Lab 3

Rong Guo

C00190890

Catalog

[Using Minikube to Create a Cluster 3](#_Toc193332897)

[Step 1: Install Required Tools 3](#_Toc193332898)

[Step 2: Start Minikube Cluster 4](#_Toc193332899)

[Step 3: Verify the Kubernetes Cluster 5](#_Toc193332900)

[Step 4: Deploy an Application 5](#_Toc193332901)

[Step 5: Stop Minikube 6](#_Toc193332902)

[Using kubectl to Create a Deployment 6](#_Toc193332903)

[Step 1: Create Your Deployment 6](#_Toc193332904)

[Step 2: Verify the Deployment 7](#_Toc193332905)

[Step 3: View the App 7](#_Toc193332906)

[Step 5: Access the App Through the Proxy 7](#_Toc193332907)

[Step 6: Clean Up 8](#_Toc193332908)

[Viewing Pods and Nodes 8](#_Toc193332909)

[Step 1: Understand Kubernetes Pods and Nodes 8](#_Toc193332910)

[Step 2: Verify Kubernetes Pods 8](#_Toc193332911)

[Step 3: Expose the Pod via kubectl Proxy 9](#_Toc193332912)

[Step 4: View Logs of the Pod's Container 10](#_Toc193332913)

[Step 5: Execute Commands Inside the Pod 11](#_Toc193332914)

[Expose Your App Publicly 11](#_Toc193332915)

[Step 1: Create a Service to Expose the App 11](#_Toc193332916)

[Step 3: Verify the Service 12](#_Toc193332917)

[Step 4: Access the App Externally 12](#_Toc193332918)

[Running Multiple Instances of Your App 12](#_Toc193332919)

[Step 1: Check Current Deployment 12](#_Toc193332920)

[Step 2: Check the ReplicaSet 12](#_Toc193332921)

[Step 3: Scale the Deployment 13](#_Toc193332922)

[Step 4: Verify the Number of Running Pods 13](#_Toc193332923)

[Step 5: Scaling Down the Deployment 13](#_Toc193332924)

[Performing a Rolling Update 14](#_Toc193332925)

[Step 1: Check the Existing Deployment 14](#_Toc193332926)

[Step 2: Update the App Version 14](#_Toc193332927)

[Roll back an update 15](#_Toc193332928)

Using Minikube to Create a Cluster

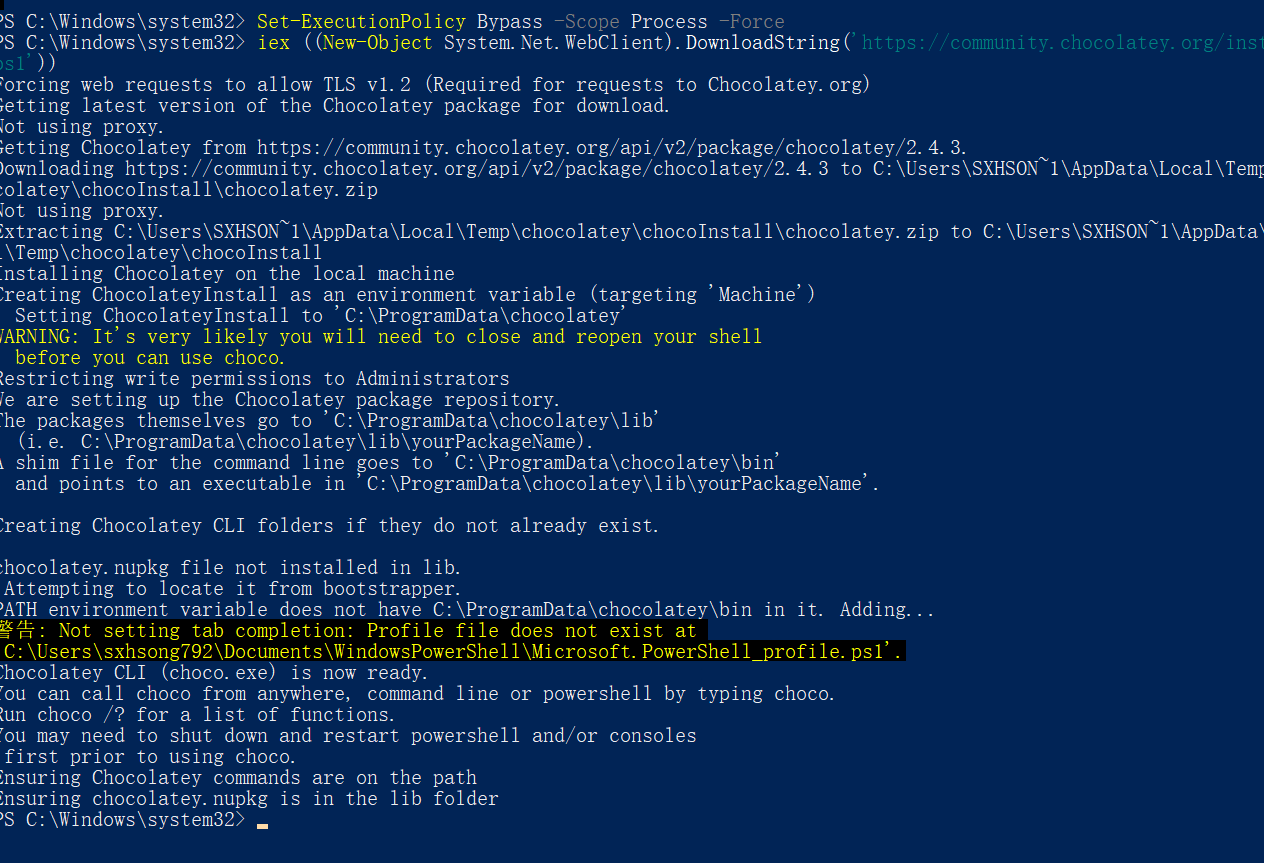
Step 1: Install Required Tools

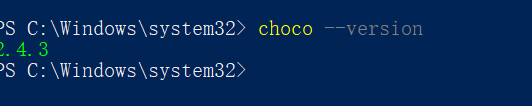
1. Install Virtualization Software

Minikube requires virtualization software to create virtual machines on your local machine. On Windows, I can use VirtualBox or Hyper-V.

2. Install Minikube

Next, I need to install Minikube. I'll use Chocolatey, a package manager for Windows, to install it. If you don’t have Chocolatey installed, you can follow the installation instructions on the Chocolatey website.





Install Minikube via Chocolatey: Open PowerShell (in Administrator mode) and run the following command

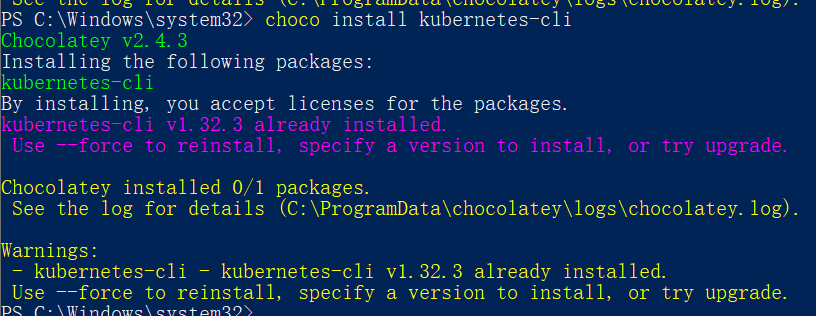
文本

描述已自动生成

3. Install kubectl

kubectl is the command-line tool that allows me to interact with my Kubernetes cluster. I'll install it using Chocolatey as well:

Install kubectl via Chocolatey:



Step 2: Start Minikube Cluster

1. Start Minikube

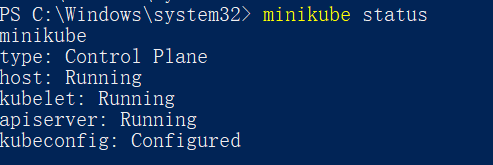
Now, I can start Minikube and create my local Kubernetes cluster. In Command Prompt or PowerShell, I run:

文本

描述已自动生成

2. Check Cluster Status

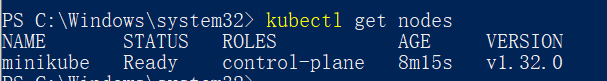
Once the cluster is up and running, I can verify its status by running:



Step 3: Verify the Kubernetes Cluster

1. List Nodes

To see the nodes in my Kubernetes cluster (Minikube typically only has one node since it's a single-node cluster), I run:



Step 4: Deploy an Application

1. Deploy Nginx

I can now deploy a Nginx application to my cluster to test things out. I use the following command:



2. Expose the Service

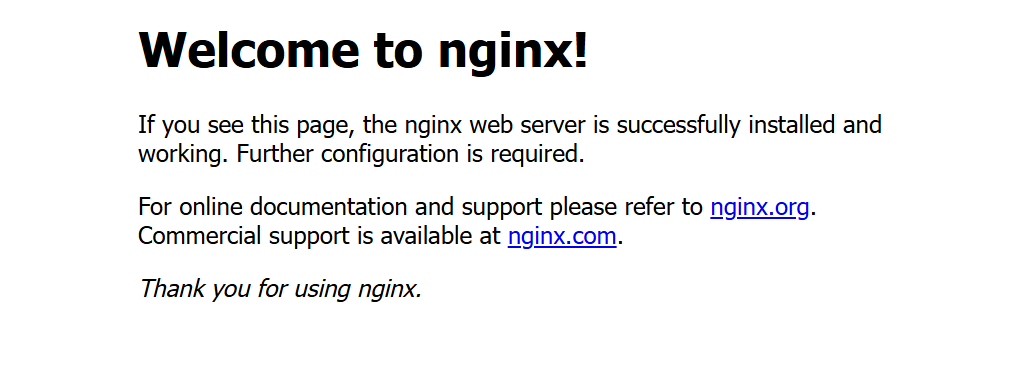
To access the Nginx application, I need to expose it via a service. I run



3. Get the Service URL

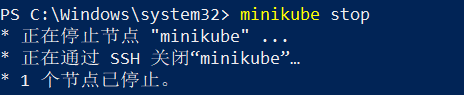
To access the service, I can get the URL where it’s exposed using the following command:





Step 5: Stop Minikube

When I'm done with my cluster, I can stop it using this command:



Using kubectl to Create a Deployment

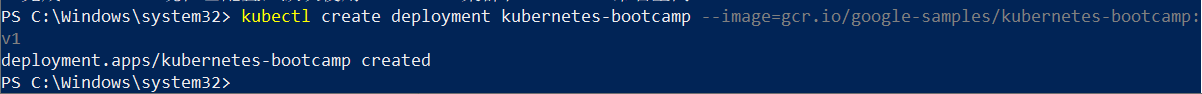
Step 1: Create Your Deployment

Now that I have a running Kubernetes cluster and kubectl configured to communicate with it, I'm ready to deploy my first app. For this tutorial, I'll deploy a sample app called kubernetes-bootcamp.

1. Create a Deployment

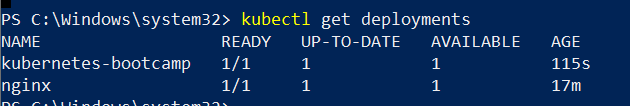
I will use the kubectl create deployment command to create a deployment. I need to specify the deployment name and the container image for the app. The image used in this tutorial is hosted on Google Container Registry.

Here’s the command I’ll run in my terminal (PowerShell or Command Prompt):



Step 2: Verify the Deployment

To check if the deployment was successful and the app is running, I’ll use the following command to list all the deployments in my cluster:



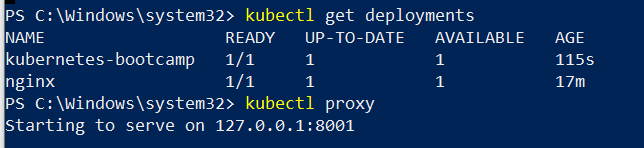
Step 3: View the App

By default, my app is running inside a Pod in Kubernetes. However, Pods in Kubernetes run on a private, isolated network, and I won’t be able to access the app directly without exposing it.

In this tutorial, I’ll use the kubectl proxy command to create a proxy that lets me access the app.

1. Start the kubectl Proxy

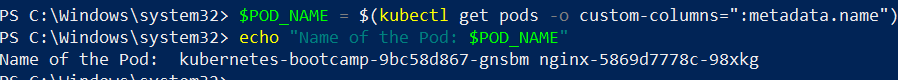
First, I’ll run this command in a second terminal window (this is important because the proxy will keep running and block the terminal until I stop it):



Step 5: Access the App Through the Proxy

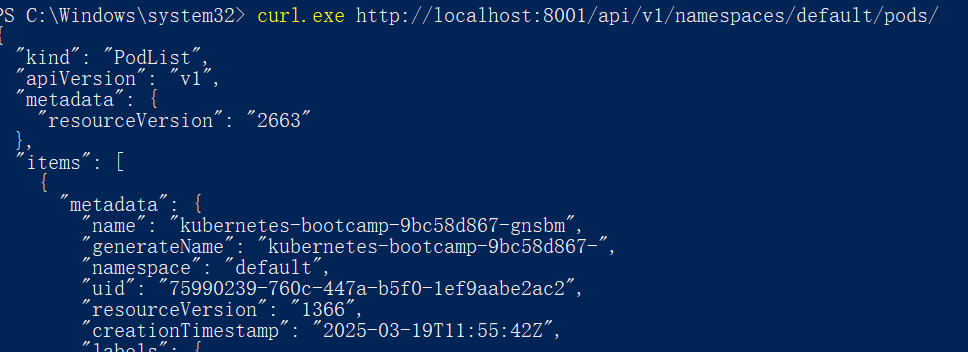
1. Find the Pod Name

Next, I need to get the name of the Pod where the kubernetes-bootcamp app is running. I’ll use the following command:



2. Query the App via the Proxy

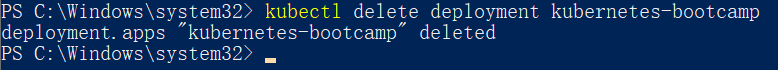
Now that I have the Pod name, I can use curl to access the app through the proxy. I’ll run this command in the terminal (still in the same window where the proxy is running):



Step 6: Clean Up

Once I’m done testing, I can stop the kubectl proxy by pressing Ctrl + C in the second terminal window.

If I want to delete the deployment and clean up the resources in my cluster, I can run:



Viewing Pods and Nodes

Step 1: Understand Kubernetes Pods and Nodes

Before troubleshooting, it’s important to understand Pods and Nodes:

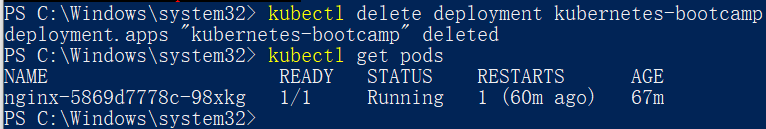
Pods: These are the smallest deployable units in Kubernetes. A Pod contains one or more containers and shares resources such as storage and networking. If I create a deployment, Kubernetes automatically creates Pods to run my app containers.

Nodes: These are the machines (physical or virtual) where the Pods are scheduled to run. Nodes are managed by the Kubernetes control plane

Step 2: Verify Kubernetes Pods

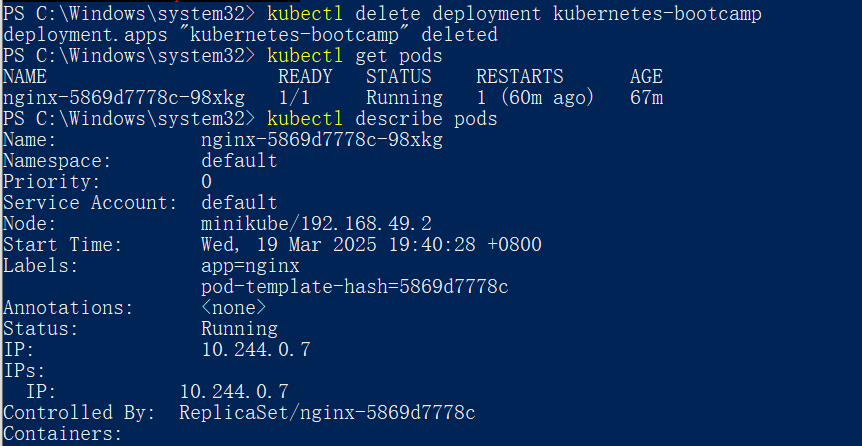
1. Check the Pods in My Cluster

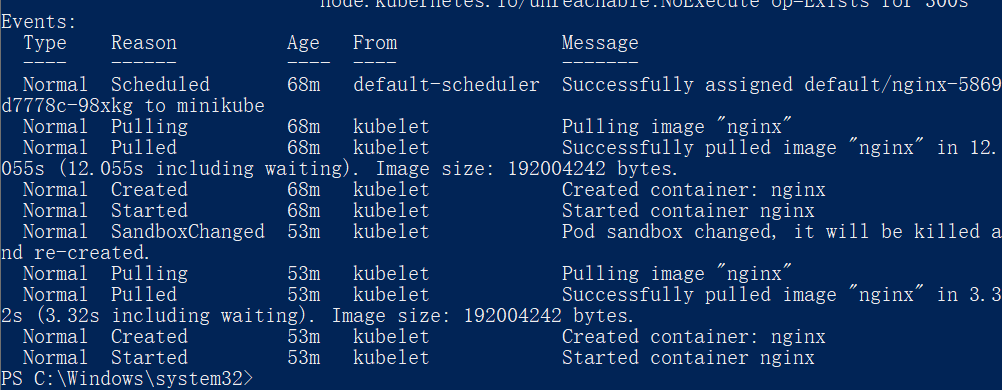
To check if my application Pods are running, I’ll run the following command:



2. Get Detailed Information About the Pod

Next, I want to know more about the Pod, like its IP address and ports. I’ll use the kubectl describe command for this:





This will give me detailed information, such as:

The Pod's IP address.

Containers inside the Pod, their image versions, and any exposed ports.

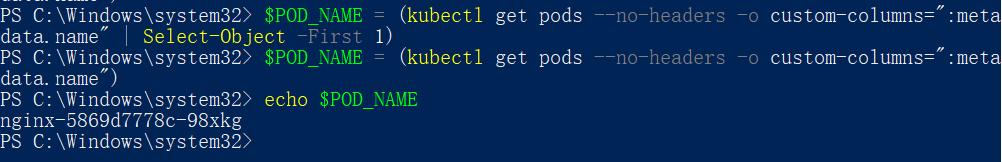
A list of events related to the lifecycle of the Pod.

Step 3: Expose the Pod via kubectl Proxy

Since Pods are on a private network, I can’t access them directly from my browser. Instead, I’ll use kubectl proxy to expose my Pod temporarily.

1. Start the kubectl Proxy

I’ll open a new terminal window and run this command to create a proxy that will allow me to interact with the Pod's API:



2. Get the Pod Name

To interact with the Pod, I first need to get its name. I can use this command:

3. Query the Pod

Now, I can query the Pod through the proxy with curl:

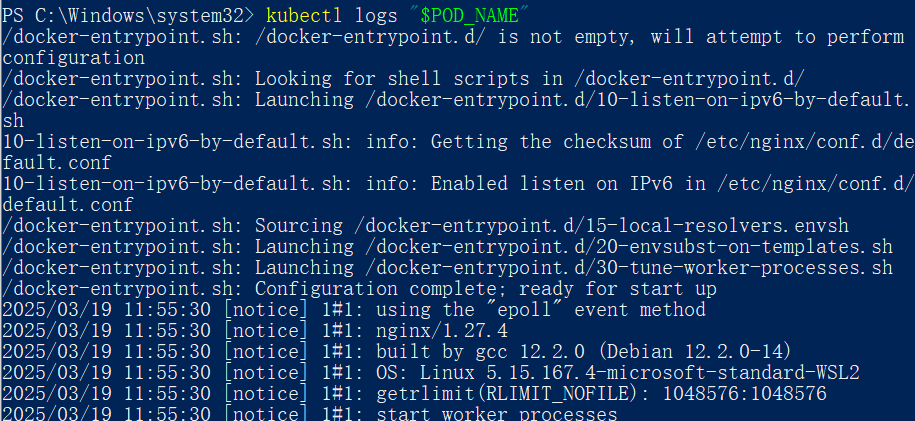


Step 4: View Logs of the Pod's Container

To troubleshoot further, I may want to see the logs of the container running inside the Pod. I can do this with the kubectl logs command.

1. View the Logs

I’ll run this command to view the logs of my Pod:



Step 5: Execute Commands Inside the Pod

Sometimes, it’s helpful to execute commands directly inside the Pod’s container to diagnose issues or check the application status.

1. Execute a Command in the Pod

Let’s say I want to see the environment variables inside the container. I can use kubectl exec to run commands inside the Pod:

文本

描述已自动生成

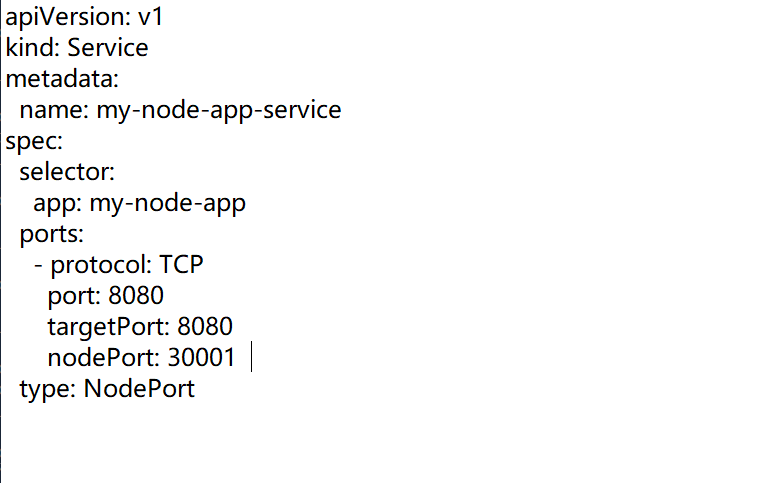
Expose Your App Publicly

Step 1: Create a Service to Expose the App

To expose my app outside the Kubernetes cluster, I need to create a Service.

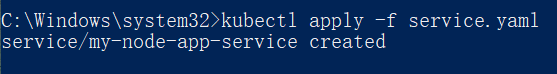
1. Create a Service YAML File

I'll create a YAML file called service.yaml to define the Service. Here’s an example:



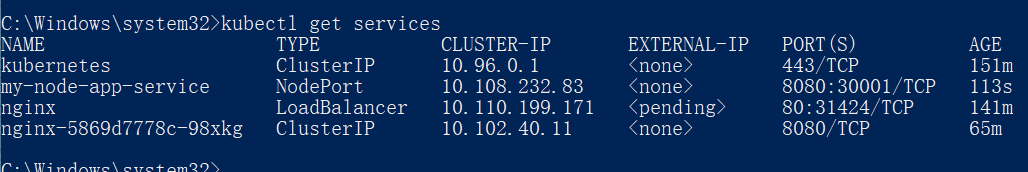
2. Apply the Service Configuration

Now, I’ll apply this YAML file to create the Service in my cluster:



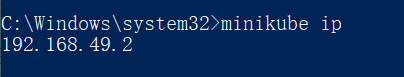
Step 3: Verify the Service

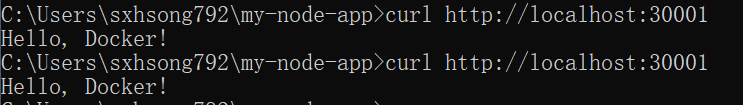
I want to check if my Service was created successfully. To do this, I can run:



Step 4: Access the App Externally

I need to find the IP address of the Node where my app is running. If I’m using Minikube, I can get the Node’s IP with:

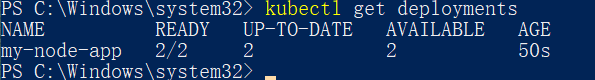




Running Multiple Instances of Your App

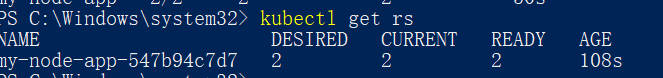
Step 1: Check Current Deployment

First, let me check my Deployments:



Step 2: Check the ReplicaSet

I run the following command to check the ReplicaSet associated with my Deployment:

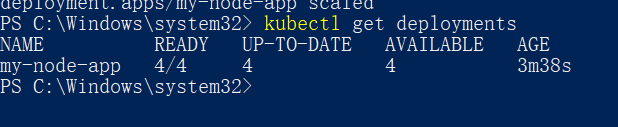


Step 3: Scale the Deployment

Now, I decide to scale my Deployment to 4 Pods:

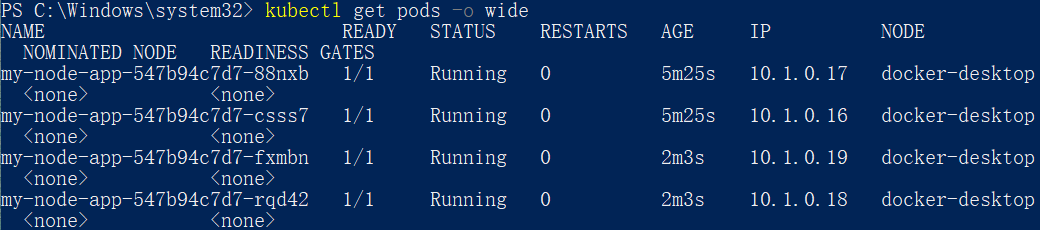


To verify the change, I check my Deployments again:



Step 4: Verify the Number of Running Pods

Next, I list the running Pods to confirm the scaling:



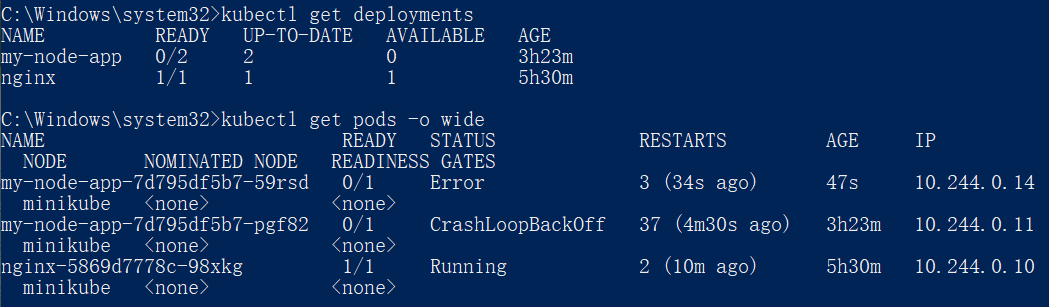
There should now be 4 Pods, each with a different IP address.

To check the detailed status, I use:



Step 5: Scaling Down the Deployment

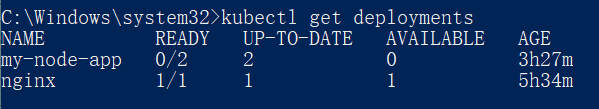
To confirm the change, I check the Deployments again. And I list the Pods to see that two have been terminated:



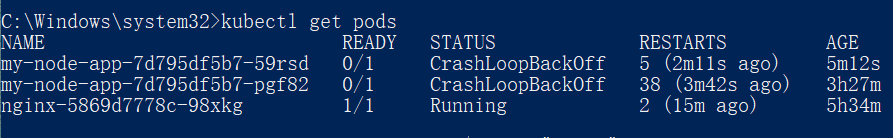
Performing a Rolling Update

Step 1: Check the Existing Deployment

First, I need to check my running Deployments:



Then, I list my running Pods

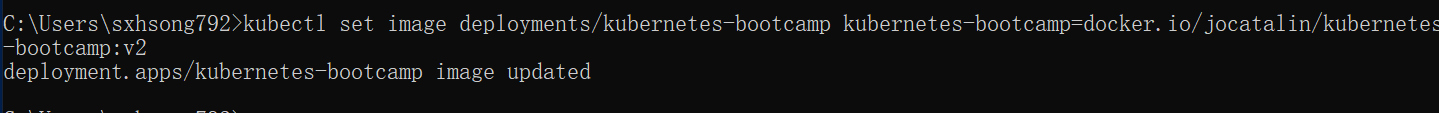


To check the current image version, I describe one of the Pods



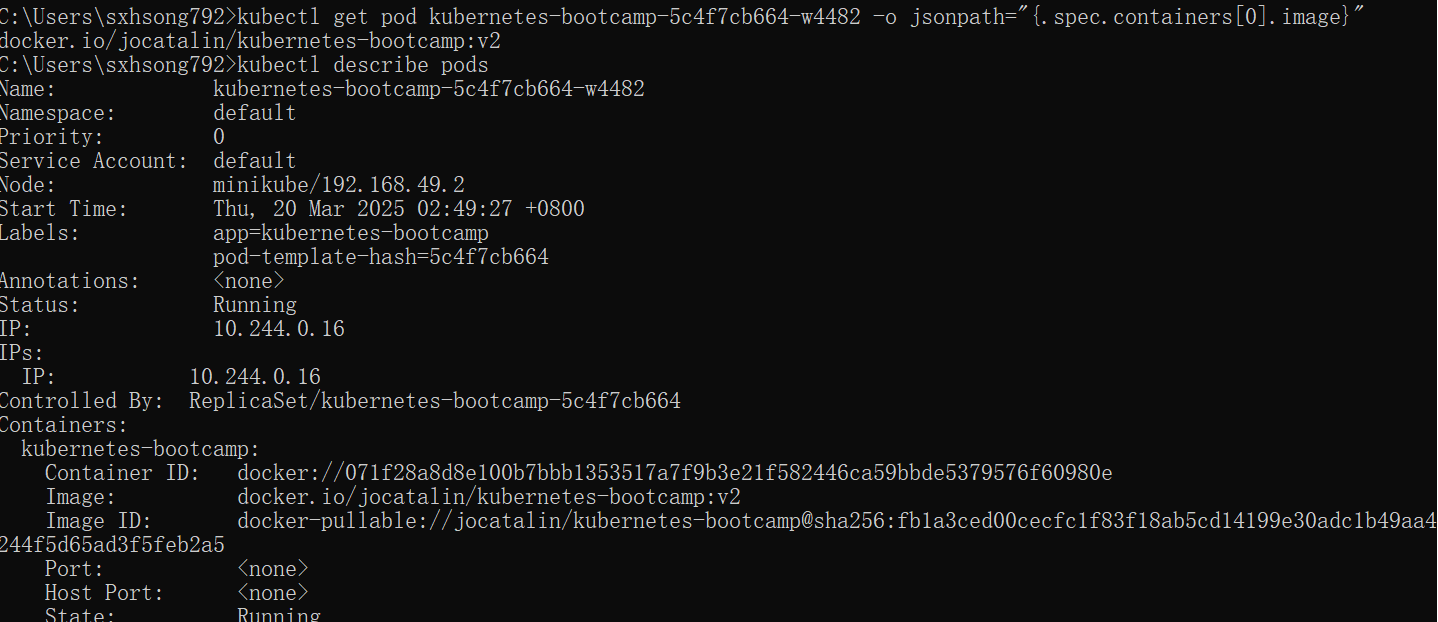
Step 2: Update the App Version

Now, I update the application image to version 2



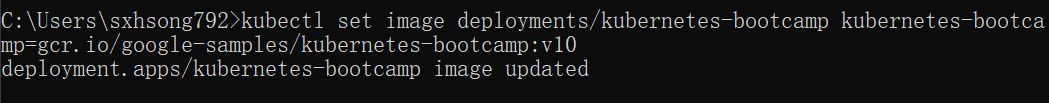
Verify an update

In the Image field of the output, verify that you are running the latest image version (v2).

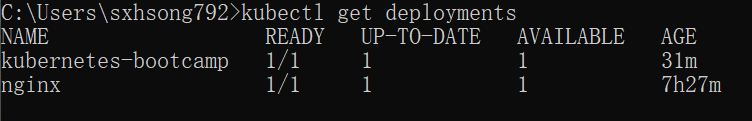


Roll back an update

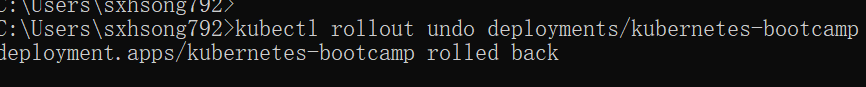
Let’s perform another update, and try to deploy an image tagged with v10



Use get deployments to see the status of the deployment



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. To check the image deployed on the running Pods, use the describe pods subcommand:

