



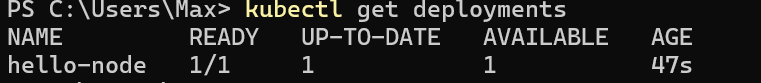
Create a deployment

A Kubernetes Pod 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 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.

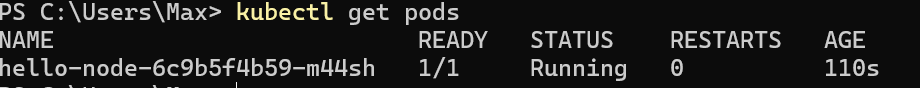
Use the kubectl create command to create a Deployment that manages a Pod. The Pod runs a Container based on the provided Docker image.



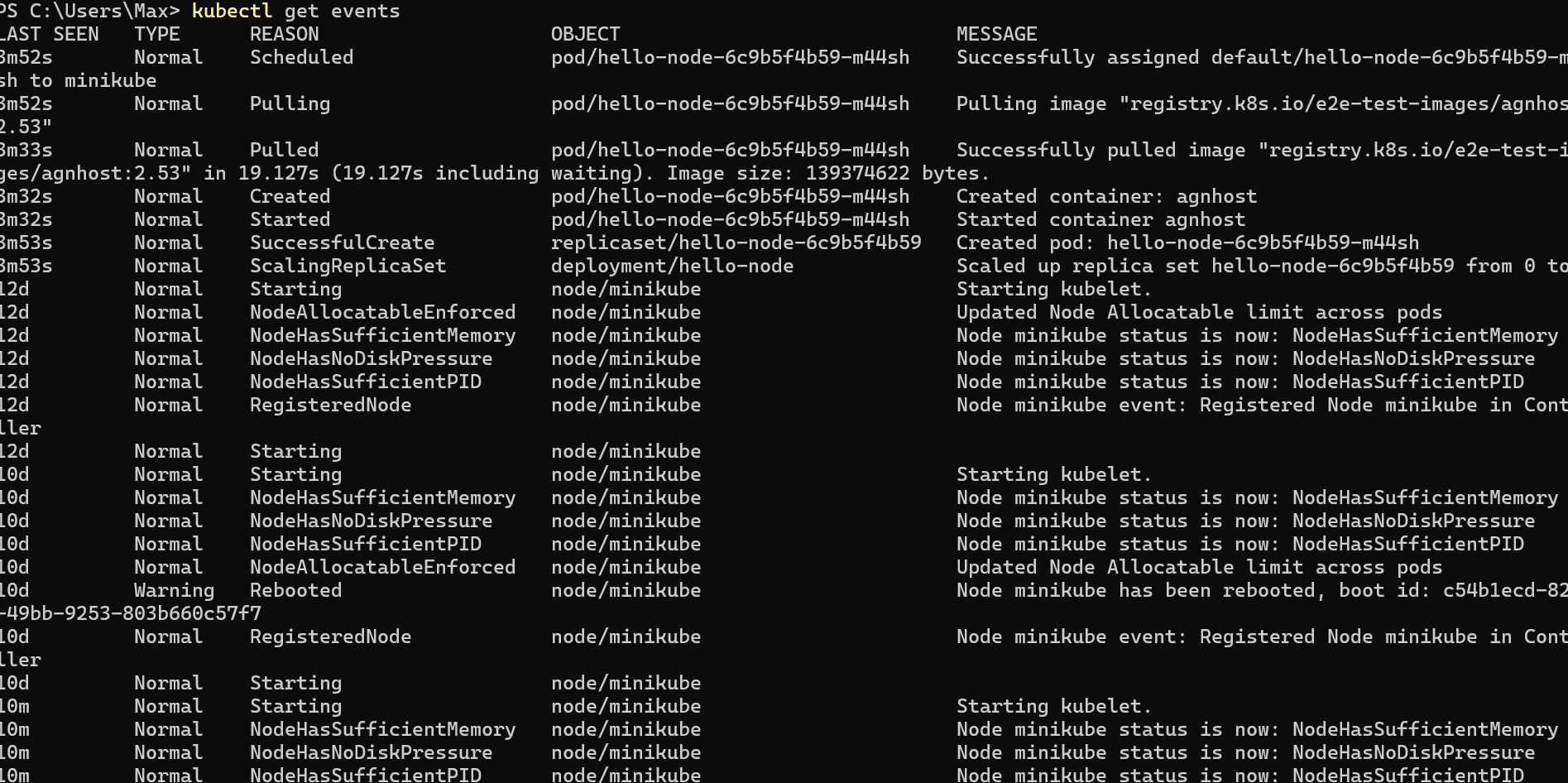
View the Deployment



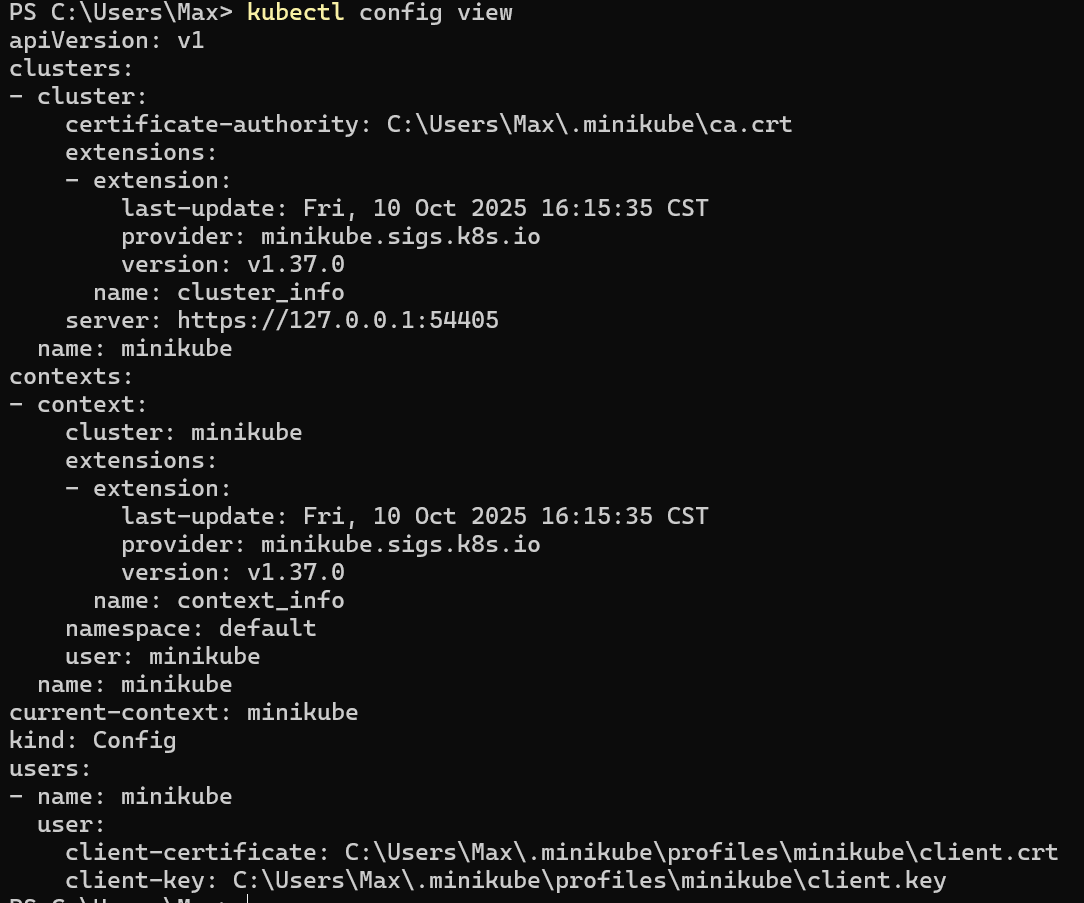
View the Pod



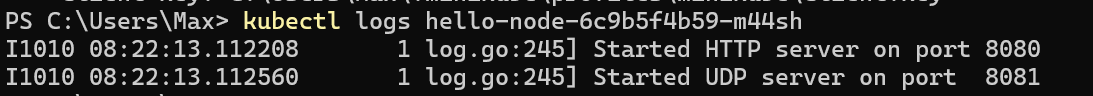
View cluster events



View the kubectl configuration



View application logs for a container in a pod



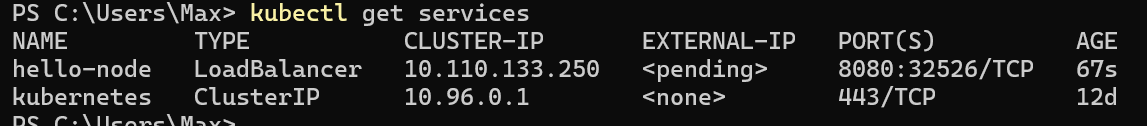
Create a Service

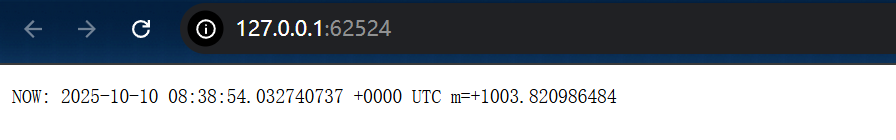
By default, the Pod is only accessible by its internal IP address within the Kubernetes cluster. To make the hello-node Container accessible from outside the Kubernetes virtual network, you have to expose the Pod as a Kubernetes Service.

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



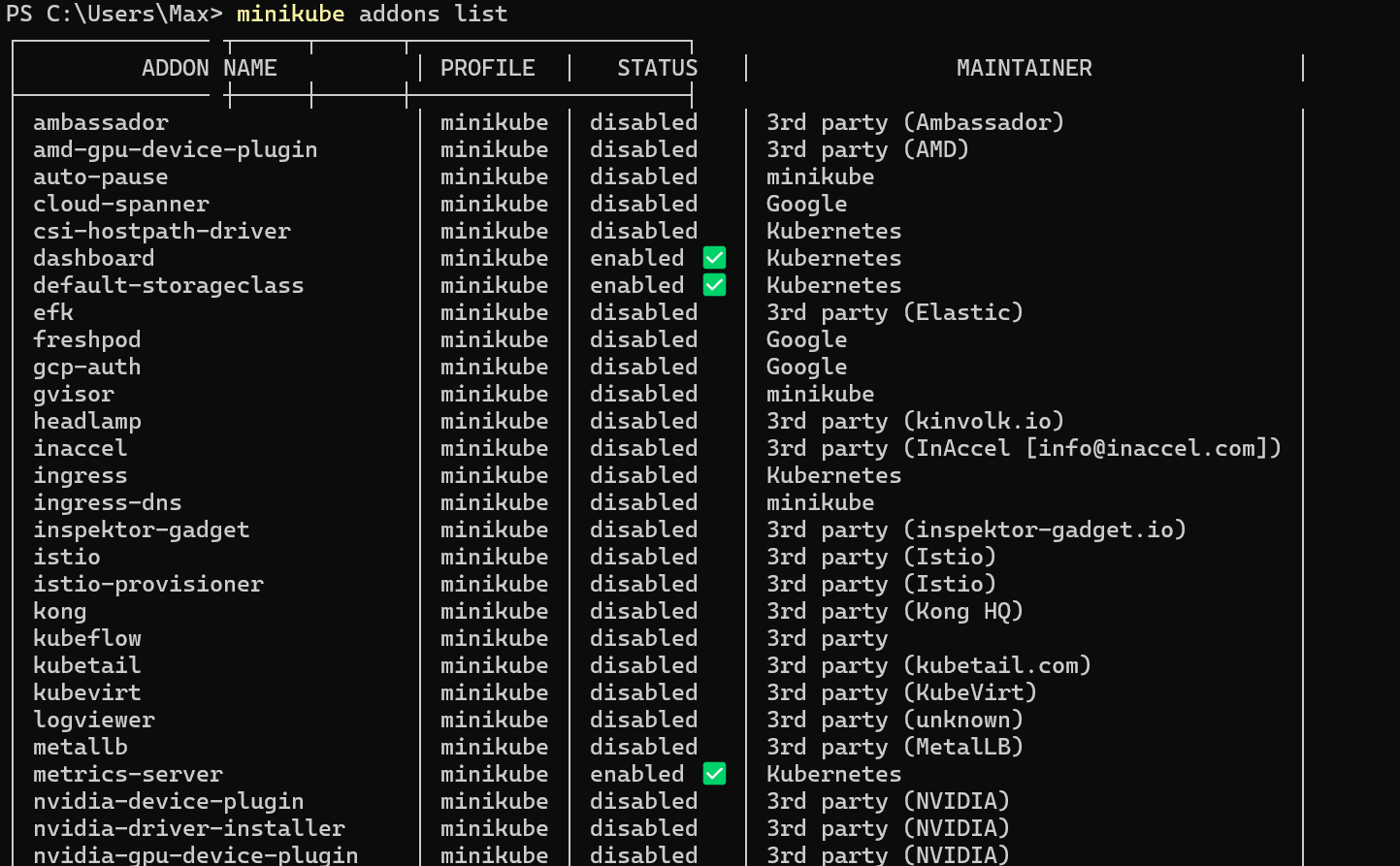
View the Service you created





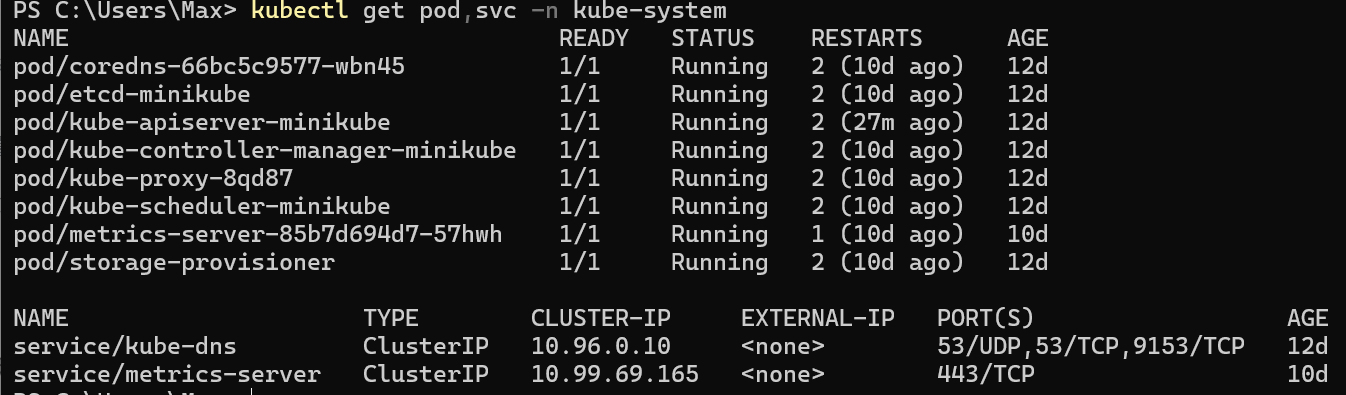


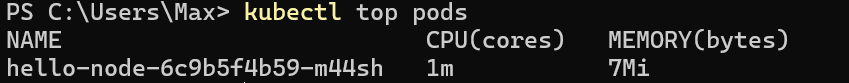
Enable addons





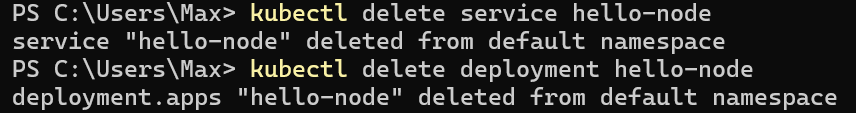
View the Pod and Service you created by installing that addon

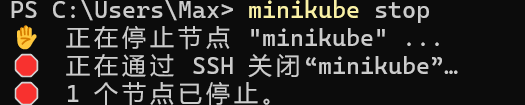






Clean up





Deploy an APP

Using kubectl to create a deployment

A Deployment is responsible for creating and updating instances of your application

Once the application instances are created, a Kubernetes Deployment controller continuously monitors those instances. If the Node hosting an instance goes down or is deleted, the Deployment controller replaces the instance with an instance on another Node in the cluster. ****This provides a self-healing mechanism to address machine failure or maintenance.****

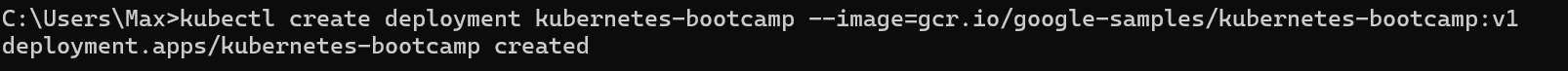
The common format of a kubectl command is: kubectl action resource

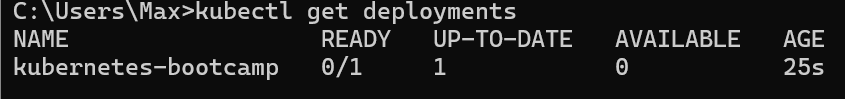
Check that kubectl is configured to talk to your cluster, by running the kubectl version command.

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

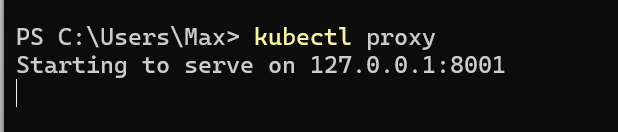
### Deploy an app

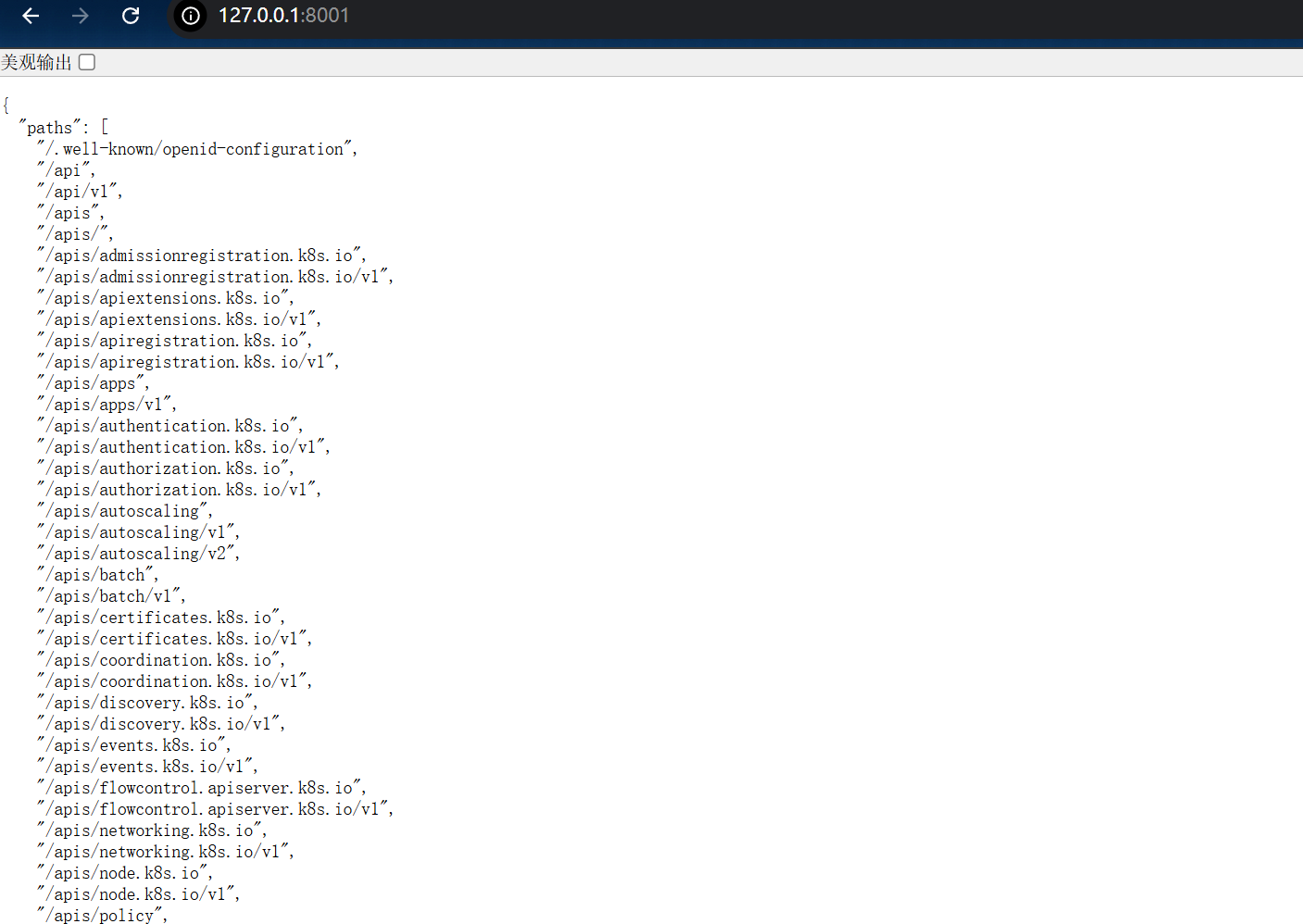
kubectl create deployment kubernetes-bootcamp --image=gcr.io/google-samples/kubernetes-bootcamp:v1

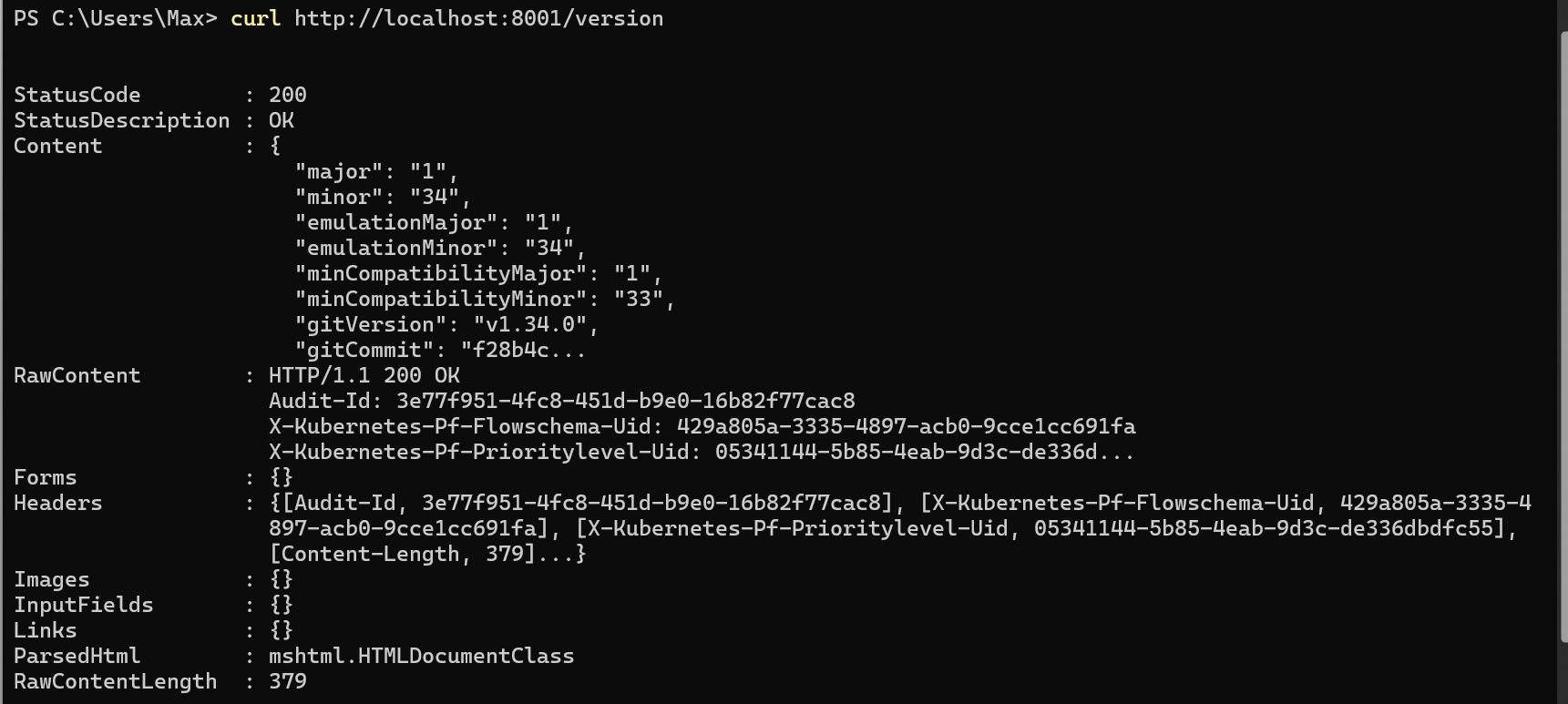




### View the app





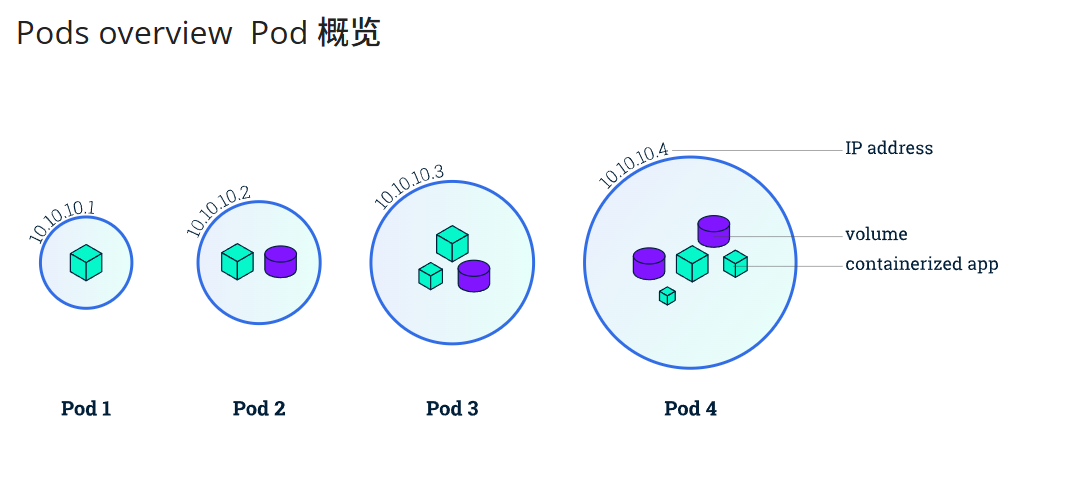


# **Viewing Pods and Nodes**

A Pod is a group of one or more application containers (such as Docker) and includes shared storage (volumes), IP address and information about how to run them.

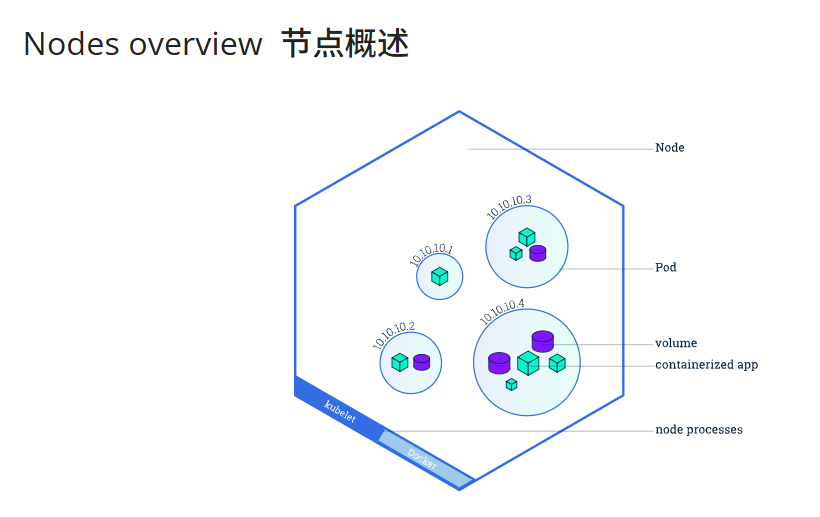
The containers in a Pod share an IP Address and port space, are always co-located and co-scheduled, and run in a shared context on the same Node.

Pods are the atomic unit on the Kubernetes platform. When we create a Deployment on Kubernetes, that Deployment creates Pods with containers inside them (as opposed to creating containers directly). Each Pod is tied to the Node where it is scheduled, and remains there until termination (according to restart policy) or deletion. In case of a Node failure, identical Pods are scheduled on other available Nodes in the cluster.

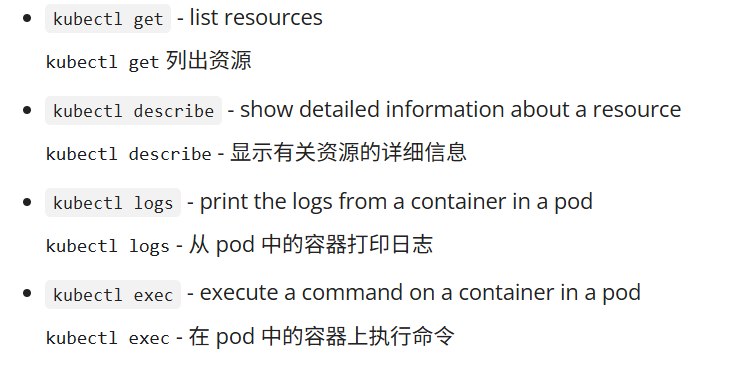


Containers should only be scheduled together in a single Pod if they are tightly coupled and need to share resources such as disk.

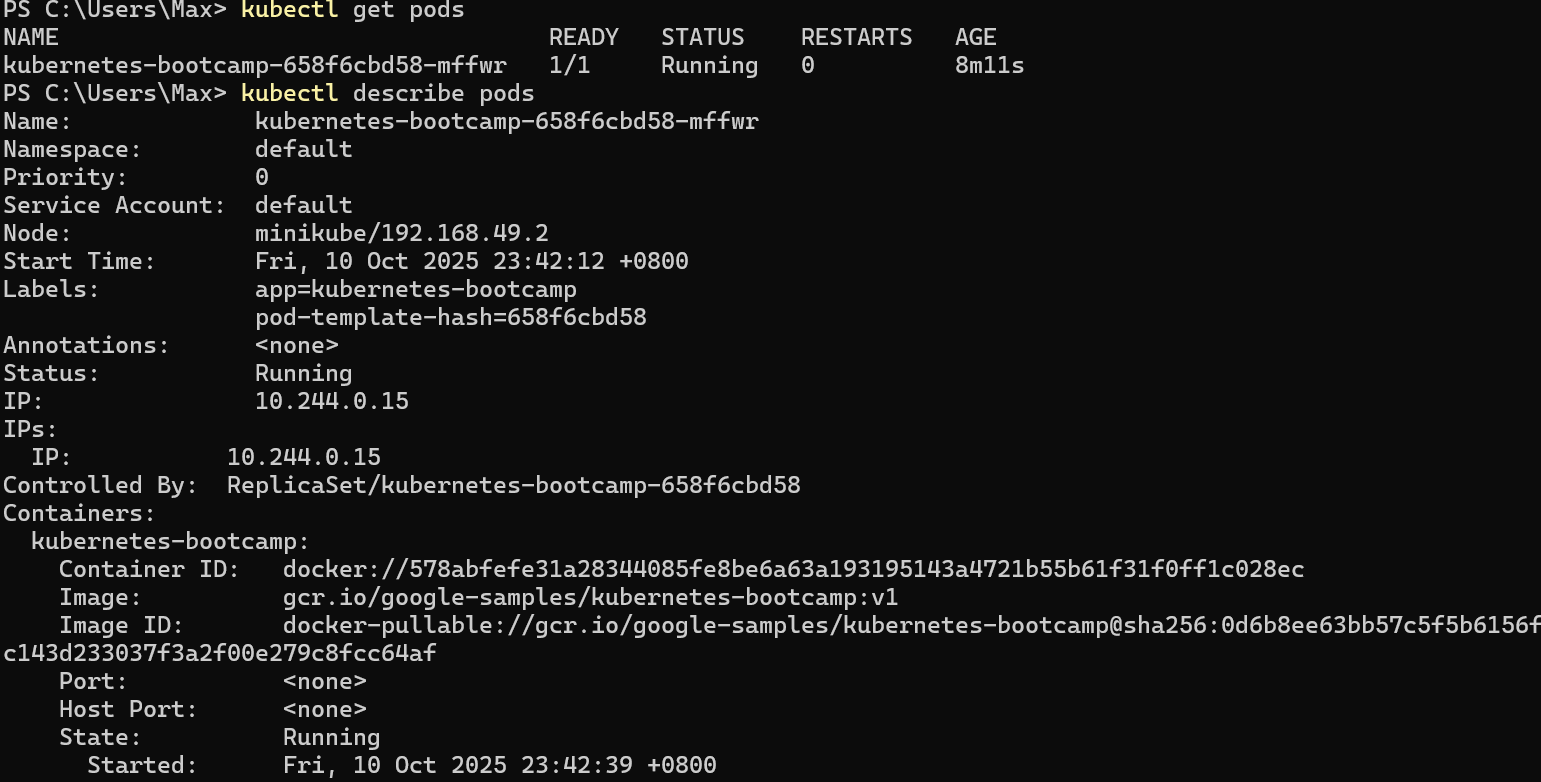
A Pod always runs on a ****Node****. A Node is a worker machine in Kubernetes and may be either a virtual or a physical machine, depending on the cluster. Each Node is managed by the control plane. A Node can have multiple pods, and the Kubernetes control plane automatically handles scheduling the pods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



## Troubleshooting with kubectl



### Check application configuration



The URL is the route to the API of the Pod.

We don't need to specify the container name, because we only have one container inside the pod.

### Executing commands on the container

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:

kubectl exec "$POD\_NAME" -- env

kubectl exec -ti $POD\_NAME -- bash

# **Using a Service to Expose Your App**

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 Kubernetes Service is an abstraction layer which defines a logical set of Pods and enables external traffic exposure, load balancing and service discovery for those Pods

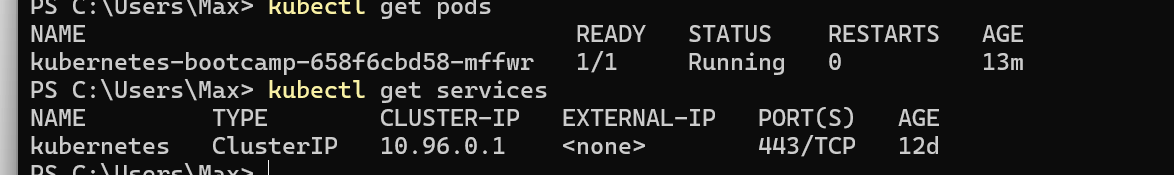
A [Service](https://kubernetes.io/docs/concepts/services-networking/service/) in Kubernetes is an abstraction which defines a logical set of Pods and a policy by which to access them. Services enable a loose coupling between dependent Pods. A Service is defined using YAML or JSON, like all Kubernetes object manifests. The set of Pods targeted by a Service is usually determined by a label selector (see below for why you might want a Service without including a selector in the spec).

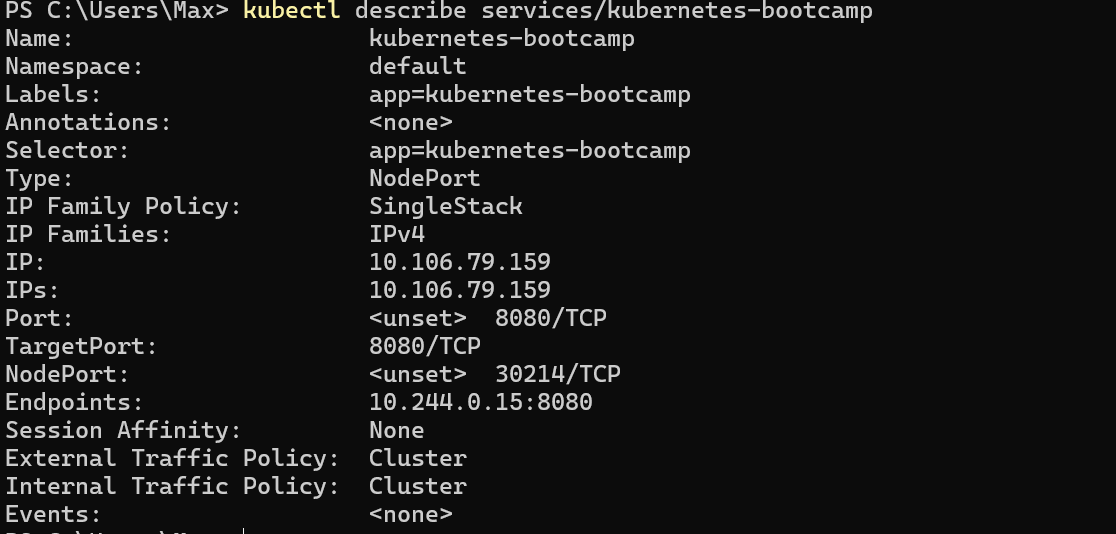
Although each Pod has a unique IP address, those IPs are not exposed outside the cluster without a Service.

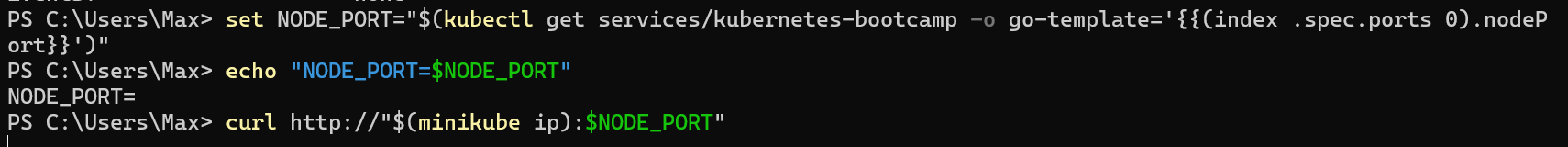
## Services and Labels



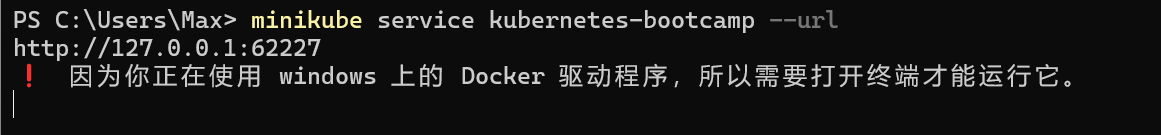
### Creating a new Service

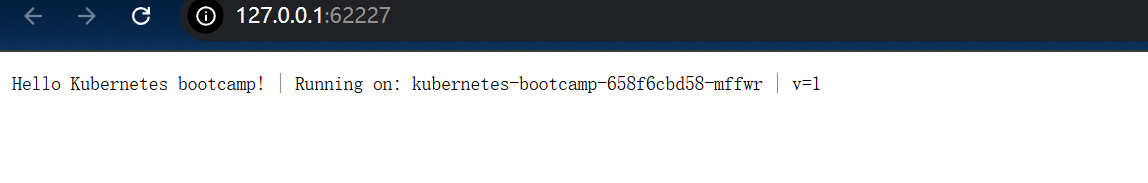


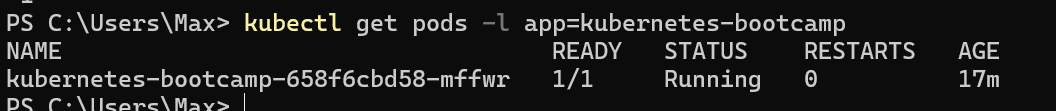


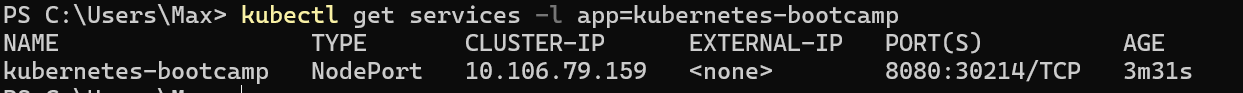


If you're running minikube with Docker Desktop as the container driver, a minikube tunnel is needed. This is because containers inside Docker Desktop are isolated from your host computer.



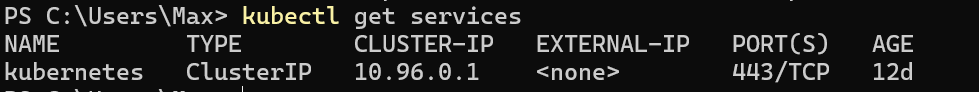




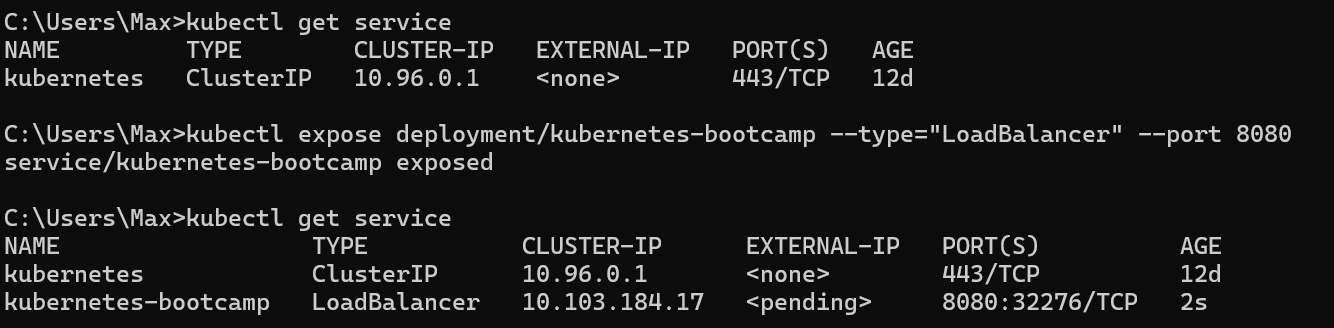


### Deleting a service

kubectl delete service -l app=kubernetes-bootcamp

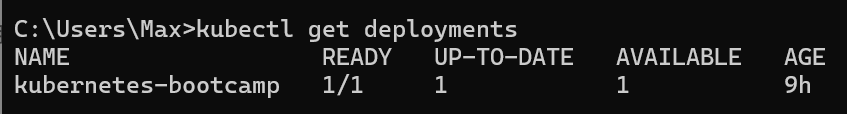


**Running Multiple Instances of Your App**You can create from the start a Deployment with multiple instances using the --replicas parameter for the kubectl create deployment command.

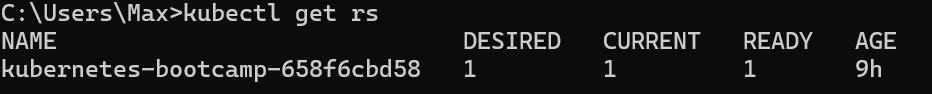


Scaling is accomplished by changing the number of replicas in a Deployment

Scaling a Deployment







Notice that the name of the ReplicaSet is always formatted as [DEPLOYMENT-NAME]-[RANDOM-STRING]. The random string is randomly generated and uses the pod-template-hash as a seed.

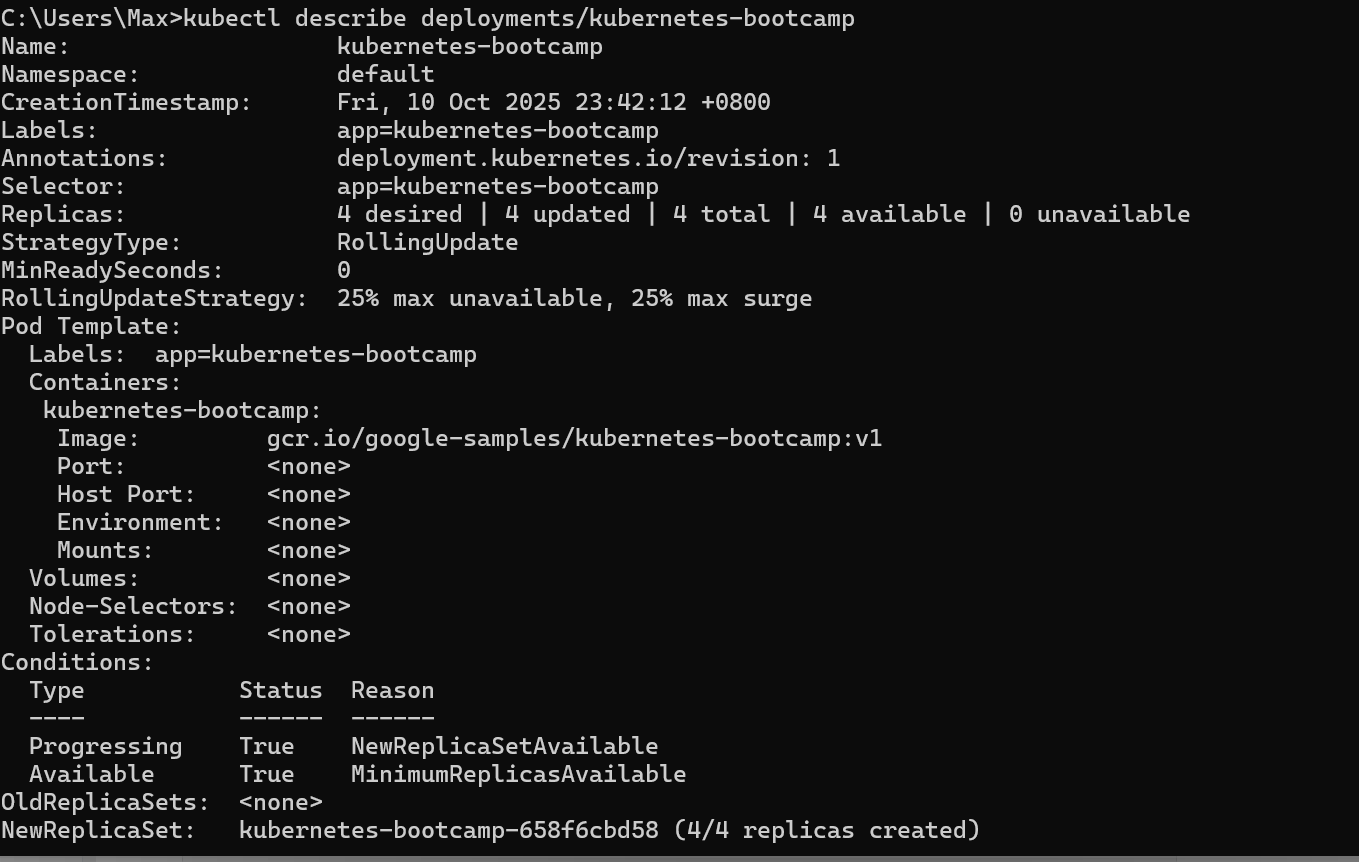
DESIRED displays the desired number of replicas of the application, which you define when you create the Deployment. This is the desired state.

CURRENT displays how many replicas are currently running.

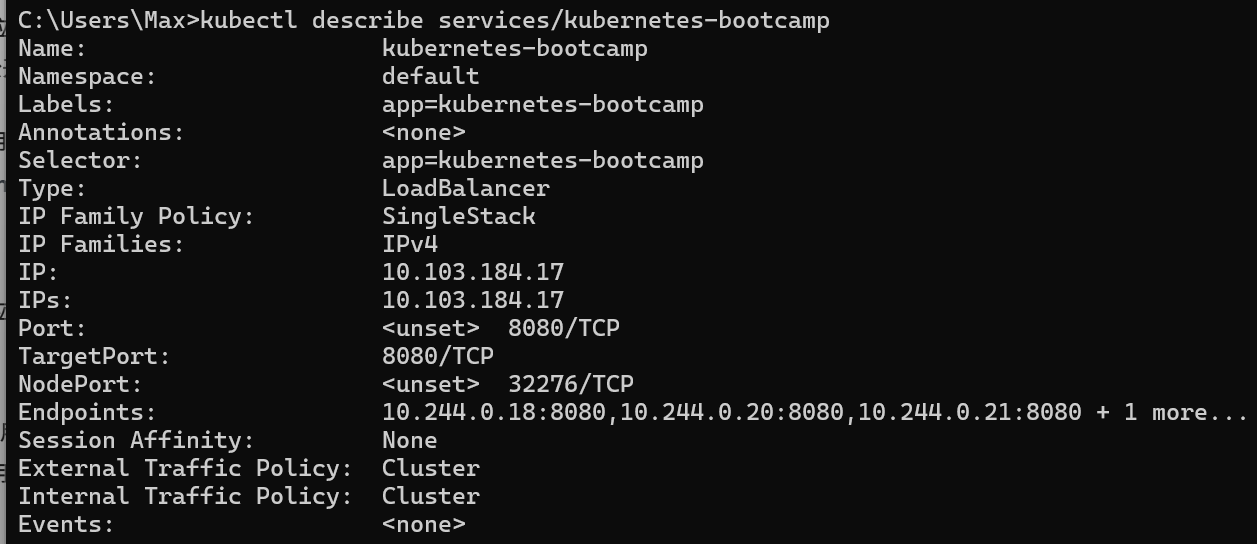
kubectl scale deployments/kubernetes-bootcamp --replicas=4  

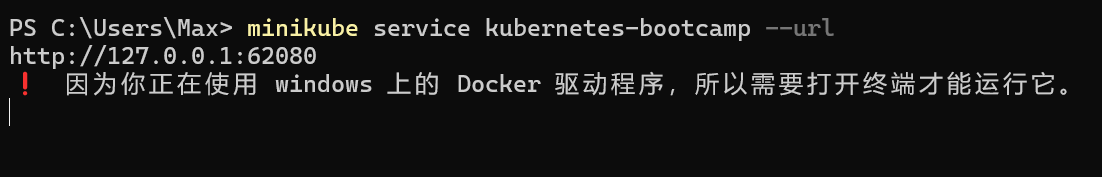


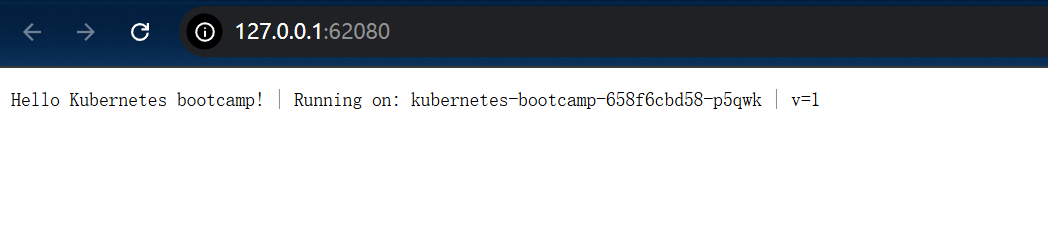

There are 4 Pods now, with different IP addresses.



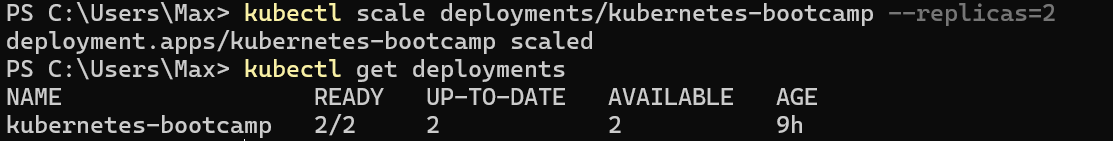
Load Balancing

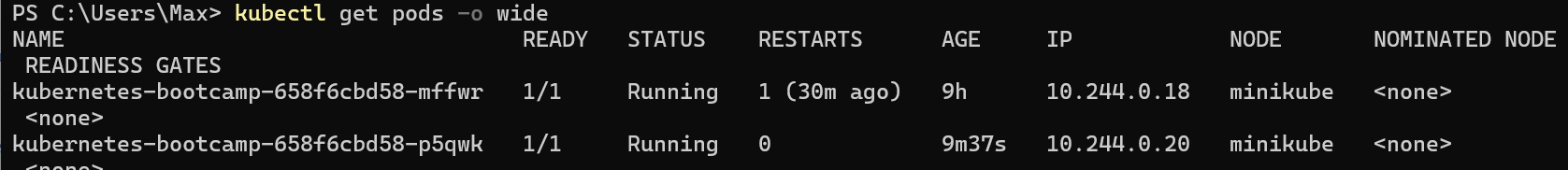






Scale down





Update your APP

Rolling updates allow Deployments' update to take place with zero downtime by incrementally updating Pods instances with new ones.

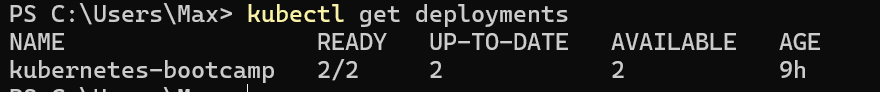
If a Deployment is exposed publicly, the Service will load-balance the traffic only to available Pods during the update.

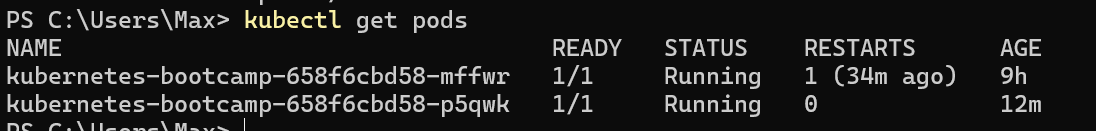
Rolling updates allow the following actions:

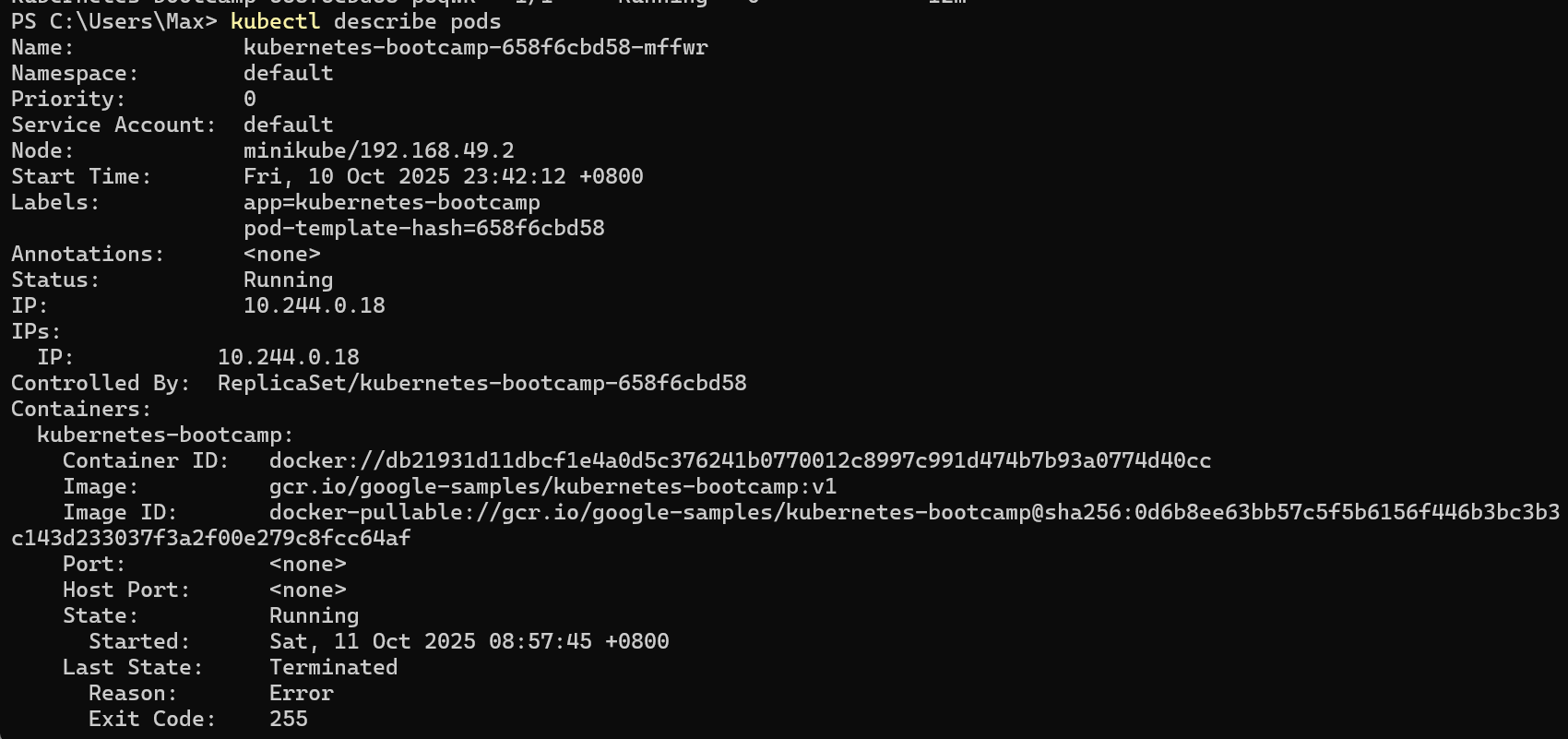
Promote an application from one environment to another (via container image updates)

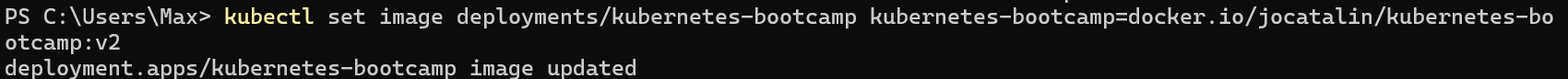
Rollback to previous versions

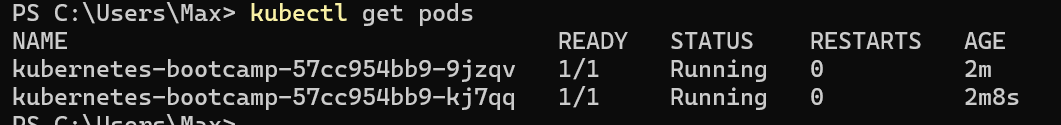
Continuous Integration and Continuous Delivery of applications with zero downtime



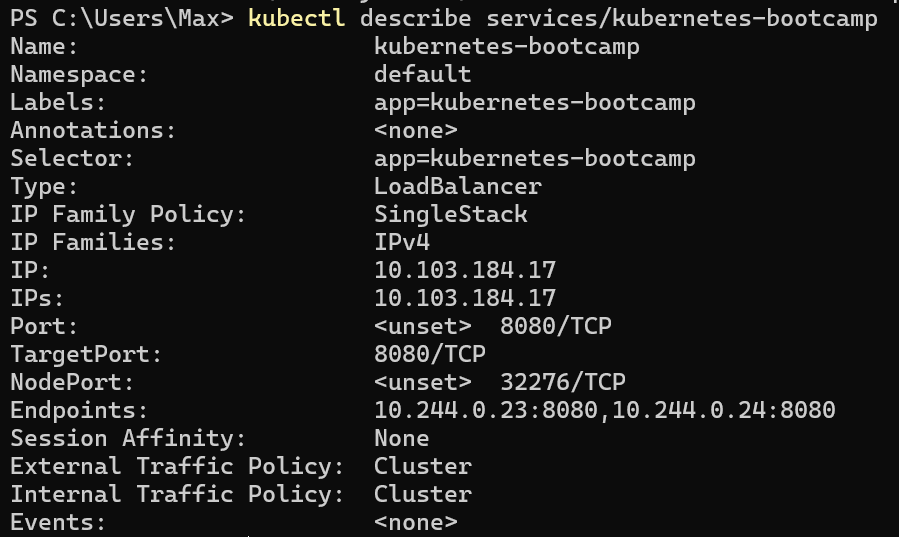


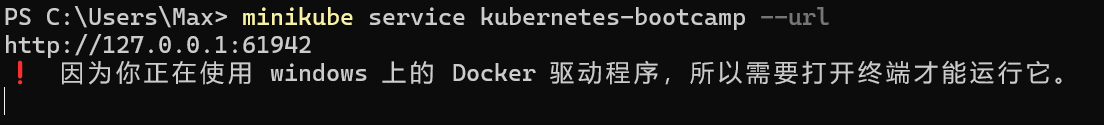


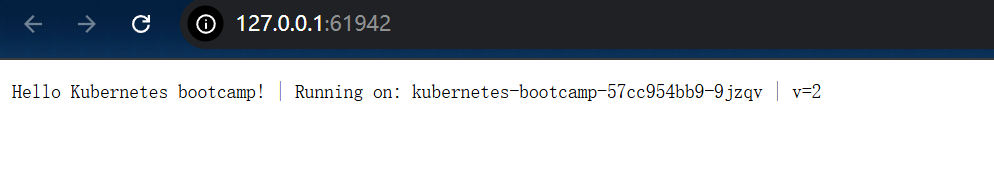


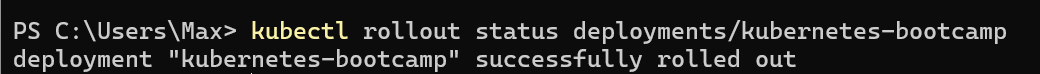


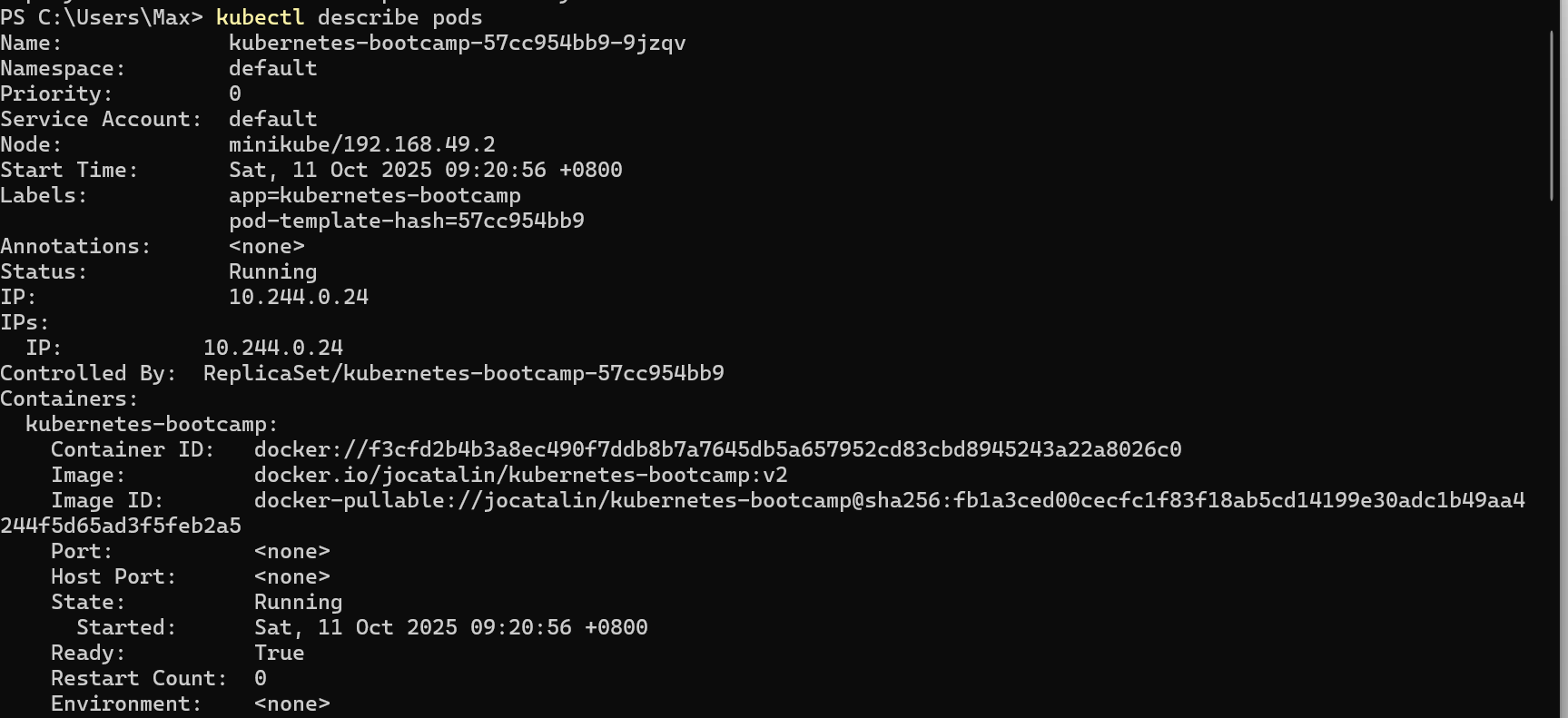
Verify an update



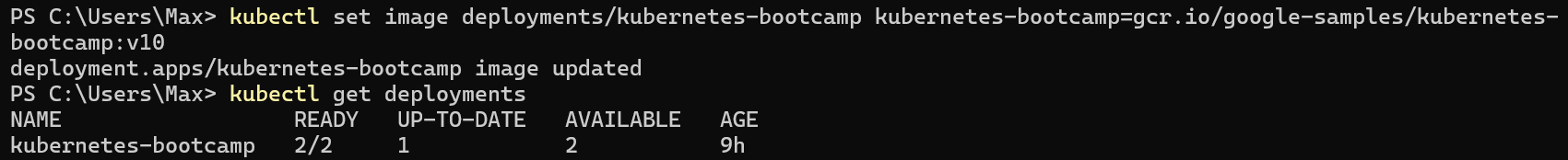


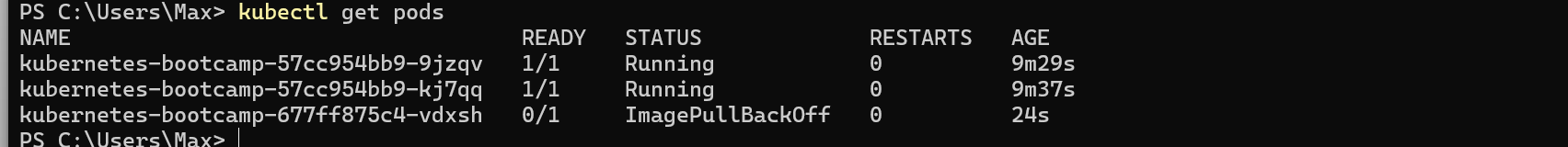


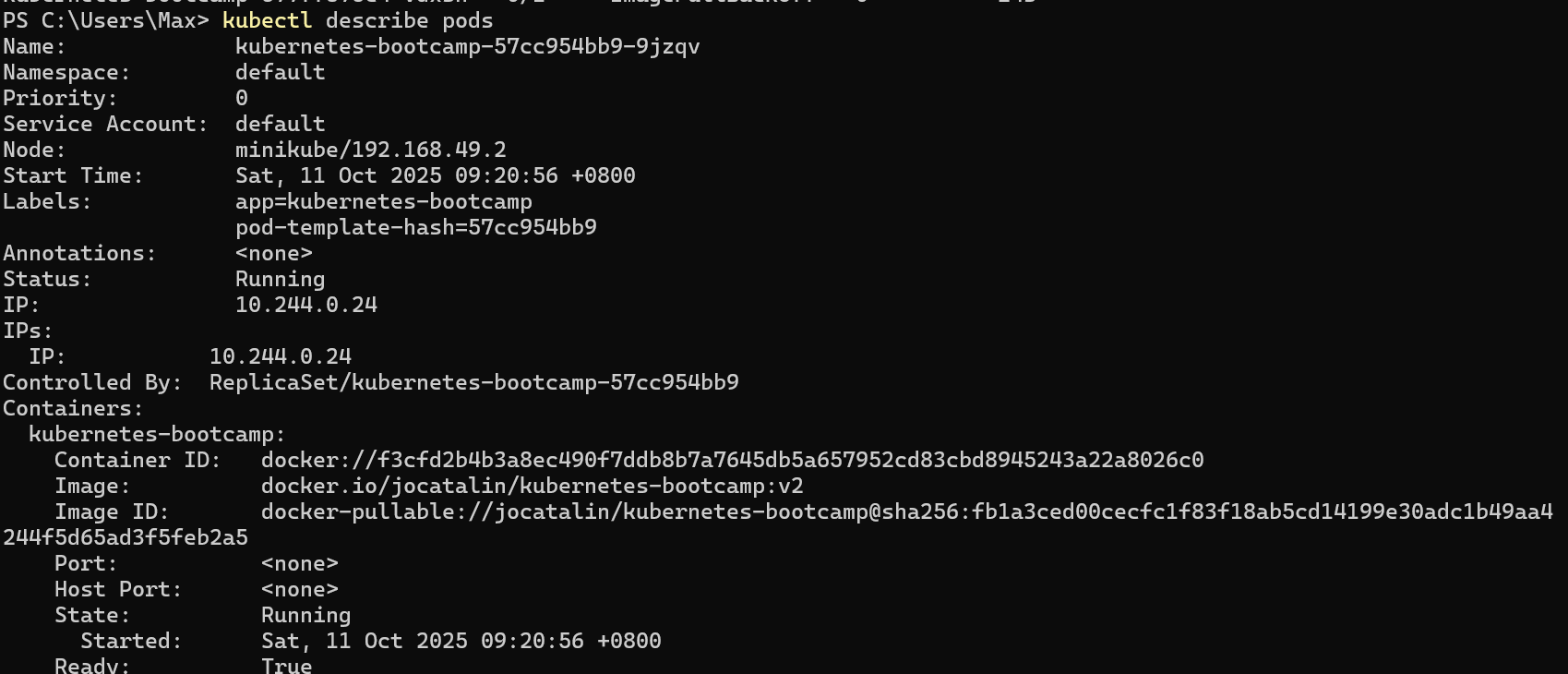




### Roll back an update









The rollout undo command reverts the deployment to the previous known state (v2 of the image). Updates are versioned and you can revert to any previously known state of a Deployment.

