Introduction to Cloud Computing Report

Lab 6: Kubernetes Basics Tutorial

Name: Yin Yuang
Student ID: 202383930027
Class: Software Engineering Class 1

Introduction to Cloud Computing
(Autumn,2025)

Nanjing University of Information Science and Technology, China
Waterford Institute

I. Object

The objective of this experiment is to understand and practice the fundamental operations of Kubernetes in a local environment using Minikube. Through hands-on experimentation, this lab focuses on deploying containerized applications, managing Pods and Deployments, exposing applications through Kubernetes Services, scaling applications to multiple replicas, and performing rolling updates and rollback operations. By completing this experiment, students gain practical experience with core Kubernetes concepts, including container orchestration, service discovery, scalability, and fault recovery, as well as proficiency in using the `kubectl` command-line tool.

II. Content

This experiment mainly includes the following tasks:

- Initializing a local Kubernetes cluster and dashboard using Minikube
- Creating and managing application Deployments using `kubectl`
- Inspecting Pods, Nodes, and application logs to verify runtime status
- Accessing applications through `kubectl proxy` and Kubernetes Services
- Exposing applications externally using NodePort and LoadBalancer Services
- Scaling applications to multiple replicas and verifying service load balancing
- Performing rolling updates, analyzing update failures, and executing rollback operations

III. Results

This section presents the experimental results obtained from deploying and managing applications on a local Kubernetes cluster using Minikube. The results are organized according to the main experimental objectives, including cluster initialization, application deployment, service exposure, scaling, and rolling updates.

1.Hello Minikube

The experiment began with initializing a local Kubernetes cluster using Minikube to provide a controlled environment for subsequent deployment and management tasks.

1.1 Starting the Minikube Cluster

The local Kubernetes cluster was initialized using the following command:

```
minikube start
```

After execution, Minikube successfully started the control-plane node and configured `kubectl` to use the Minikube context.

Figure 1: Successful startup of the Minikube cluster

1.2 Opening the Kubernetes Dashboard

To visualize and manage Kubernetes resources, the Kubernetes Dashboard was launched in a new terminal:

```
minikube dashboard
```

This command enabled the dashboard addon and created a temporary proxy, allowing access to the dashboard from a web browser.

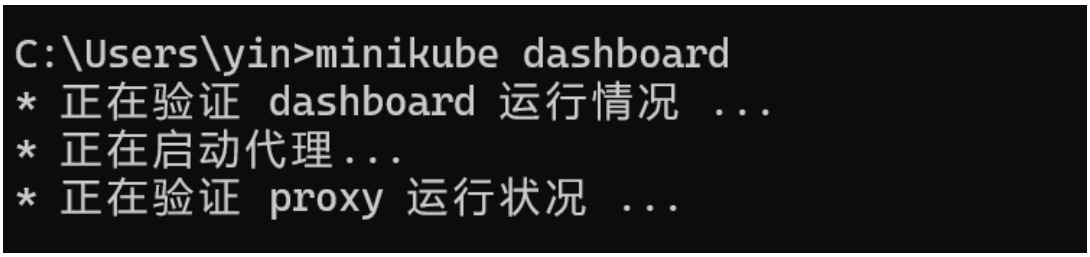

Figure 2: Kubernetes Dashboard accessed through Minikube

1.3 Creating a Deployment

A Deployment named `hello-node` was created using a test container image that provides a web server on port 8080:

```
kubectl create deployment hello-node --image=registry.k8s.io/e2e-test-images/agnhost:2.53 --
/agnhost netexec --http-port=8080
```

1.4 Verifying Deployment and Pod Status

The status of the Deployment and Pod was verified using the following commands:

```
kubectl get deployments
kubectl get pods
```

The output confirmed that the Pod was in the `Running` state.

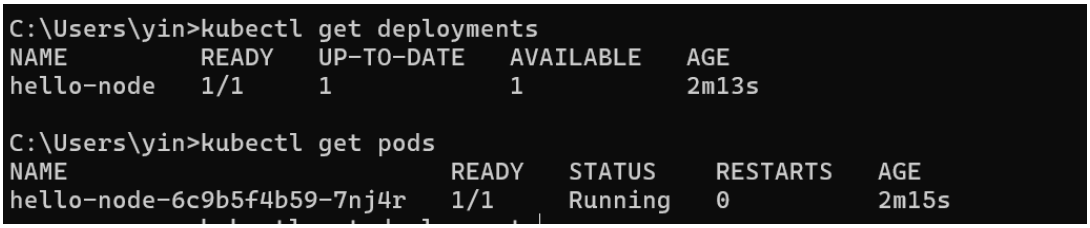

Figure 3: Deployment and Pod status

1.5 Viewing Application Logs

The logs of the running container were obtained using:

```
kubectl logs hello-node-<pod-name>
```

The logs showed that the HTTP server started successfully on port 8080.

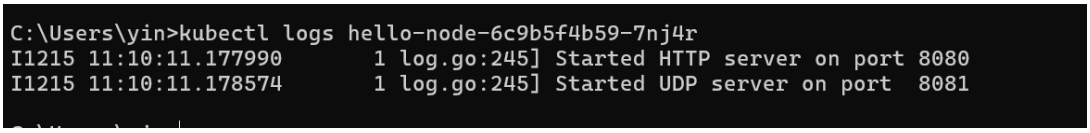

Figure 4: Application logs of the hello-node Pod

1.6 Exposing the Application as a Service

To allow external access, the Deployment was exposed as a Service of type `LoadBalancer`:

```
kubectl expose deployment hello-node --type=LoadBalancer --port=8080
```

The Service was accessed using:

```
minikube service hello-node
```

This command opened the application in the default web browser.

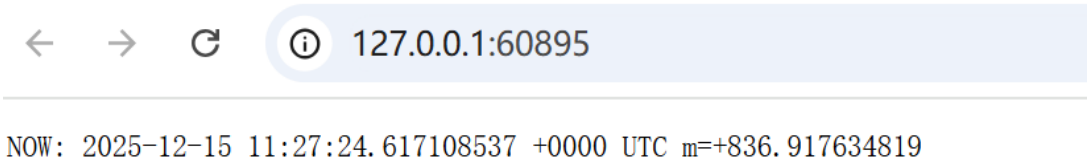

Figure 5: Hello Minikube application accessed in browser

2.Using kubectl to Create a Deployment

2.1 Cluster Verification

After starting the cluster, the status of the Kubernetes node was verified using the following command:

```
kubectl get nodes
```

The output confirms that the Minikube node is in the `Ready` state, indicating that the cluster is operational.

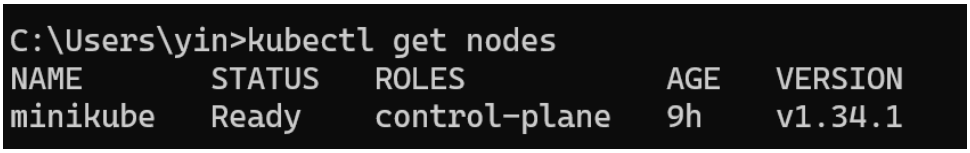

Figure 6: Kubernetes node status

2.2 Application Deployment

A Kubernetes Deployment named `kubernetes-bootcamp` was created using the following command:

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

This command instructs Kubernetes to create a Deployment using the specified container image. Kubernetes automatically schedules the Pod on the available node and ensures its availability.

```
C:\Users\yin>kubectl create deployment hello-node --image=registry.k8s.io/e2e-test-images/agnhost:2.53 -- /
agnhost netexec --http-port=8080
deployment.apps/hello-node created
```

Figure 7: Creating the Kubernetes Deployment

The Deployment status was verified using:

```
kubectl get deployments
```

```
C:\Users\yin>kubectl get deployments
NAME                  READY   UP-TO-DATE   AVAILABLE   AGE
kubernetes-bootcamp   1/1     1            1           2m27s
```

Figure 8: Deployment status showing one running replica

2.3 Accessing the Application Using kubectl Proxy

By default, Pods run in an internal Kubernetes network and are not accessible externally. To access the application, the `kubectl proxy` command was used to create a temporary proxy between the local machine and the Kubernetes API server.

```
kubectl proxy
```

This proxy enables access to Kubernetes services and Pods via `localhost`.

```
C:\Users\yin>kubectl proxy
Starting to serve on 127.0.0.1:8001
```

Figure 9: kubectl proxy running

2.4 Pod Access and Verification

The name of the running Pod was obtained using:

```
kubectl get pods
```

The application was then accessed through the Kubernetes API proxy using the following command:

```
curl http://localhost:8001/api/v1/namespaces/default/pods/
<POD_NAME>:8080/proxy/
```

The successful response confirms that the application is running correctly inside the Pod.

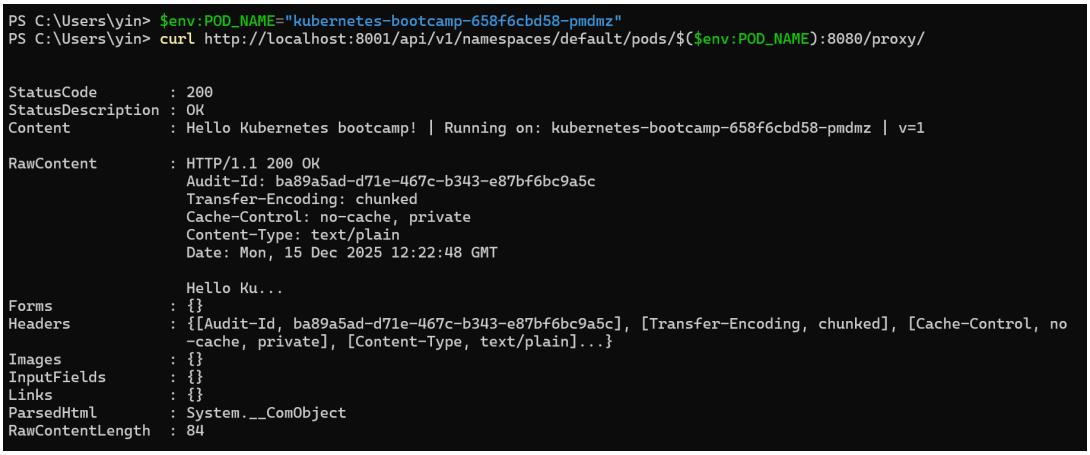

Figure 10: Successful access to the application through API proxy

3.Viewing Pods and Nodes

To verify that the application Pod is running, the following command was executed:

```
kubectl get pods
```

The output shows that the Pod is in the `Running` state, indicating that the application container has been successfully started.

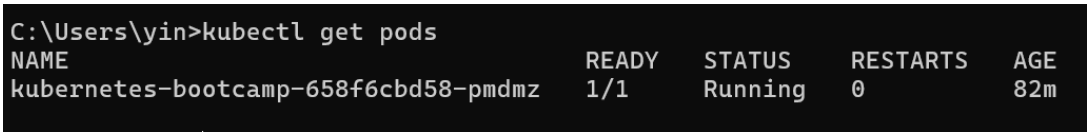

Figure 11: Viewing Running Pods

To view detailed information about the Pod, including its IP address, container image, and events, the following command was used:

```
kubectl describe pod <pod-name>
```

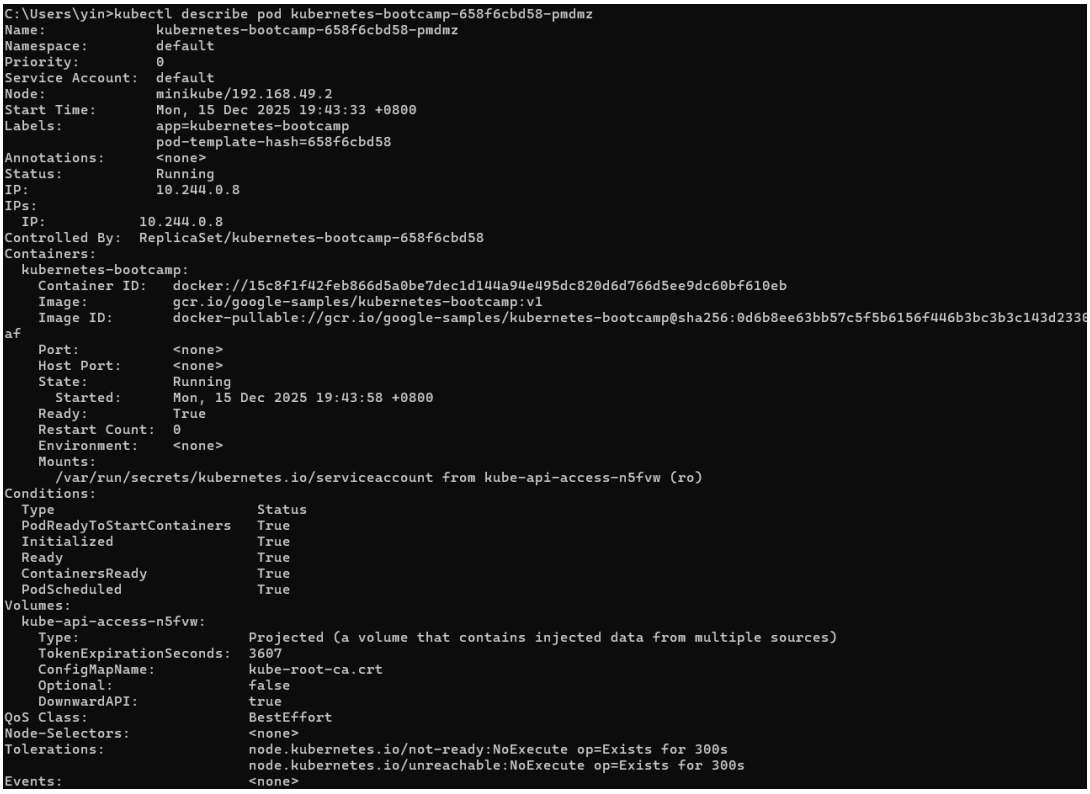

Figure 12: Detailed Pod Information

To inspect the Nodes in the cluster, the following command was executed:

```
kubectl get nodes
```

This confirms that the Pod is scheduled on the Minikube Node.

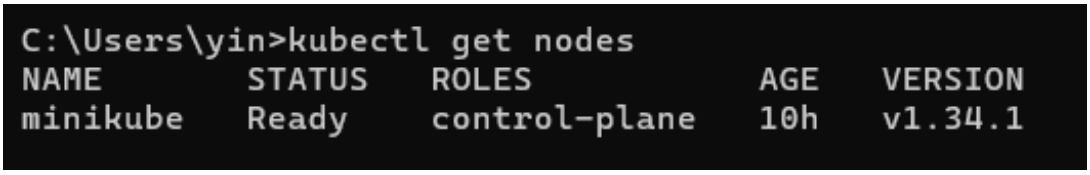

Figure 13: Viewing Cluster Nodes

3.1 Accessing the Pod Using kubectl Proxy

Pods run in an isolated internal network and are not directly accessible from outside the Kubernetes cluster. To access the application for debugging purposes, a proxy was created using the following command in a separate terminal:

```
kubectl proxy
```

The Pod name was retrieved and stored in an environment variable, and the application was accessed through the Kubernetes API using a curl request.

```
curl http://localhost:8001/api/v1/namespaces/default/pods/<pod-name>:8080/proxy/
```

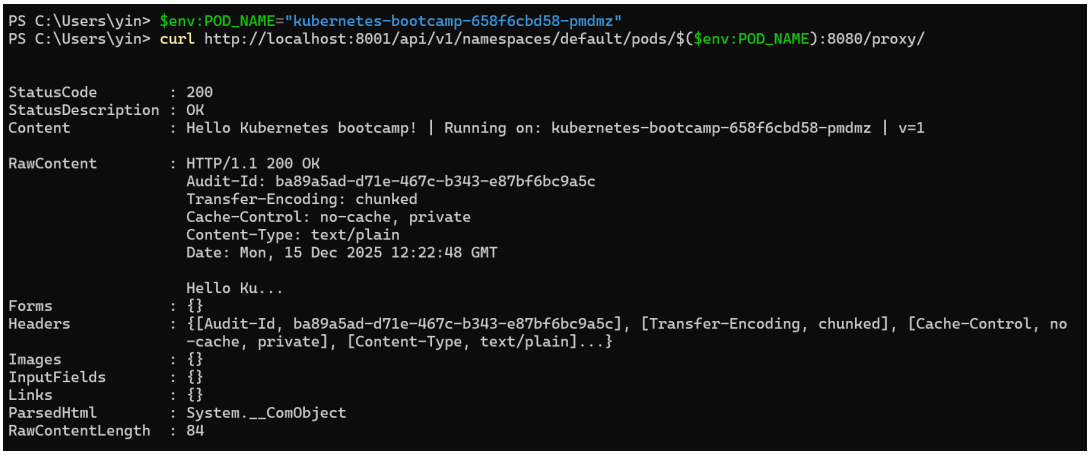

Figure 14: Accessing Pod via kubectl Proxy

3.2 Executing Commands Inside the Pod

To further inspect the running container, commands were executed directly inside the Pod using:

`kubectl exec -it <pod-name> -- bash`

Inside the container, the application was tested by sending a request to the local service:

`curl http://localhost:8080`

This confirmed that the application was running correctly inside the Pod.

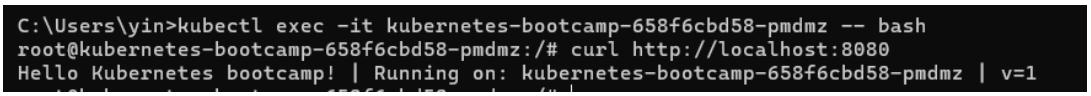

Figure 15: Executing Commands Inside the Pod

4.Using a Service to Expose Your App

4.1 Checking Running Pods

First, the running Pods in the cluster were verified to ensure that the application was successfully deployed.

`kubectl get pods`

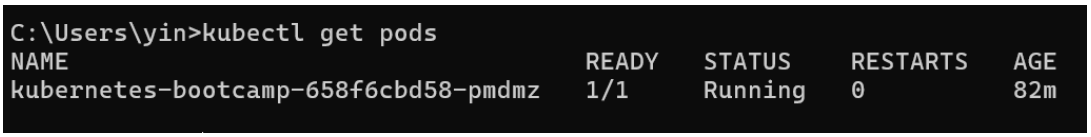

Figure 16: Running Pods in the Cluster

4.2 Creating a Service

To expose the application to external traffic, a NodePort Service was created using the following command:

`kubectl expose deployment/kubernetes-bootcamp --type=NodePort --port 8080`

This command created a Service that forwards external traffic to the application Pod.

Figure 17: Creating a NodePort Service

4.3 Inspecting the Service

The details of the Service were examined to obtain the assigned NodePort:

```
kubectl describe services kubernetes-bootcamp
```

```
C:\Users\yin>kubectl describe services/kubernetes-bootcamp
Name:                     kubernetes-bootcamp
Namespace:                default
Labels:                   app=kubernetes-bootcamp
Annotations:              <none>
Selector:                 app=kubernetes-bootcamp
Type:                     NodePort
IP Family Policy:         SingleStack
IP Families:              IPv4
IP:                       10.111.141.233
IPs:                      10.111.141.233
Port:                     <unset>  8080/TCP
TargetPort:               8080/TCP
NodePort:                 <unset>  30369/TCP
Endpoints:                10.244.0.8:8080
Session Affinity:         None
External Traffic Policy:  Cluster
Internal Traffic Policy:  Cluster
Events:                   <none>
```

Figure 18: Service Details and NodePort

4.4 Accessing the Application

Since the experiment was performed using Minikube, the following command was used to obtain the service access URL:

```
minikube service kubernetes-bootcamp --url
```

The application was successfully accessed through the returned URL, confirming that the Service was working correctly.

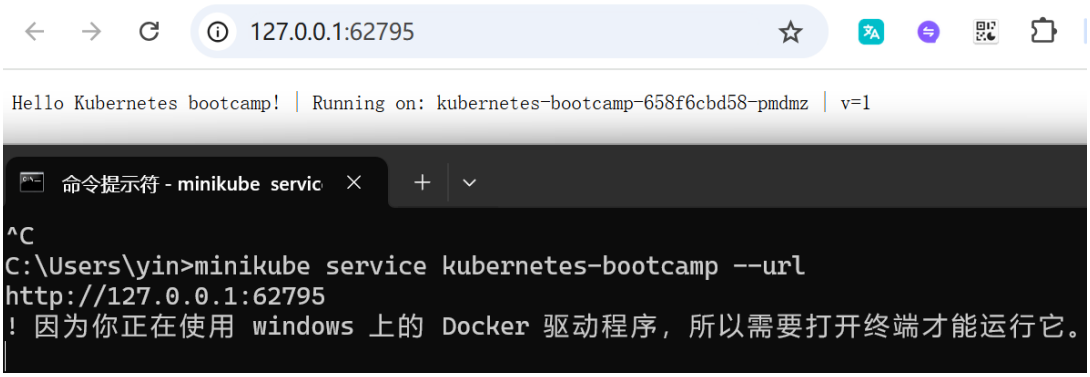

Figure 19: Accessing the Application via Service

5.Running Multiple Instances of Your App

5.1 Initial Deployment State

First, the current state of the Deployment is checked:

```
kubectl get deployments
```

The output shows that the application is running with only one replica.

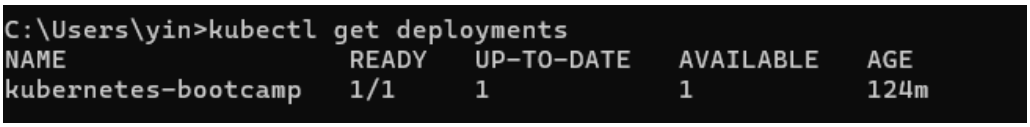

Figure 20: Initial Deployment with a Single Replica

The ReplicaSet created by the Deployment is inspected using:

```
kubectl get rs
```

Figure 21: ReplicaSet Created by the Deployment

5.2 Scaling the Deployment

To handle increased workload, the Deployment is scaled to four replicas:

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

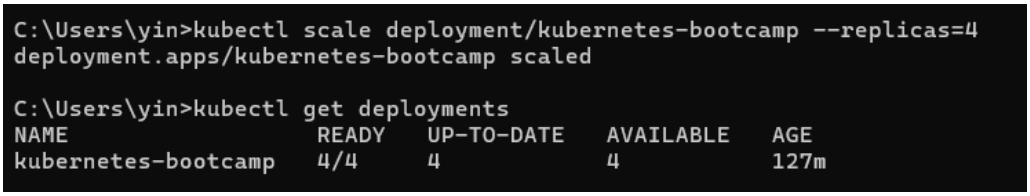

Figure 22: Scaling the Deployment to Four Replicas

The updated Pod status is verified:

```
kubectl get pods -o wide
```

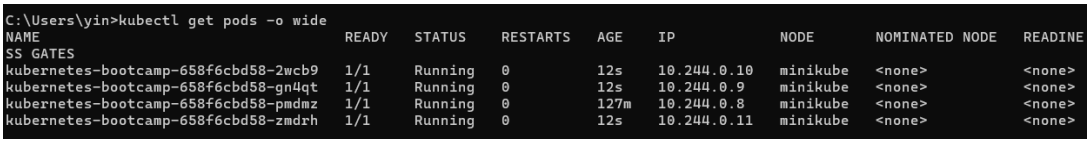

Figure 23: Four Running Pods After Scaling

Each Pod has a unique IP address, confirming that new Pods were successfully created.

5.3 Service Load Balancing Verification

The Service configuration is examined using:

```
kubectl describe service kubernetes-bootcamp
```

```
C:\Users\yin>kubectl describe services kubernetes-bootcamp
Name:                     kubernetes-bootcamp
Namespace:                default
Labels:                   app=kubernetes-bootcamp
Annotations:              <none>
Selector:                 app=kubernetes-bootcamp
Type:                     LoadBalancer
IP Family Policy:         SingleStack
IP Families:              IPv4
IP:                       10.111.204.123
IPs:                      10.111.204.123
Port:                     <unset>  8080/TCP
TargetPort:               8080/TCP
NodePort:                 <unset>  32300/TCP
Endpoints:                10.244.0.8:8080,10.244.0.11:8080,10.244.0.10:8080 + 1 more...
Session Affinity:         None
External Traffic Policy:  Cluster
Internal Traffic Policy:  Cluster
Events:                   <none>
```

Figure 24: Service Configuration for kubernetes-bootcamp

Since Minikube is running with the Docker driver on Windows, the Service is accessed through a temporary local URL:

```
minikube service kubernetes-bootcamp --url
```

Multiple HTTP requests are sent to the application:

```
curl http://127.0.0.1:<PORT>
```

```
C:\Users\yin>curl http://127.0.0.1:65480/
Hello Kubernetes bootcamp! | Running on: kubernetes-bootcamp-658f6cbd58-gn4qt | v=1

C:\Users\yin>curl http://127.0.0.1:65480/
Hello Kubernetes bootcamp! | Running on: kubernetes-bootcamp-658f6cbd58-zmdrh | v=1

C:\Users\yin>curl http://127.0.0.1:65480/
Hello Kubernetes bootcamp! | Running on: kubernetes-bootcamp-658f6cbd58-pmdmz | v=1

C:\Users\yin>curl http://127.0.0.1:65480/
Hello Kubernetes bootcamp! | Running on: kubernetes-bootcamp-658f6cbd58-pmdmz | v=1

C:\Users\yin>curl http://127.0.0.1:65480/
Hello Kubernetes bootcamp! | Running on: kubernetes-bootcamp-658f6cbd58-2wcb9 | v=1

C:\Users\yin>curl http://127.0.0.1:65480/
Hello Kubernetes bootcamp! | Running on: kubernetes-bootcamp-658f6cbd58-pmdmz | v=1

C:\Users\yin>curl http://127.0.0.1:65480/
Hello Kubernetes bootcamp! | Running on: kubernetes-bootcamp-658f6cbd58-gn4qt | v=1

C:\Users\yin>curl http://127.0.0.1:65480/
Hello Kubernetes bootcamp! | Running on: kubernetes-bootcamp-658f6cbd58-pmdmz | v=1
```

Figure 25: Load Balancing Across Different Pods

The responses show different Pod names, demonstrating that the Service correctly distributes traffic among multiple Pods.

5.4Scaling Down the Deployment

After verification, the Deployment is scaled down to two replicas:

```
kubectl scale deployments/kubernetes-bootcamp --replicas=2
```

The Pod list is checked again:

```
kubectl get pods -o wide
```

Figure 26: Two Pods Remaining After Scaling Down

This confirms that Kubernetes successfully terminated excess Pods.

6.Performing a Rolling Update

6.1 Checking the Current Deployment Status

The existing Deployment and Pods were first inspected to verify that the application was running correctly.

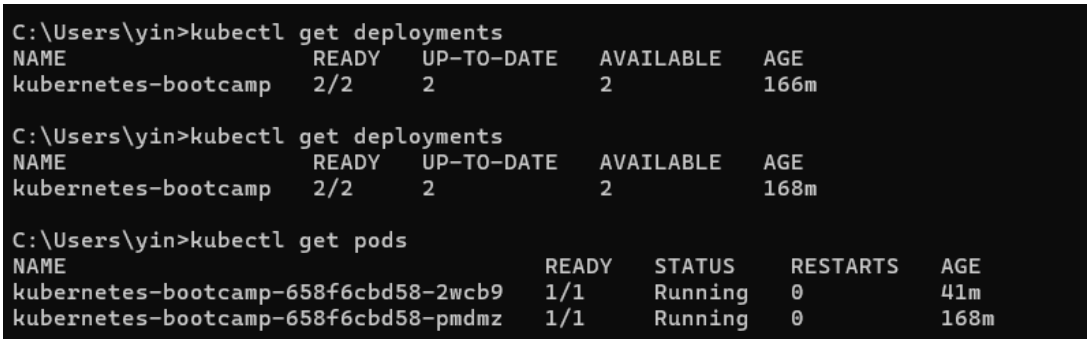

Figure 27: Deployment status before rolling update

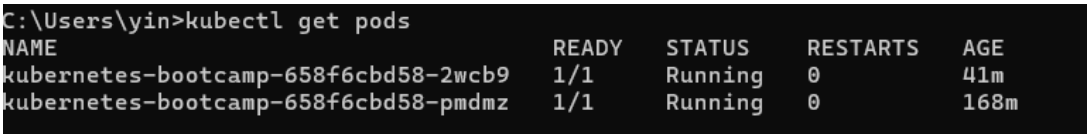

Figure 28: Pods running before rolling update (version v2)

6.2 Verifying the Current Application Version

The image version of the running Pods was verified using the `kubectl describe pods` command.

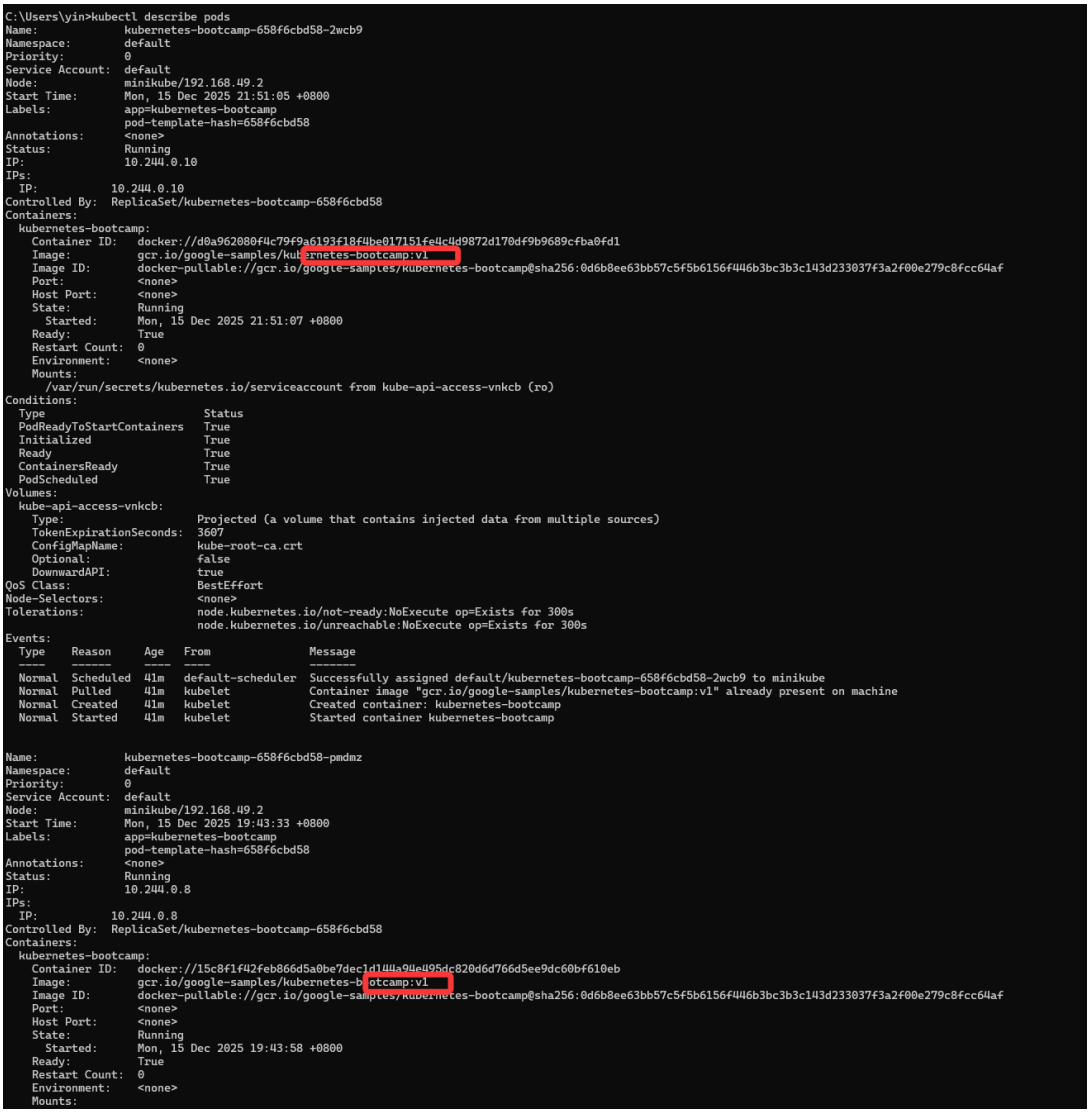

Figure 29: Pod image version before update (v2)

6.3 Updating the Application Version

The Deployment was updated to use a new image version using the following command:

```
kubectl set image deployments/kubernetes-bootcamp
kubernetes-bootcamp=gcr.io/google-samples/kubernetes-bootcamp:v10
```

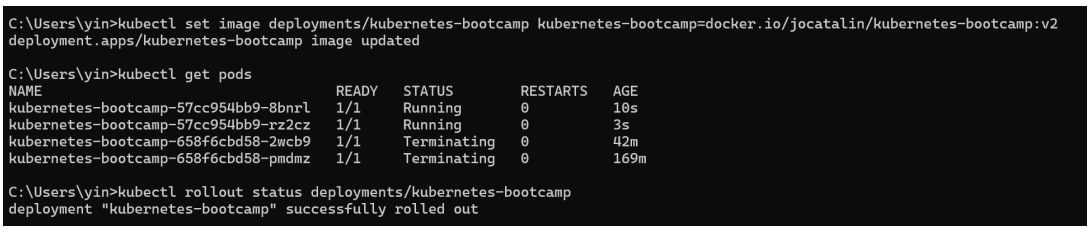

Figure 30: Rolling update process after updating the image version

6.4 Updating the Application to an Invalid Version

The Deployment was updated to use a non-existent image version (v10) in order to observe Kubernetes rolling update behavior under failure conditions.

```
PS C:\Users\yin> kubectl set image deployments/kubernetes-bootcamp `
>> kubernetes-bootcamp=gcr.io/google-samples/kubernetes-bootcamp:v10
deployment.apps/kubernetes-bootcamp image updated
PS C:\Users\yin> kubectl get deployments
NAME                  READY   UP-TO-DATE   AVAILABLE   AGE
kubernetes-bootcamp   2/2     1            2           177m
PS C:\Users\yin> kubectl get deployments
NAME                  READY   UP-TO-DATE   AVAILABLE   AGE
kubernetes-bootcamp   2/2     1            2           177m
PS C:\Users\yin> kubectl get pods
NAME                                   READY   STATUS             RESTARTS   AGE
kubernetes-bootcamp-57cc954bb9-8bnrl   1/1     Running            0          8m15s
kubernetes-bootcamp-57cc954bb9-rz2cz   1/1     Running            0          8m8s
kubernetes-bootcamp-677ff875c4-ghch5   0/1     ImagePullBackOff   0          24s
```

Figure 31: Pods status after updating to invalid image version (v10)

6.5 Analyzing the Update Failure

Detailed information about the failure was obtained using the `kubectl describe pod` command. The output showed that the image could not be pulled from the container registry.

```
PS C:\Users\yin> kubectl get pods
NAME                                   READY   STATUS             RESTARTS   AGE
kubernetes-bootcamp-57cc954bb9-8bnrl   1/1     Running            0          8m15s
kubernetes-bootcamp-57cc954bb9-rz2cz   1/1     Running            0          8m8s
kubernetes-bootcamp-677ff875c4-ghch5   0/1     ImagePullBackOff   0          24s
```

Figure 32: Pod events showing ImagePullBackOff error

6.6 Rolling Back to the Previous Stable Version

The Deployment was rolled back to the last known stable version using the `kubectl rollout undo` command.

```
PS C:\Users\yin> kubectl rollout undo deployments/kubernetes-bootcamp
deployment.apps/kubernetes-bootcamp rolled back
```

Figure 33: Rolling back the Deployment

6.7 Verifying the Rollback Result

After the rollback, the Pods were inspected again to ensure that the application returned to a stable state.

```
PS C:\Users\yin> kubectl get pods
NAME                                   READY   STATUS    RESTARTS   AGE
kubernetes-bootcamp-57cc954bb9-8bnrl   1/1     Running   0          8m41s
kubernetes-bootcamp-57cc954bb9-rz2cz   1/1     Running   0          8m34s
```

Figure 34: Pods running after rollback

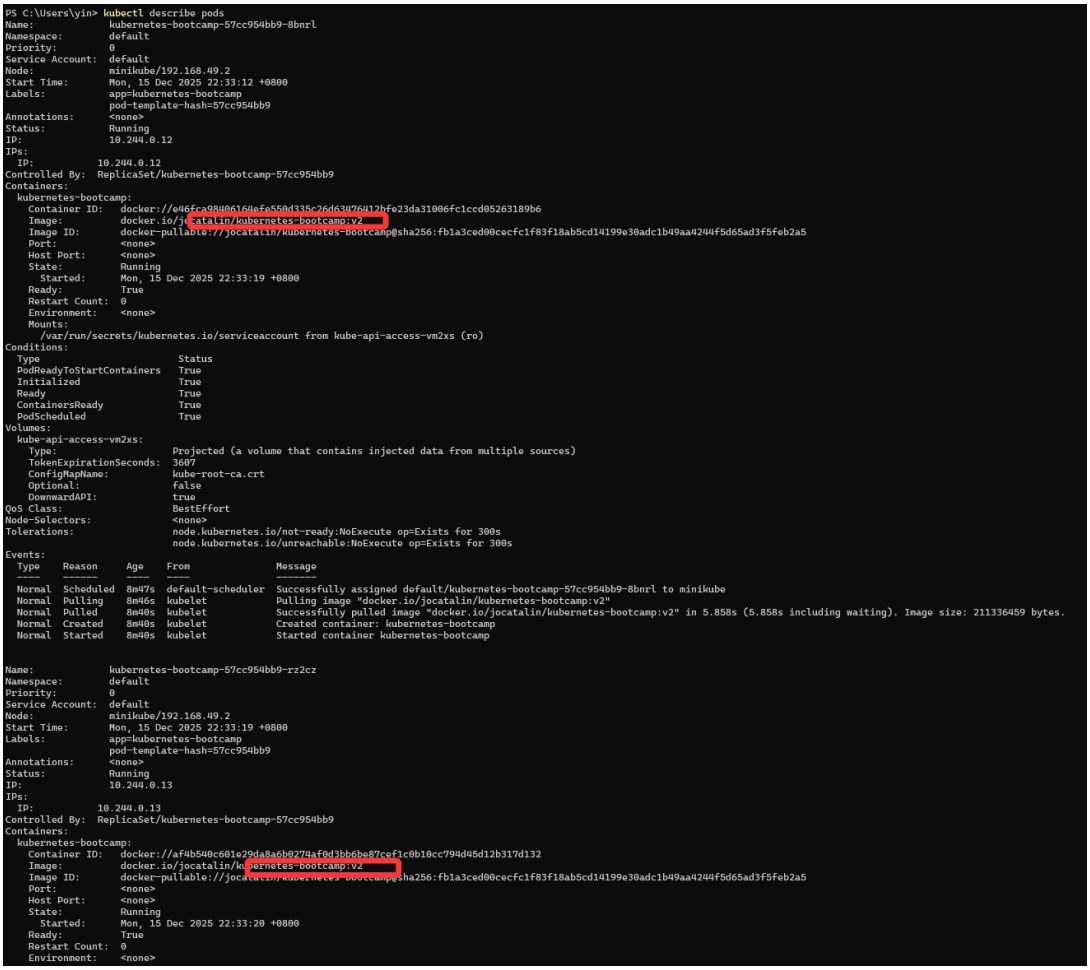

Figure 35: Pod image version after rollback (v2)

IV. Conclusion

In this experiment, a local Kubernetes cluster was successfully deployed and managed using Minikube. Through a series of practical operations, containerized applications were deployed using Kubernetes Deployments, and their runtime status was verified by inspecting Pods, Nodes, and application logs. The experiment demonstrated how applications can be accessed internally via `kubectl proxy` and externally through Kubernetes Services, including NodePort and LoadBalancer types.

Furthermore, application scalability was validated by dynamically adjusting the number of replicas and observing service-level load balancing across multiple Pods. Rolling update and rollback mechanisms were also tested, including failure scenarios caused by invalid image versions, which highlighted Kubernetes' ability to automatically detect errors and recover to a stable state.

Overall, this experiment provided hands-on experience with core Kubernetes features such as deployment management, service exposure, scalability, and fault tolerance, strengthening the understanding of container orchestration concepts and practical cluster operations.