Rashika Singh
IST 615

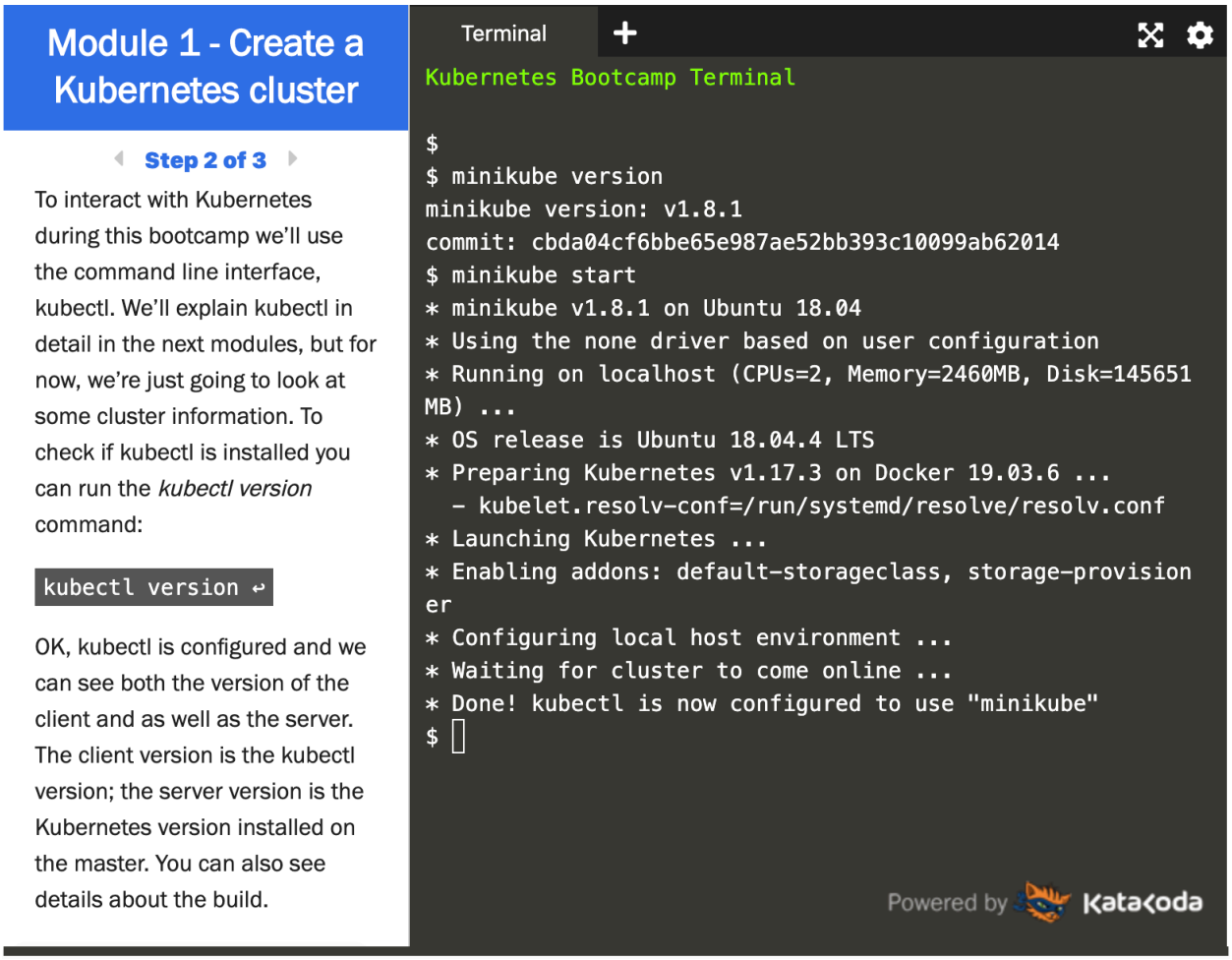
Lab 4
Kubernetes

Kubernetes allows you to run containerization application run and helps to find the resources they need.

Module 1 – Create a Kubernetes Cluster

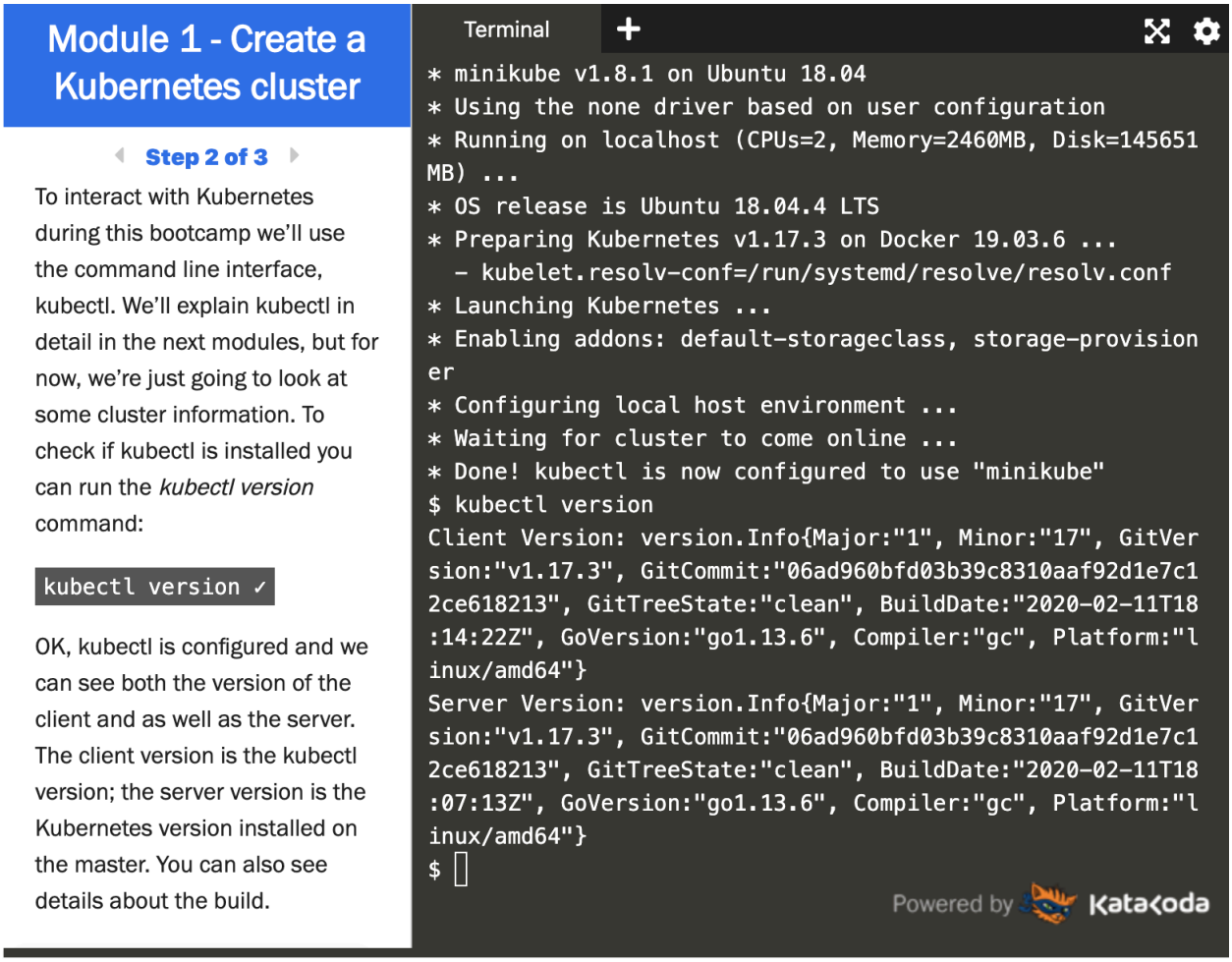
- **Cluster up and running**

Minikube is a tool that makes it easy to learn Kubernetes locally. It runs a lightweight Kubernetes implementation that creates a VM and runs a single-node Kubernetes cluster. It allows you to deploy containerized applications to a cluster rather than individual machines, Kurbernetes needs a single command to start using 'minikube start'. Now a Kubernetes cluster runs in the online terminal and Minikube started a virtual machine for you, and a Kubernetes cluster is running in that VM.



- **Cluster version**

Kubectl allows you to control Kubernetes and perform every possible Kubernetes operation. It is a command line interface to interact with Kubernetes. We can check that it is installed correctly by using 'kubectl version'. Now we can see both the client and server version. The client version is the kubectl version; the server version is the Kubernetes version installed on the master.



- **Cluster details**

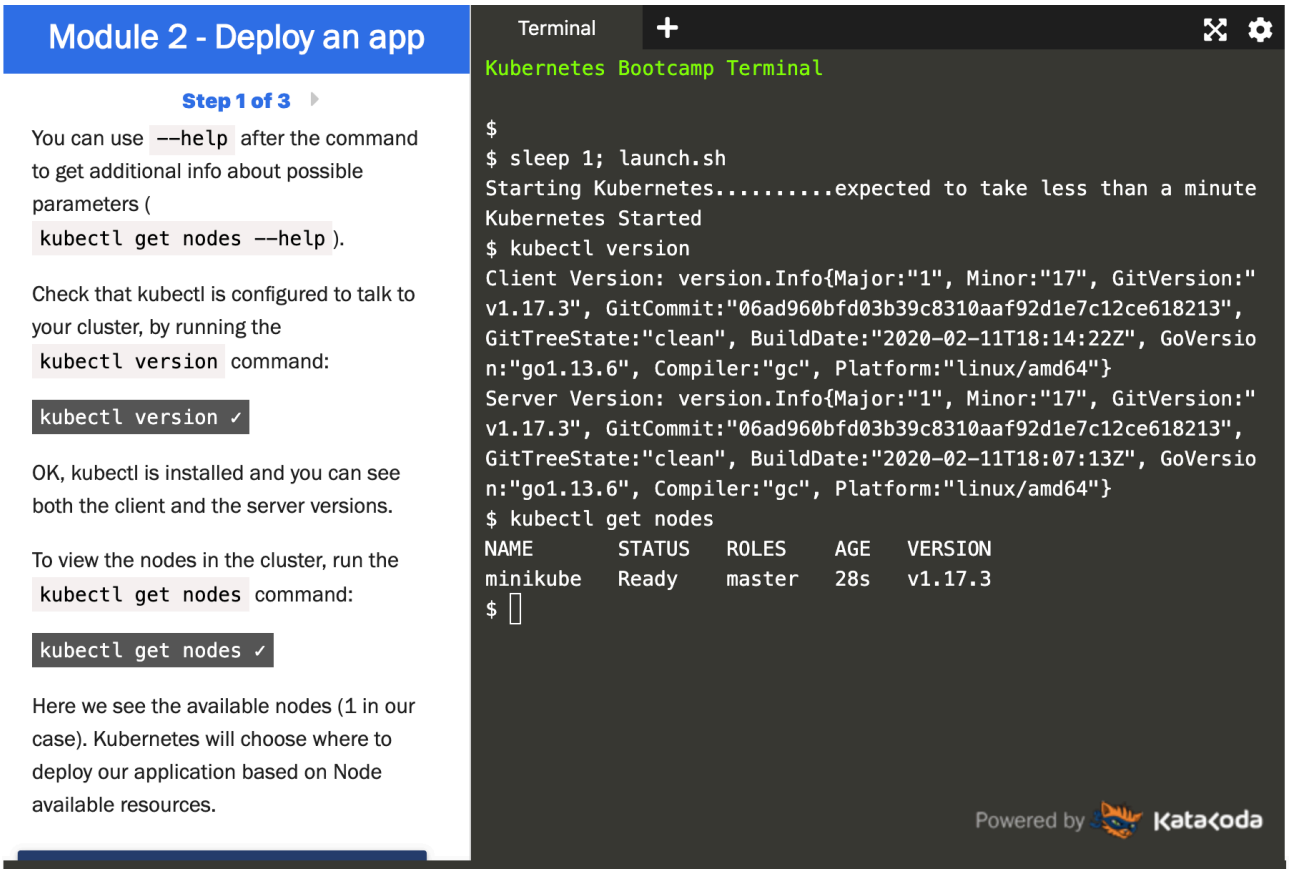
We can see the cluster details using the 'kubectl cluster-info' which shows the details about the running master and the location. The 'kubectl get nodes' shows all the nodes that can be used to host the application. We can see that we have one node running with its status as ready which means it is ready to accept applications for deployment. The role is master which shows that it is not a worker node.

Module 2 – Deploy an App

Once we have a running Kubernetes cluster, you can deploy the containerized applications on top of it. To do so, we create a Kubernetes Deployment configuration which instructs Kubernetes on how to create and update instances of the application.

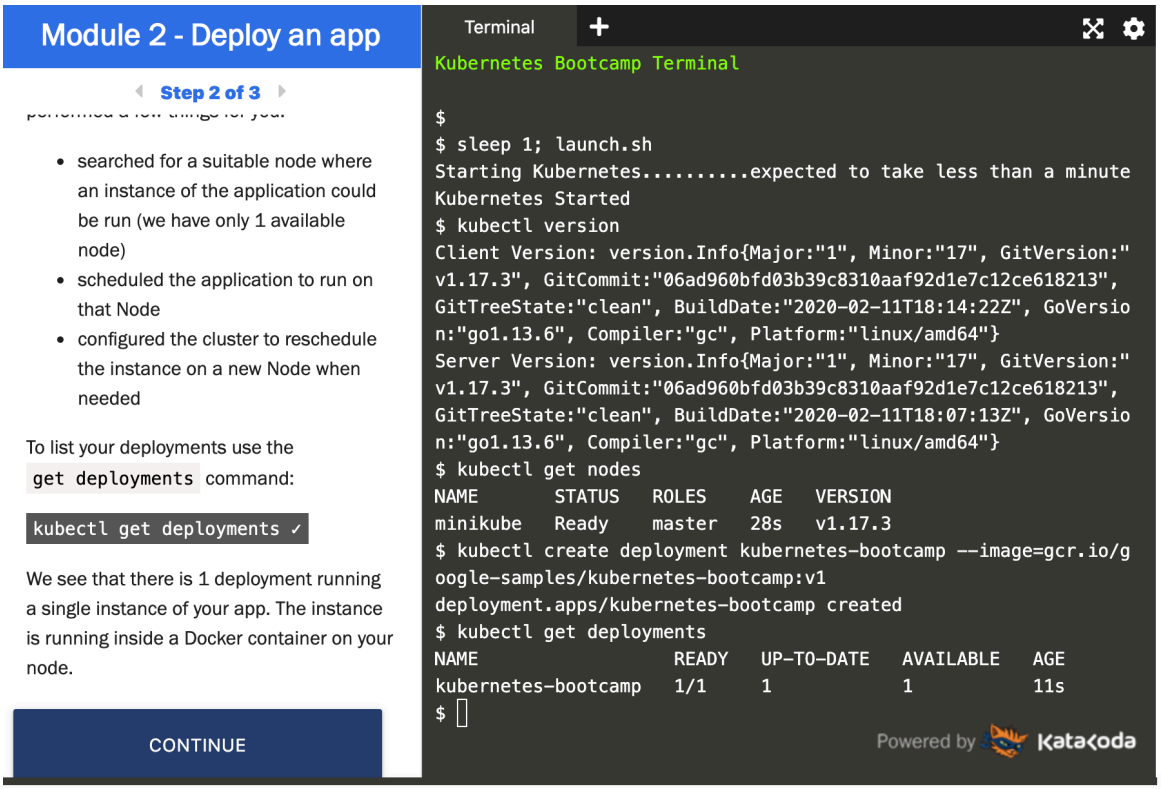
- **kubectl basics**

First, we check using Kubernetes is installed using ‘kubectl version’ and we can see both the client and server versions. To view the nodes in the cluster, we use the ‘kubectl get nodes’. We can see that there is one node running with its status as ready which means it is ready to accept applications for deployment. The role is master which shows that it is not a worker node.



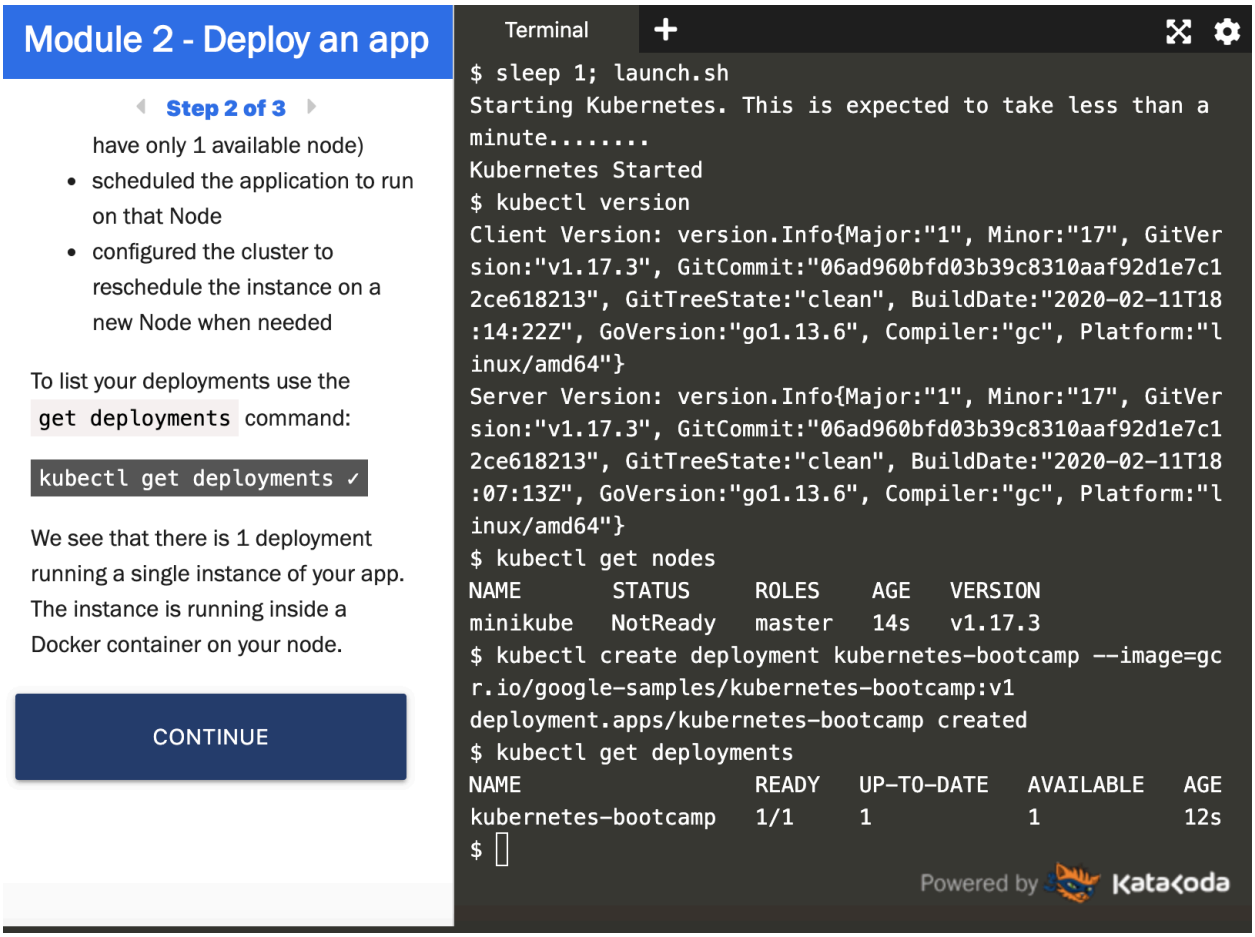
- **Deploy our app**

We create our first application on Kubernetes with ‘kubectl create deployment’ command where we need to provide the deployment name and app image location. In this case, we provide the deployment name as ‘kubernetes-bootcamp’ and we can also specify the port on which we need to run the app. The command searched a suitable node where an instance of the application could be run, scheduled the app to run on that node and configured the cluster to reschedule the instance on a new node.

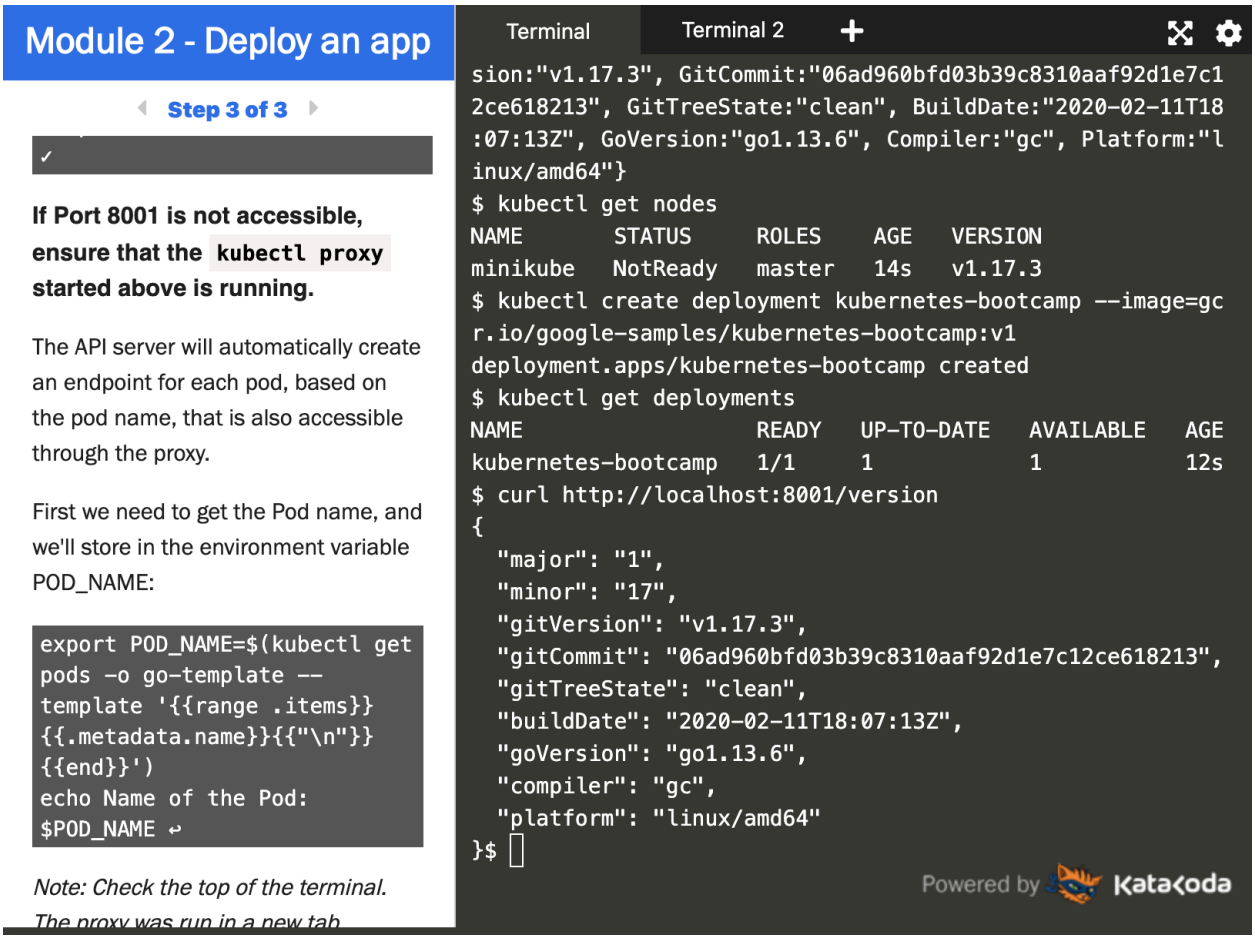


- **View our app**

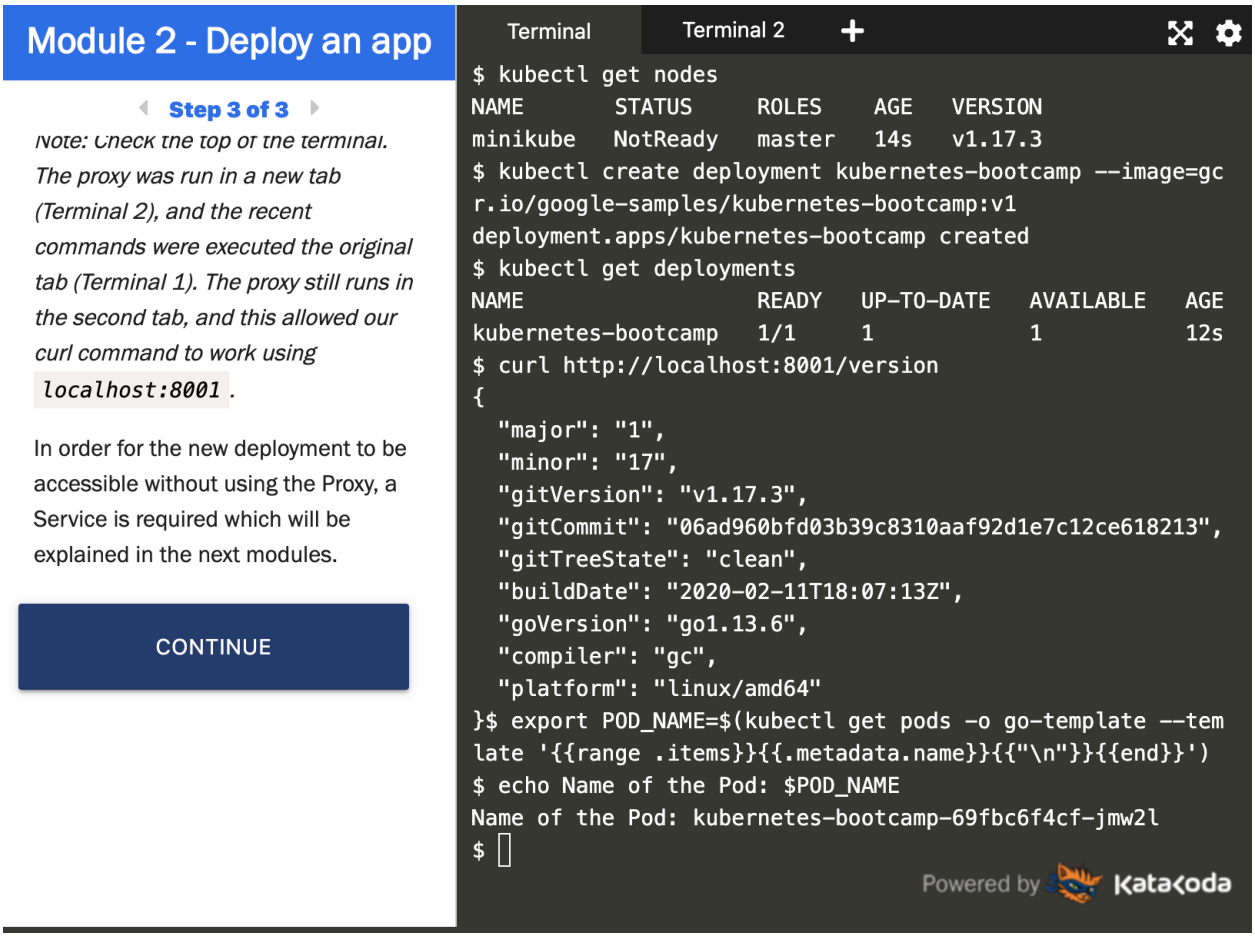
We can see the deployments using the ‘kubectl get deployments’. This shows that one deployment is running on a single instance of the app. Ths instance is running inside a Docker container on the node.



Pods are the smallest, most basic deployable objects in Kubernetes. It represents a single instance of a running process in the cluster. Pods that are running inside Kubernetes are running on a private, isolated network. By default, they are visible from other pods and services within the same Kubernetes cluster, but not outside that network. The kubectl command will create a proxy that will forward communications into the cluster-wide, private network. The proxy enables direct access to the API from the terminal. We can see all APIs hosted through the proxy endpoint.



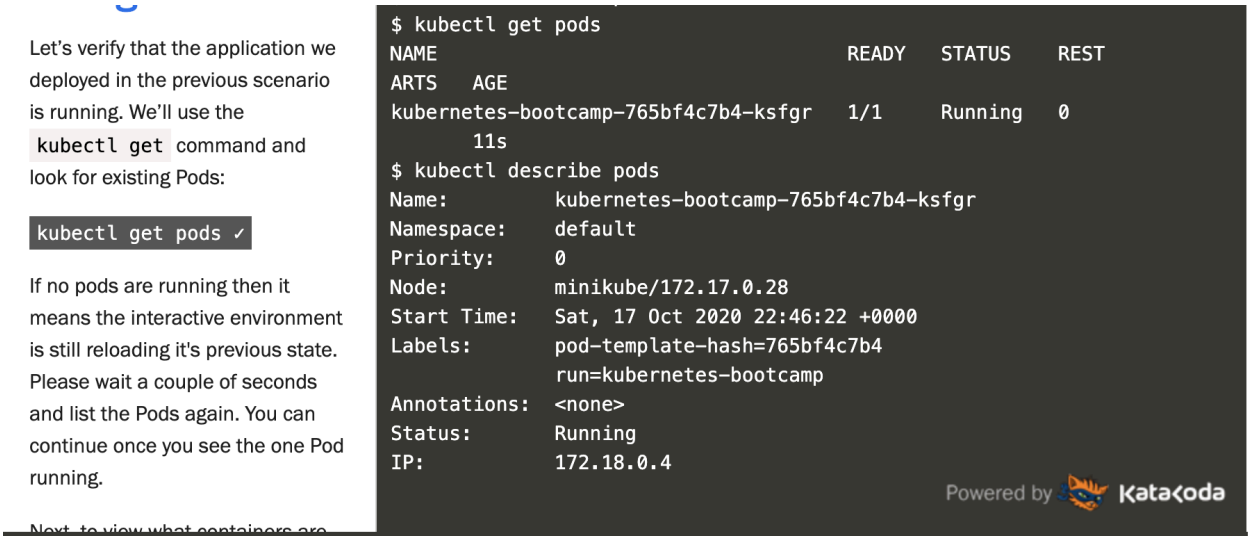
We can query the version directly through the API using the curl command. Curl allows you to send or receive data using URL, here we used port 800. If it is not accessible, we can check that the 'kubectl proxy' is running. The API server will create an endpoint for each pod based on its name and is accessible through the proxy. We need to get the Pod name and store it in an environment variable POD_NAME which can be used to connect to the app running in the pod. We can see in the snippet that a connection to a pod is established.



Module 3 – Explore Your APP

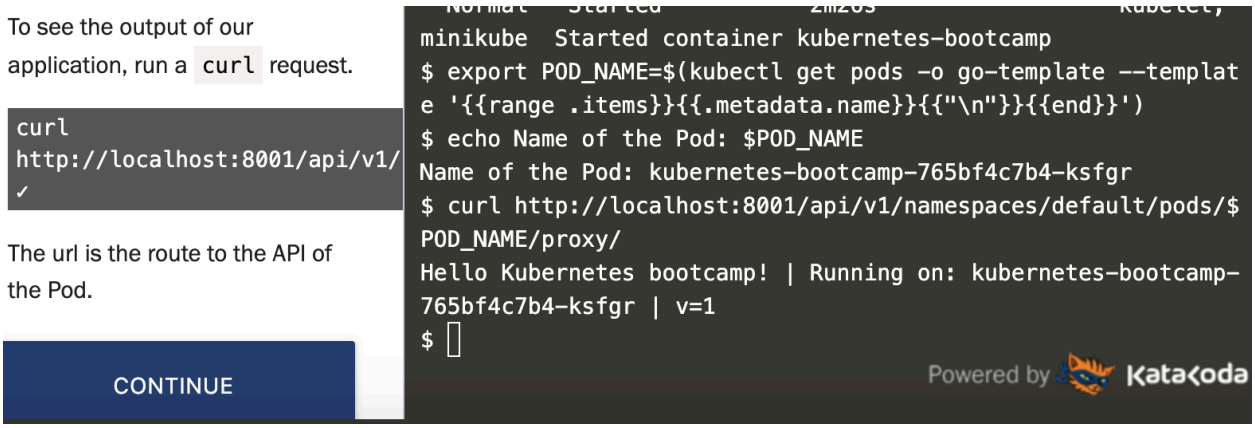
- **Check application configuration**

Pods are the smallest, most basic deployable objects in Kubernetes. It represents a single instance of a running process in the cluster. We use ‘kubectl get pods’ to look for existing pods and use the ‘kubectl describe pods’ to describe the pods. We can see details about the Pod’s container such as IP address, the ports used, and a list of events related to the lifecycles of the Pod.



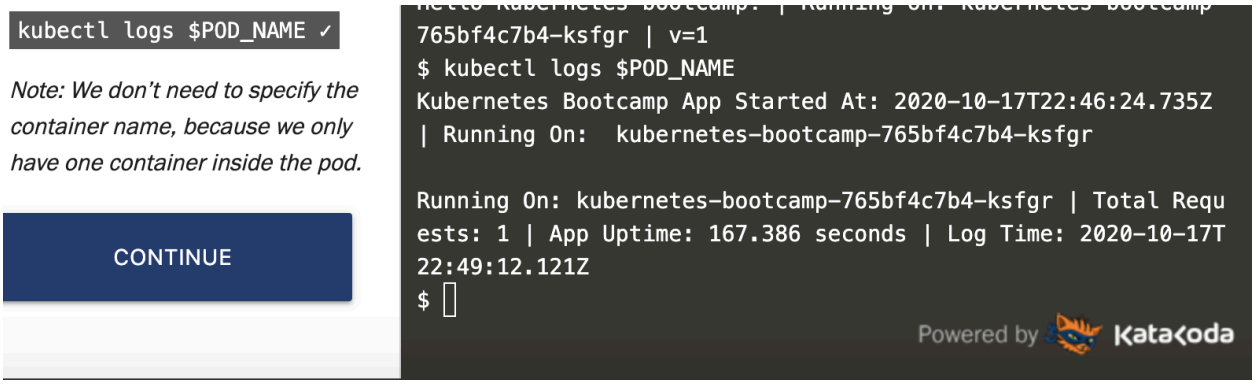
- **Show the app in the terminal**

As seen above, we run 'kubectl proxy' to run a proxy. Now we query the pod directly through the proxy and get the pod name and store it in the POD_NAME environment variable. We run a curl request to see the output of the application. Curl allows you to send or receive data using URL.



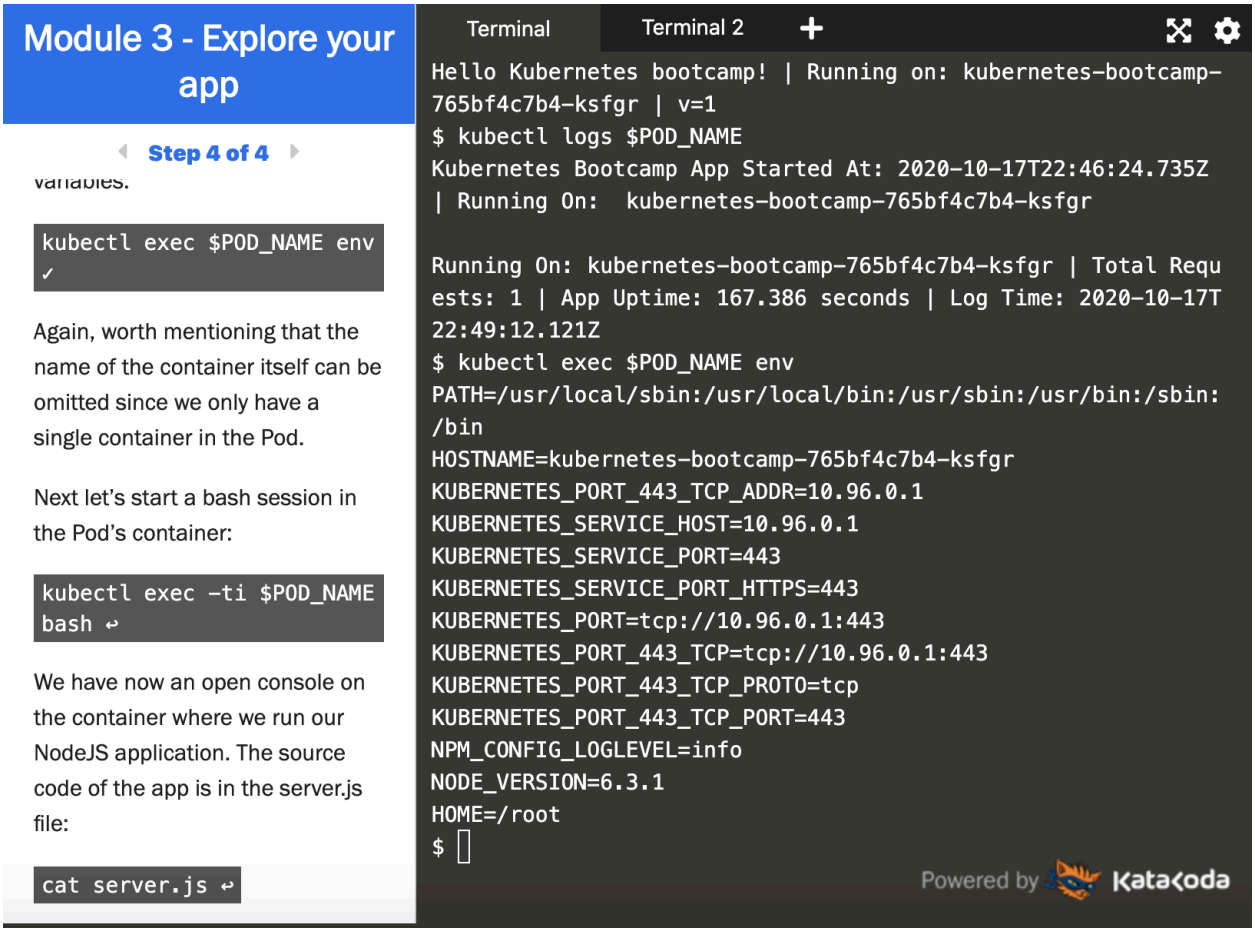
- **View the container logs**

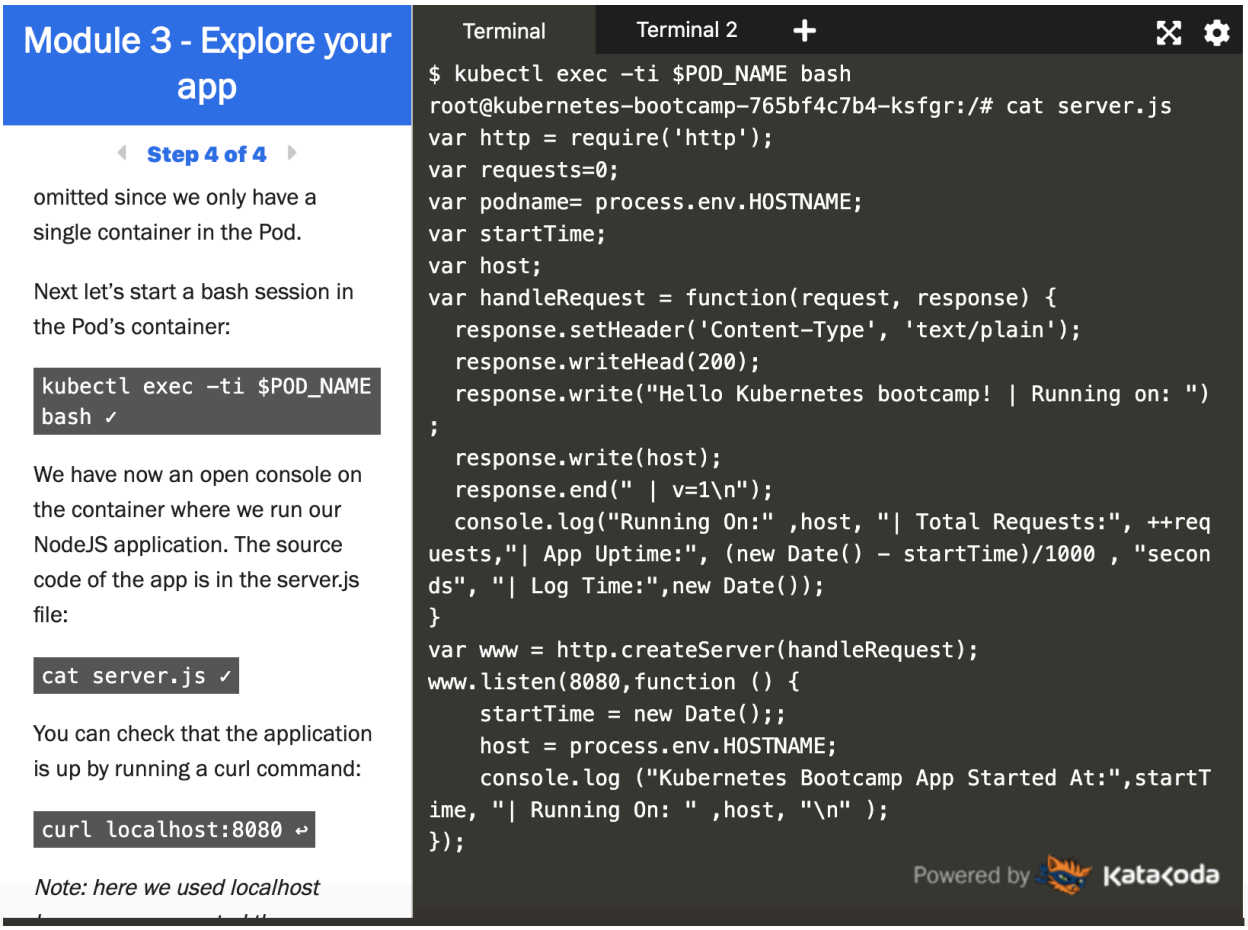
We can retrieve logs of the container within the pods using 'kubectl logs', these are anything that the application would normally send to STDOUT. We can see the details for the logs in the snippet.



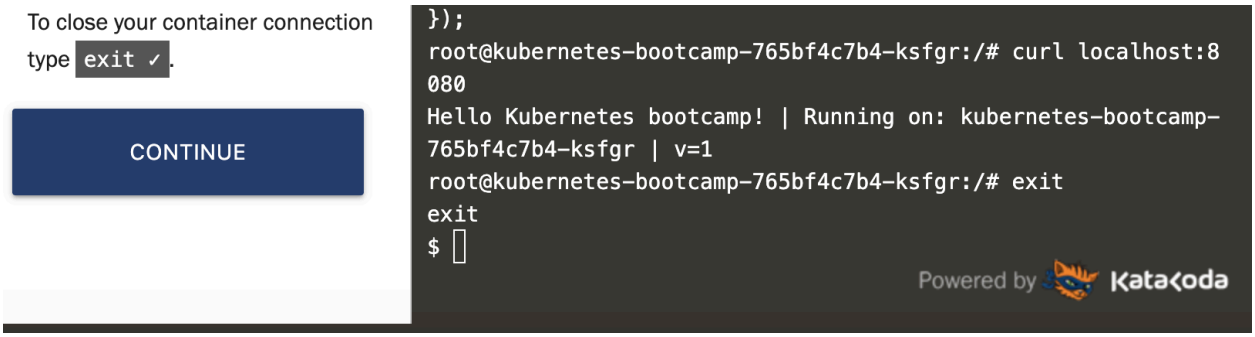
- **Executing command on the container**

Once the pod is running, we can execute commands directly on the container. We use the exec command with the pod name and list the environment variables. We can start a bash session using the Pod's container using the exec command. Then we run the NodeJS application as the source code of the app is in the server.js file. We can check that the app is running using the curl command.





We can close the container connection using 'exit' commnad.

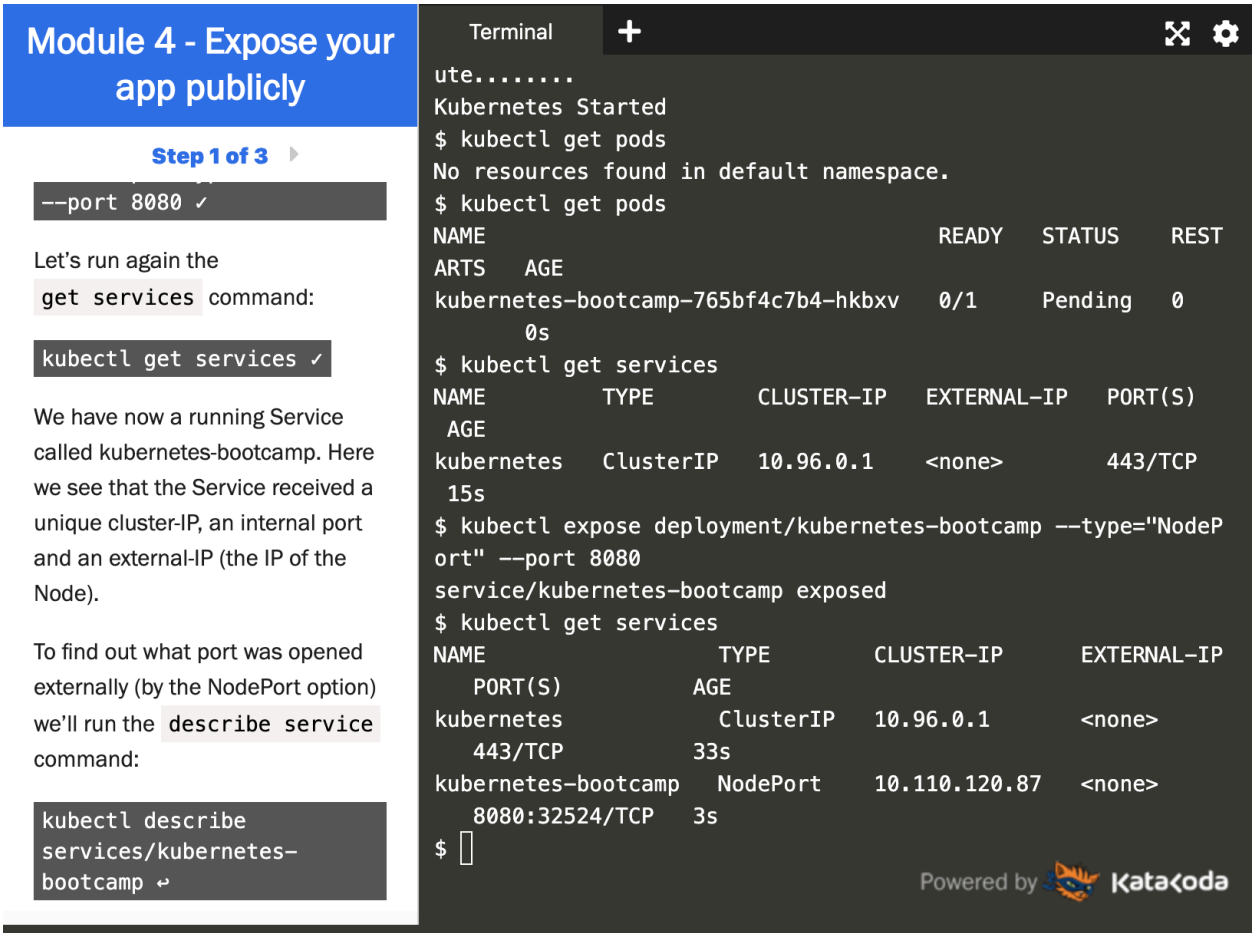


Module 4 – Expose Your App Publicly

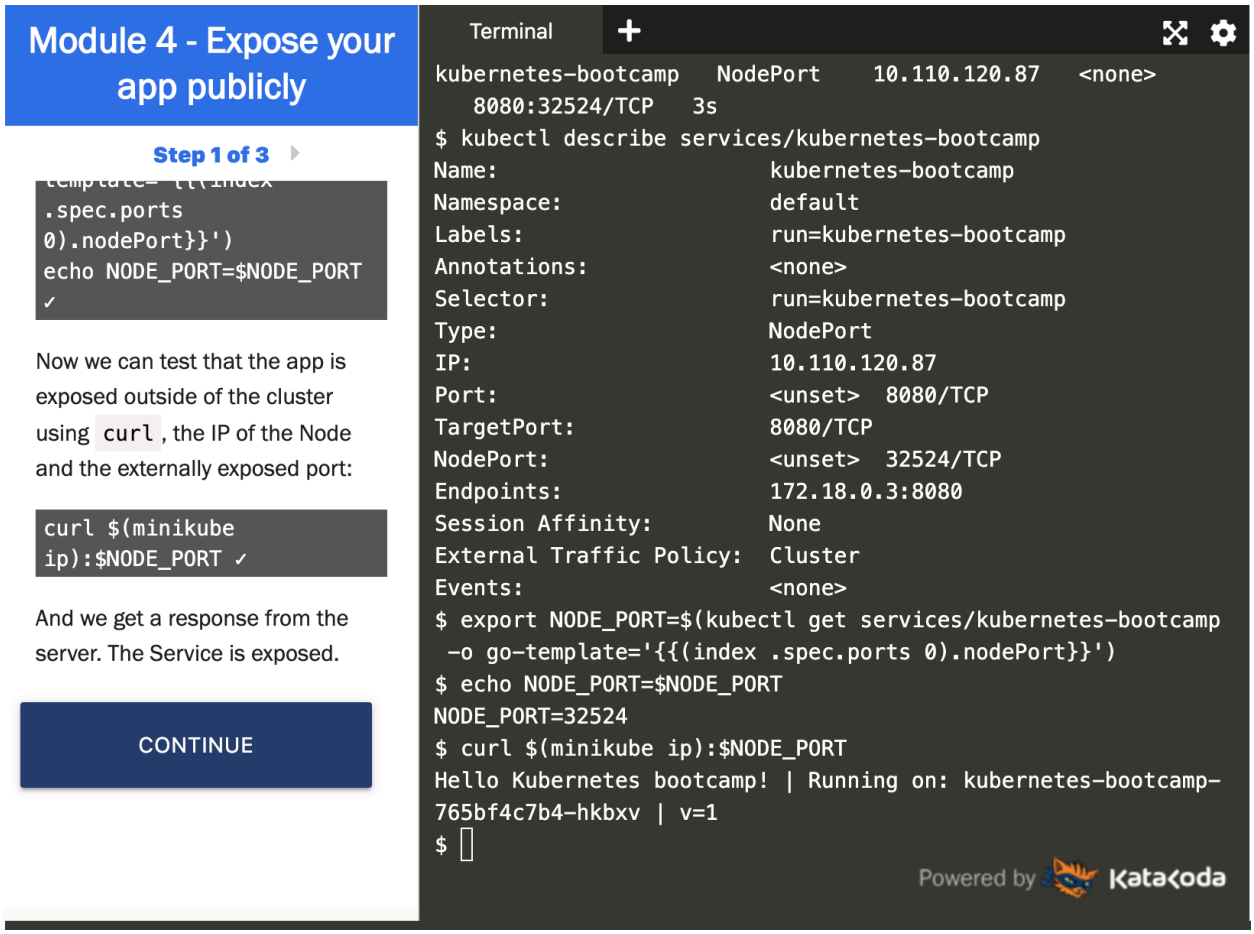
- **Create a new service**

The containers are launched in pods which are not public, so now we will need to expose the app publicly. We can see the existing pods using the 'kubectl get pods' command which initially does not show any, this is because the resources are not yet loaded completely. If we try again, we can see our pod is running. We can see the current services from the cluster using 'kubectl

get services'. We see a default service called Kubernetes which is present by default when minikube starts the cluster. We need to create a new service and cxpose it to external traffic using NodePort parameter. We use the expose deployment with the nodeport. Now if we try get services again, we can see our service too.



We can describe the services and create an environment variable called NODE_PORT which has the value of the Node port assigned to it. We use the curl command to check if the app is exposed publicly, we can access the app using the IP address and the port.



- **Using labels**

We can see the list of pods and services using the commands above, we store the pod name in a variable. We can apply a new label using the command 'kubectl label pod $POD_NAME app=v1' where we can specify the object type, object name and new label. The name is stored in $POD_NAME which is replaced by app=v1. We can check it with the describe pod command. The new label is attached to the pod

Module 4 - Expose your app publicly

Step 2 of 3

Step 2: Using labels

The Deployment created automatically a label for our Pod. With `describe deployment` command you can see the name of the label:

```
kubectl describe deployment ✓
```

Let's use this label to query our list of Pods. We'll use the `kubectl get pods` command with -l as a parameter, followed by the label values:

```
kubectl get pods -l run=kubernetes-bootcamp ↵
```

Terminal

```
$ kubectl describe deployment
Name:                   kubernetes-bootcamp
Namespace:              default
CreationTimestamp:      Sat, 17 Oct 2020 22:57:51 +0000
Labels:                 run=kubernetes-bootcamp
Annotations:            deployment.kubernetes.io/revision: 1
Selector:               run=kubernetes-bootcamp
Replicas:               1 desired | 1 updated | 1 total | 1 a
vailable | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  run=kubernetes-bootcamp
  Containers:
   kubernetes-bootcamp:
    Image:        gcr.io/google-samples/kubernetes-bootcamp:v
1
    Port:         8080/TCP
    Host Port:    0/TCP
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Conditions:
  Type           Status  Reason
```

Powered by Katacoda

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

```
kubectl label pod $POD_NAME app=v1 ✓
```

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:

```
kubectl describe pods $POD_NAME ↵
```

We see here that the label is attached now to our Pod. And we

```
$ kubectl get pods -l run=kubernetes-bootcamp
NAME                                  READY   STATUS    RESTA
RTS   AGE
kubernetes-bootcamp-765bf4c7b4-hkbxv   1/1     Running   0
      3m23s
$ kubectl get services -l run=kubernetes-bootcamp
NAME                  TYPE       CLUSTER-IP      EXTERNAL-IP
 PORT(S)          AGE
kubernetes-bootcamp   NodePort   10.110.120.87   <none>
 8080:32524/TCP   3m9s
$ export POD_NAME=$(kubectl get pods -o go-template --template
 '{{range .items}}{{.metadata.name}}{{"\n"}}{{end}}')
$ echo Name of the Pod: $POD_NAME
Name of the Pod: kubernetes-bootcamp-765bf4c7b4-hkbxv
$ kubectl label pod $POD_NAME app=v1
pod/kubernetes-bootcamp-765bf4c7b4-hkbxv labeled
$
```

Powered by Katacoda

```
$ kubectl get pods -l app=v1
NAME                                  READY   STATUS    RESTA
RTS   AGE
kubernetes-bootcamp-765bf4c7b4-hkbxv   1/1     Running   0
      4m38s
$
```

- **Deleting a service**

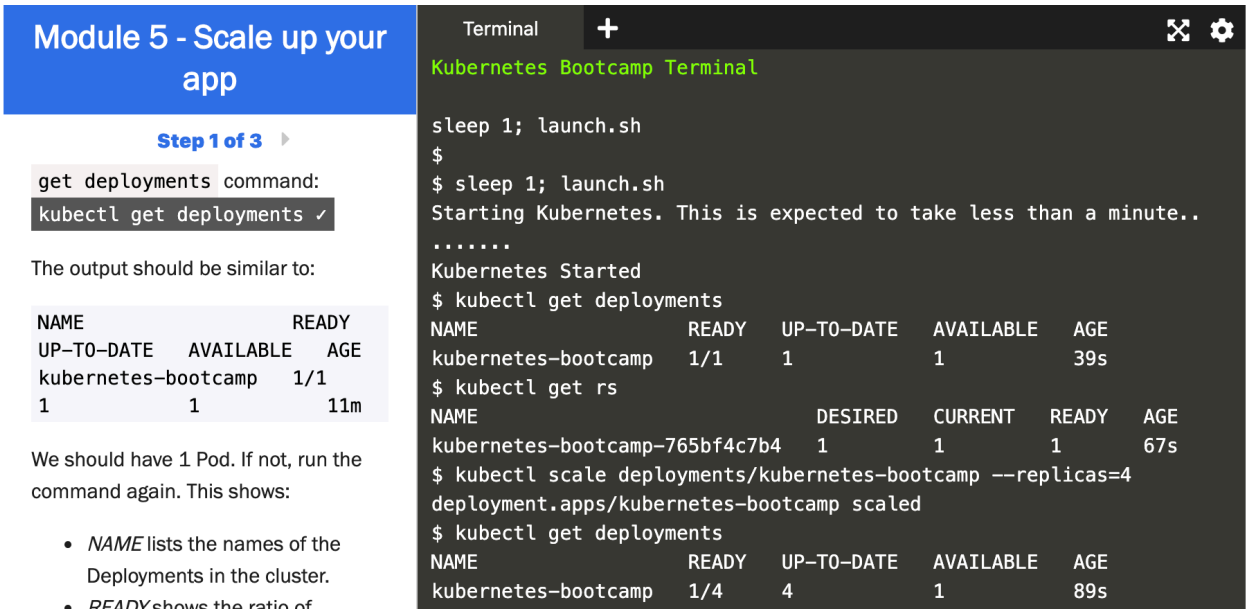
We can delete services using the delete service command and we can confirm it is deleted by using get services. You can see that the service is removed. We can check if it is exposed to outside using the curl command which shows it is not reachable anymore from outside the cluster. We can confirm that the app is running with a curl inside the pod.

```
container "kubernetes-bootcamp"
$ kubectl get pods -l app=v1
NAME                                  READY   STATUS    RESTA
RTS   AGE
kubernetes-bootcamp-765bf4c7b4-hkbxv   1/1     Running   0
      4m38s
$ kubectl delete service -l run=kubernetes-bootcamp
service "kubernetes-bootcamp" deleted
$ kubectl get services
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)
AGE
kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP
5m40s
$ curl $(minikube ip):$NODE_PORT
curl: (7) Failed to connect to 172.17.0.17 port 32524: Connect
ion refused
$ kubectl exec -ti $POD_NAME curl localhost:8080
Hello Kubernetes bootcamp! | Running on: kubernetes-bootcamp-7
65bf4c7b4-hkbxv | v=1
$
```

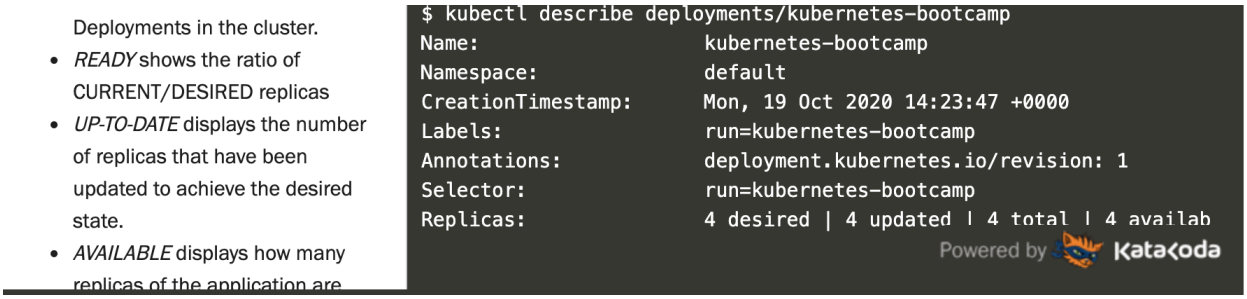
Module 5 – Scale Your App

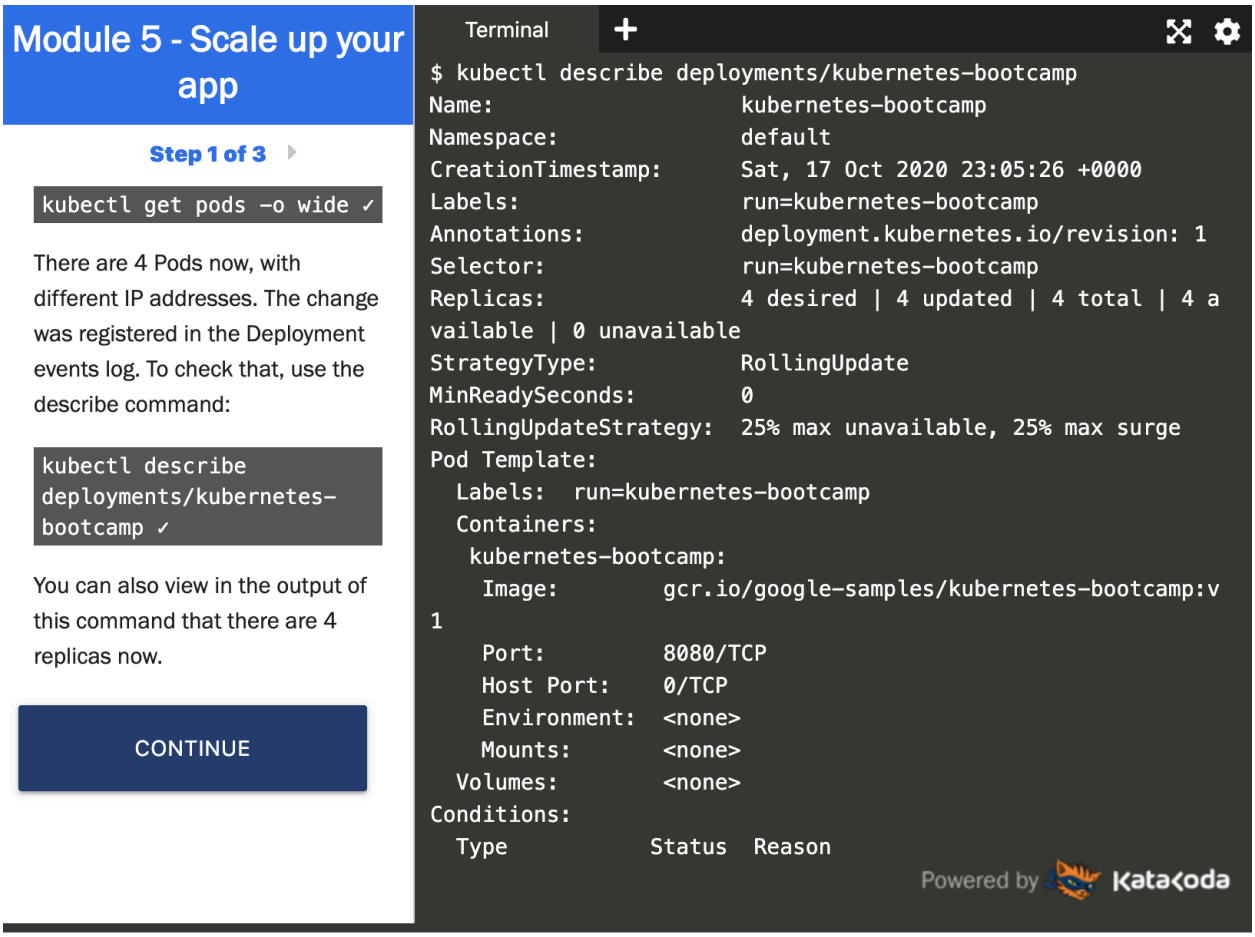
- **Scaling a deployment**

We can get a list of deployments using ‘kubectl get deployments’ command. This shows that we have 1 pod which is ready, up-to-date and available. The ready is the ratio of current/ desired replicas and the age shows the amount of time that the application is running. The desired is the number of replicas which you define when you create the deployment and the current shows how many replicas are currently running. A replica set ensures that a specified number of pod replicas are running at any given time. This ensures that a specified number of identical pods are available. We can check this by ‘kubectl get rs’. The name is followed by random string which is generated. Next, we scale the deployment to 4 replicas using the kubectl scale command where we specify the name, type and the number of instances; here we put 4.



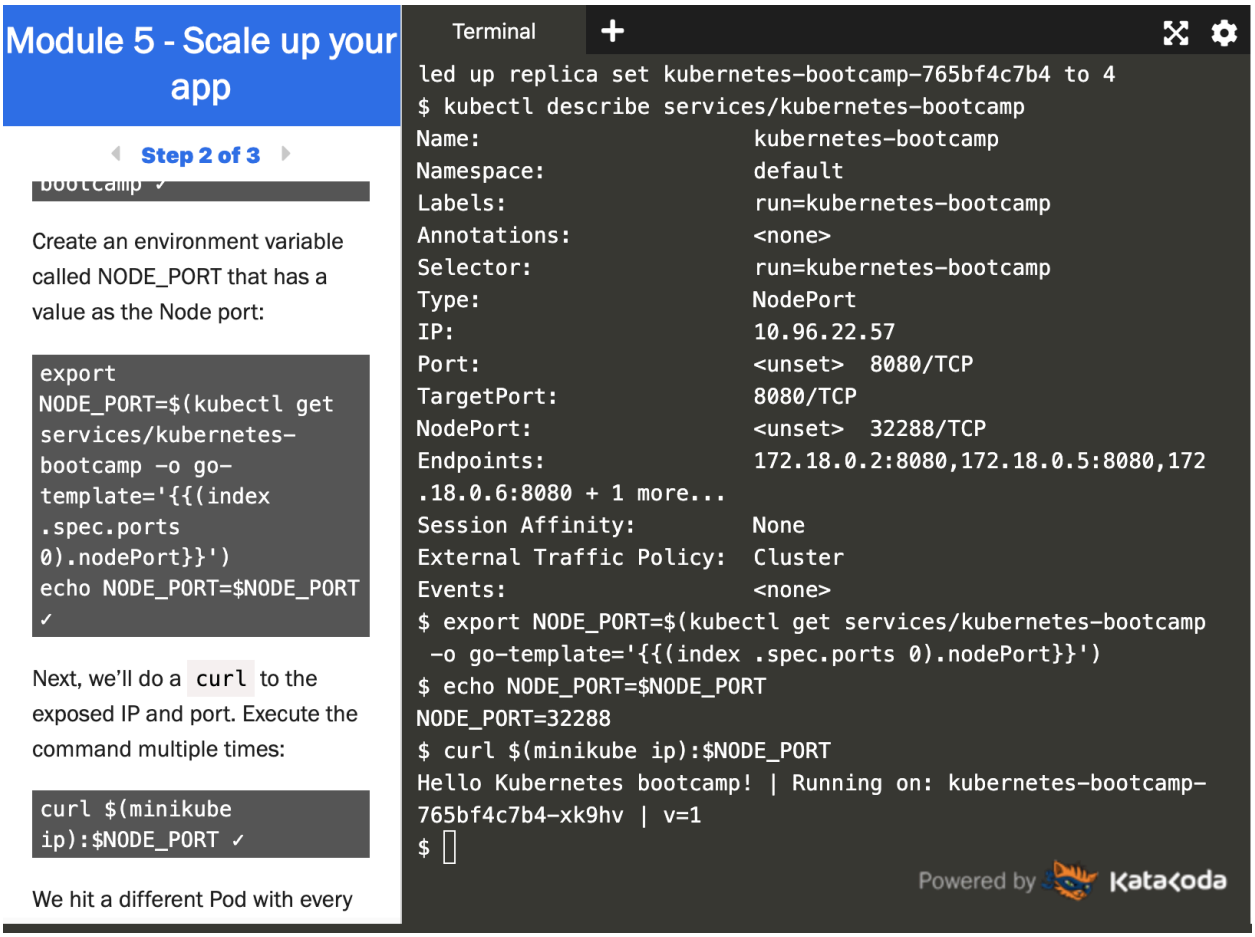
Now we can check the deployments using get deployments which shows that there are 4 instances of the application with different IP addresses. The describe command also shows that there are 4 replicas now.





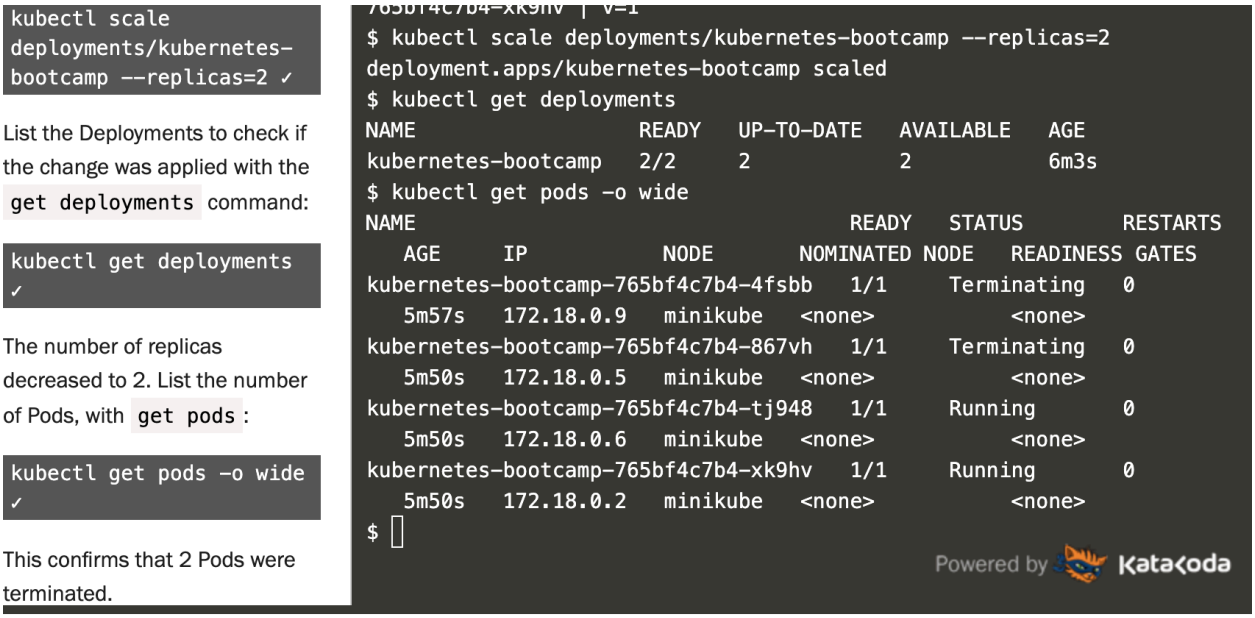
- **Load balancing**

Load balancing is the process of distributing the network traffic efficiently among the backend services so that there is maximum stability and availability. We can check if the service is load-balancing the traffic, we can use the describe services to find the IP and port. We can create an environment variable called NODE_PORT which has the value of the Node port using the command. Then we use curl to the exposed IP and port. The curl hits a different pods with every request which shows load balancing.



- **Scale down**

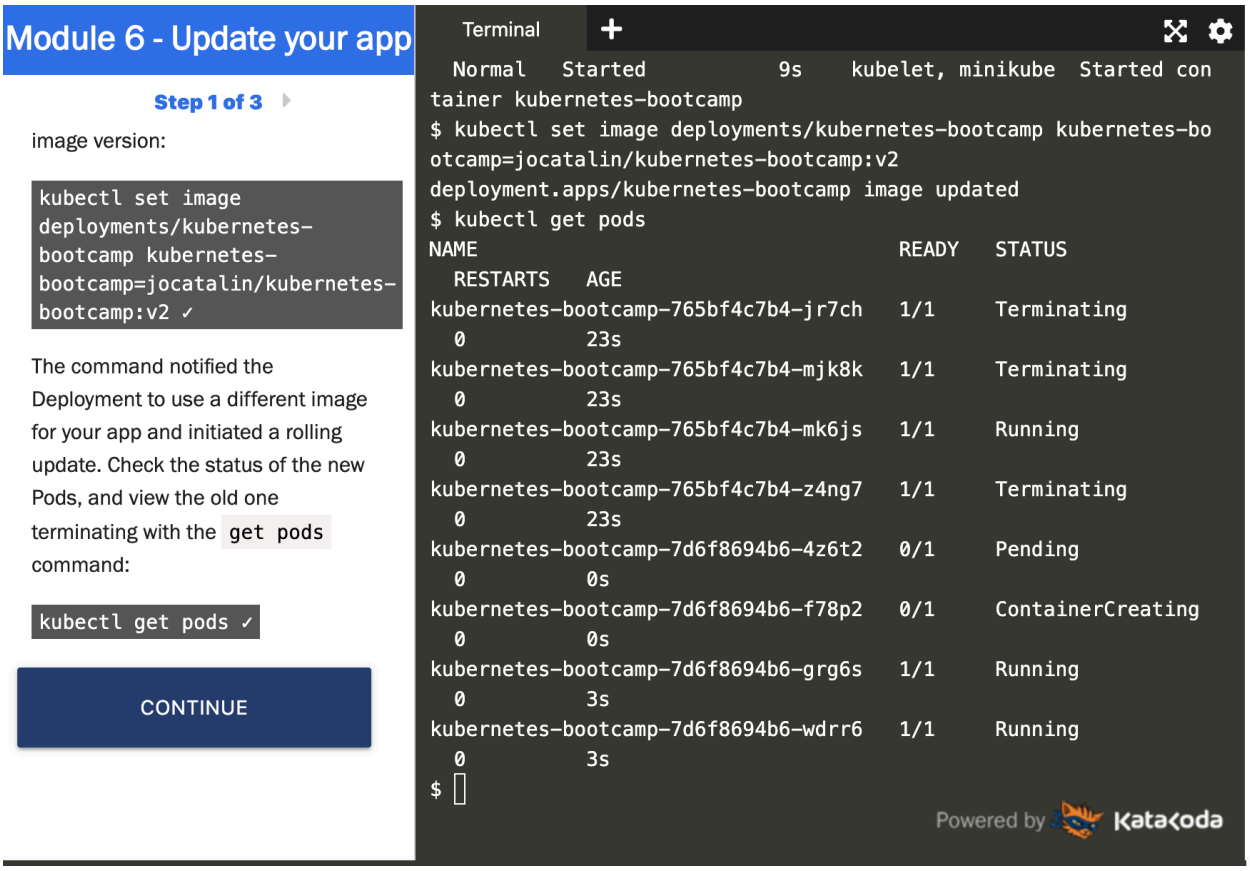
We can scale down the services to 2 replicas using the scale command. We can check the change is made using get deployments which shows that the number of replicas has changed to 2. We can see the number of pods which shows that 2 pods are terminated and 2 are running.

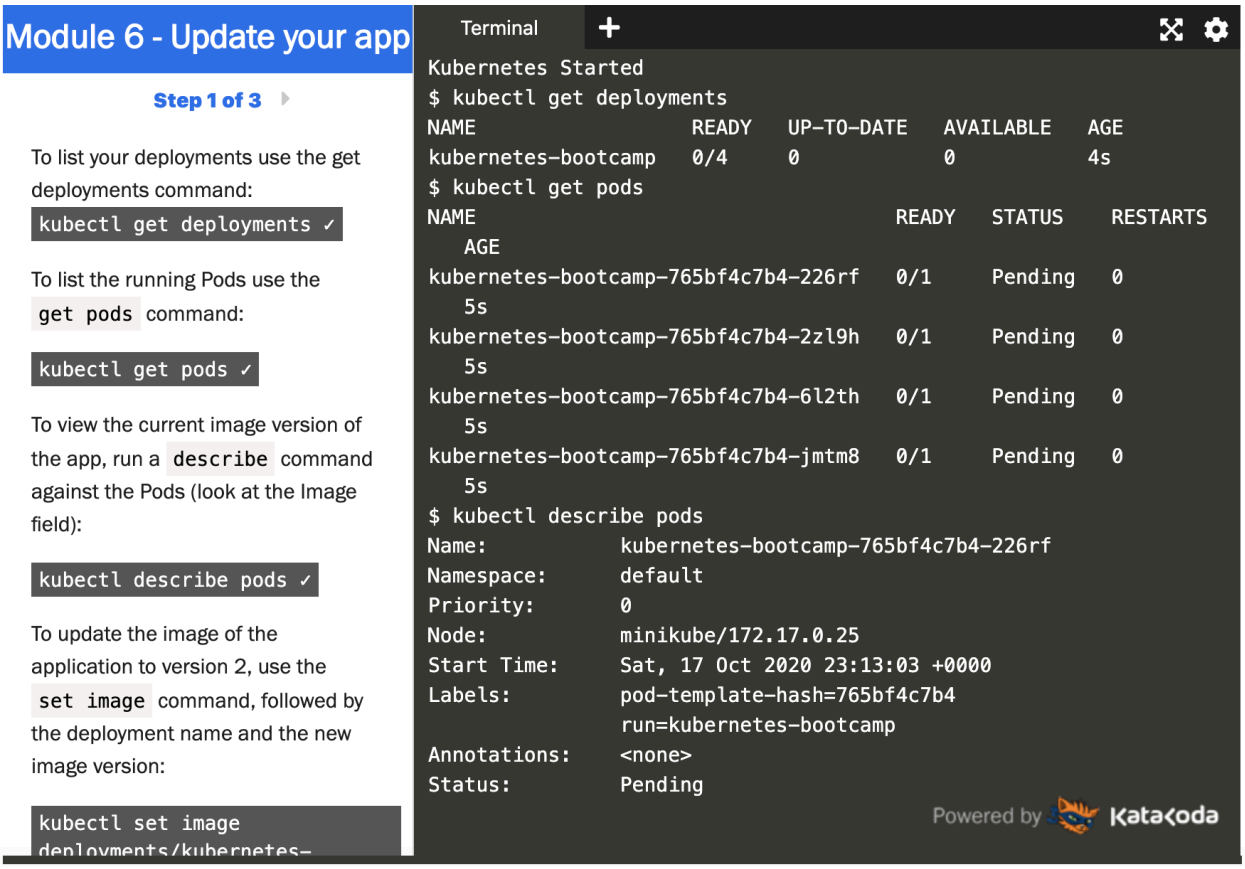


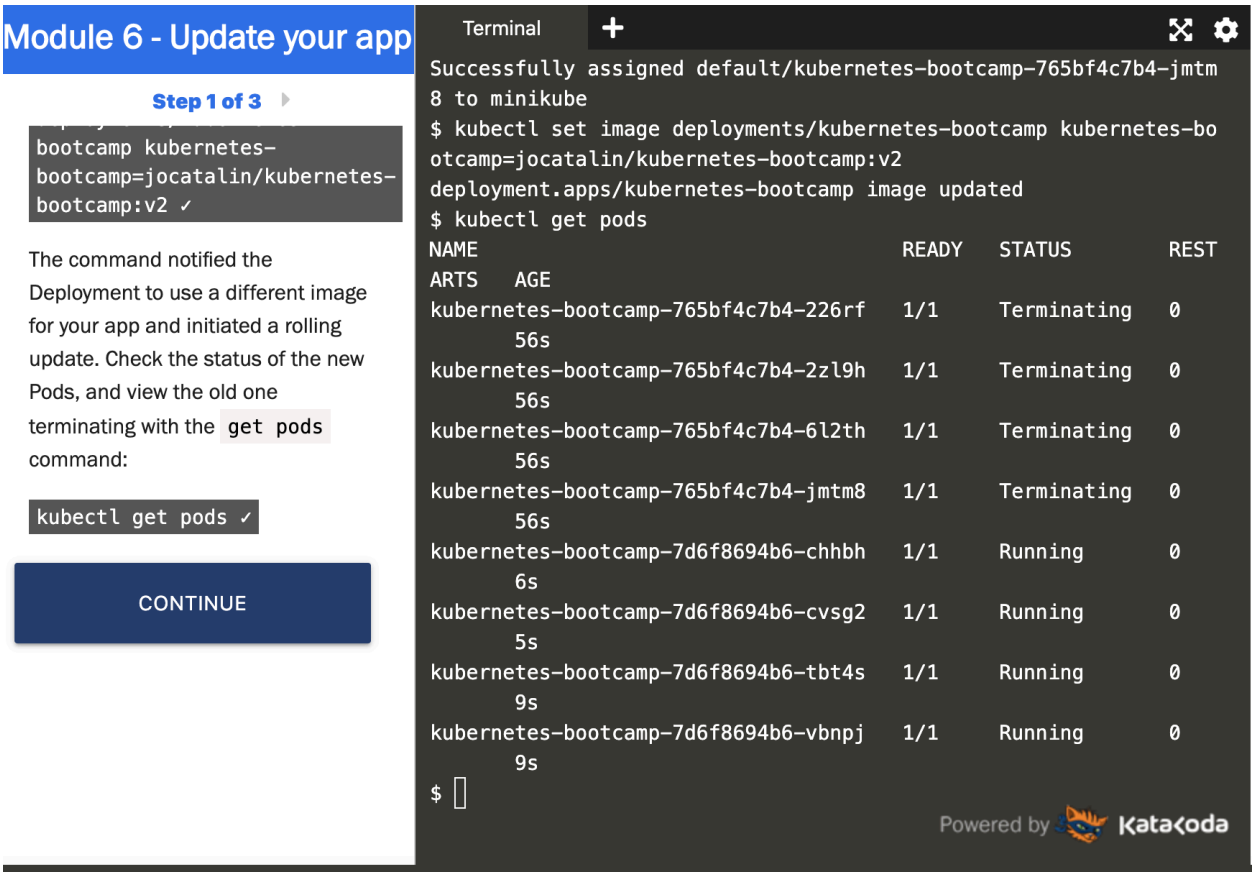
Module 6 – Update Your App

- **Update version of app**

We can again check the deployments and pods using the commands we used earlier. We need to update the image of the application to another version using set image command where e specify the deployment name and the new image version. This sets the version to 2. On checking the pods, we see that the old ones are terminating and the new ones are running.

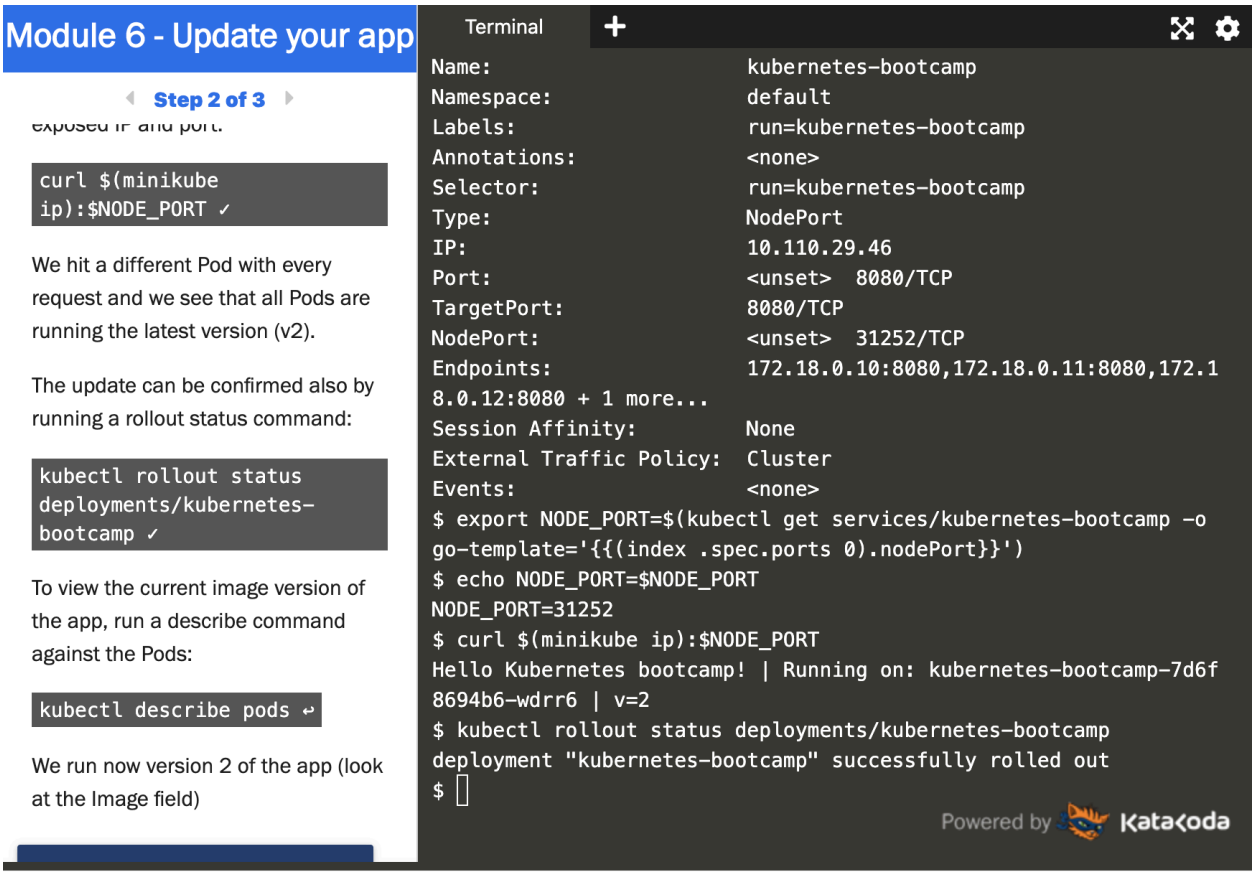

Module 6 - Update your app
Step 1 of 3
image version:
kubectl set image deployments/kubernetes-bootcamp kubernetes-bootcamp=jocatalin/kubernetes-bootcamp:v2 ✓
The command notified the Deployment to use a different image for your app and initiated a rolling update. Check the status of the new Pods, and view the old one terminating with the get pods command:
kubectl get pods ✓
CONTINUE
Terminal
Normal Started 9s kubelet, minikube Started container kubernetes-bootcamp
$ kubectl set image deployments/kubernetes-bootcamp kubernetes-bootcamp=jocatalin/kubernetes-bootcamp:v2
deployment.apps/kubernetes-bootcamp image updated
$ kubectl get pods
NAME READY STATUS RESTARTS AGE
kubernetes-bootcamp-765bf4c7b4-jr7ch 1/1 Terminating 0 23s
kubernetes-bootcamp-765bf4c7b4-mjk8k 1/1 Terminating 0 23s
kubernetes-bootcamp-765bf4c7b4-mk6js 1/1 Running 0 23s
kubernetes-bootcamp-765bf4c7b4-z4ng7 1/1 Terminating 0 23s
kubernetes-bootcamp-7d6f8694b6-4z6t2 0/1 Pending 0 0s
kubernetes-bootcamp-7d6f8694b6-f78p2 0/1 ContainerCreating 0 0s
kubernetes-bootcamp-7d6f8694b6-grg6s 1/1 Running 0 3s
kubernetes-bootcamp-7d6f8694b6-wdrr6 1/1 Running 0 3s
$
Powered by Katacoda


Module 6 - Update your app
Step 1 of 3
To list your deployments use the get deployments command:
kubectl get deployments ✓
To list the running Pods use the get pods command:
kubectl get pods ✓
To view the current image version of the app, run a describe command against the Pods (look at the Image field):
kubectl describe pods ✓
To update the image of the application to version 2, use the set image command, followed by the deployment name and the new image version:
kubectl set image deployments/kubernetes-
Terminal
Kubernetes Started
$ kubectl get deployments
NAME READY UP-TO-DATE AVAILABLE AGE
kubernetes-bootcamp 0/4 0 0 4s
$ kubectl get pods
NAME READY STATUS RESTARTS AGE
kubernetes-bootcamp-765bf4c7b4-226rf 0/1 Pending 0 5s
kubernetes-bootcamp-765bf4c7b4-2zl9h 0/1 Pending 0 5s
kubernetes-bootcamp-765bf4c7b4-6l2th 0/1 Pending 0 5s
kubernetes-bootcamp-765bf4c7b4-jmtm8 0/1 Pending 0 5s
$ kubectl describe pods
Name: kubernetes-bootcamp-765bf4c7b4-226rf
Namespace: default
Priority: 0
Node: minikube/172.17.0.25
Start Time: Sat, 17 Oct 2020 23:13:03 +0000
Labels: pod-template-hash=765bf4c7b4
run=kubernetes-bootcamp
Annotations: <none>
Status: Pending
Powered by Katacoda



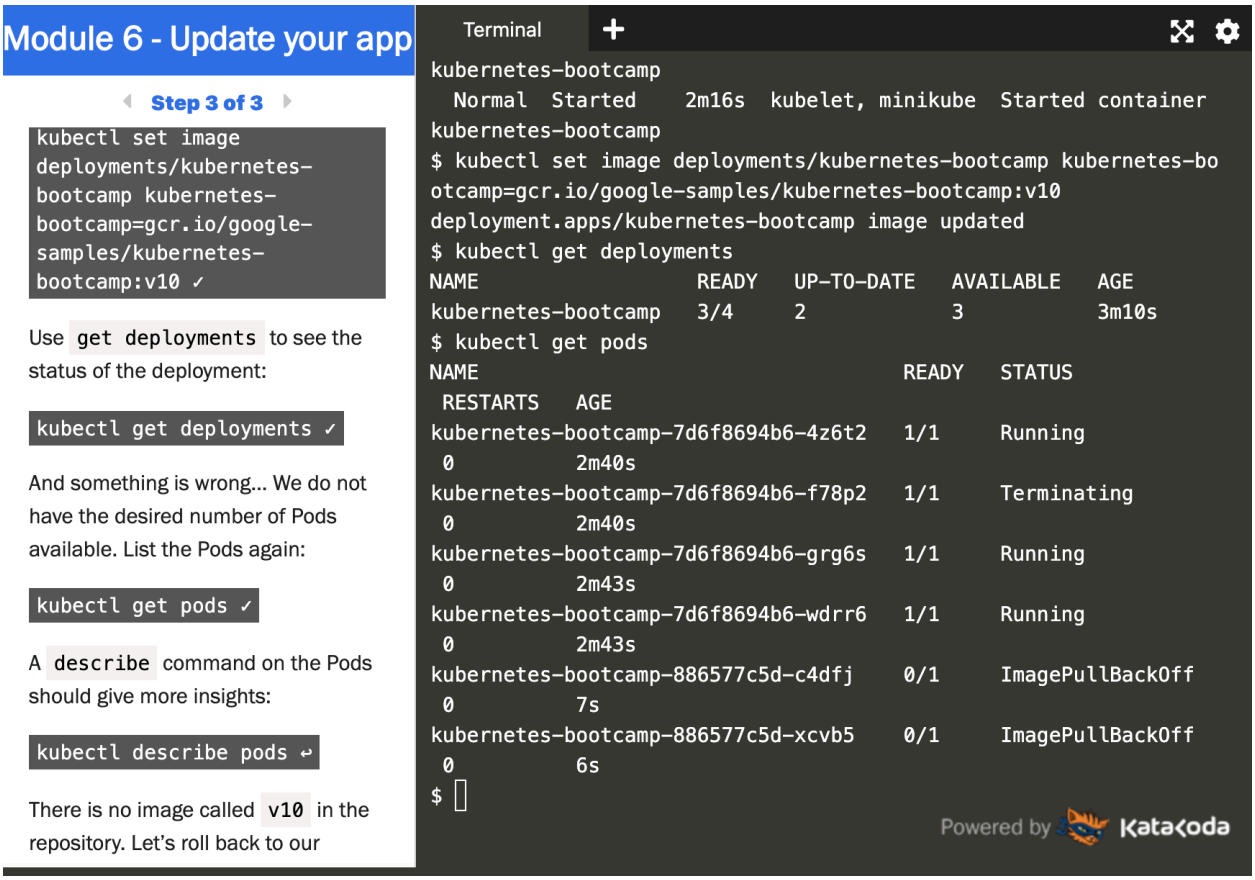
- **Verify an update**

We need to check that the app is running and find the exposed IP and port using the describe services. We again create an environment variable which has the port value and we do the curl to the exposed port and IP. This hits a different request every time and pods are running on v2. We can also check the status using the rollout command which shows it is successfully rolled out.

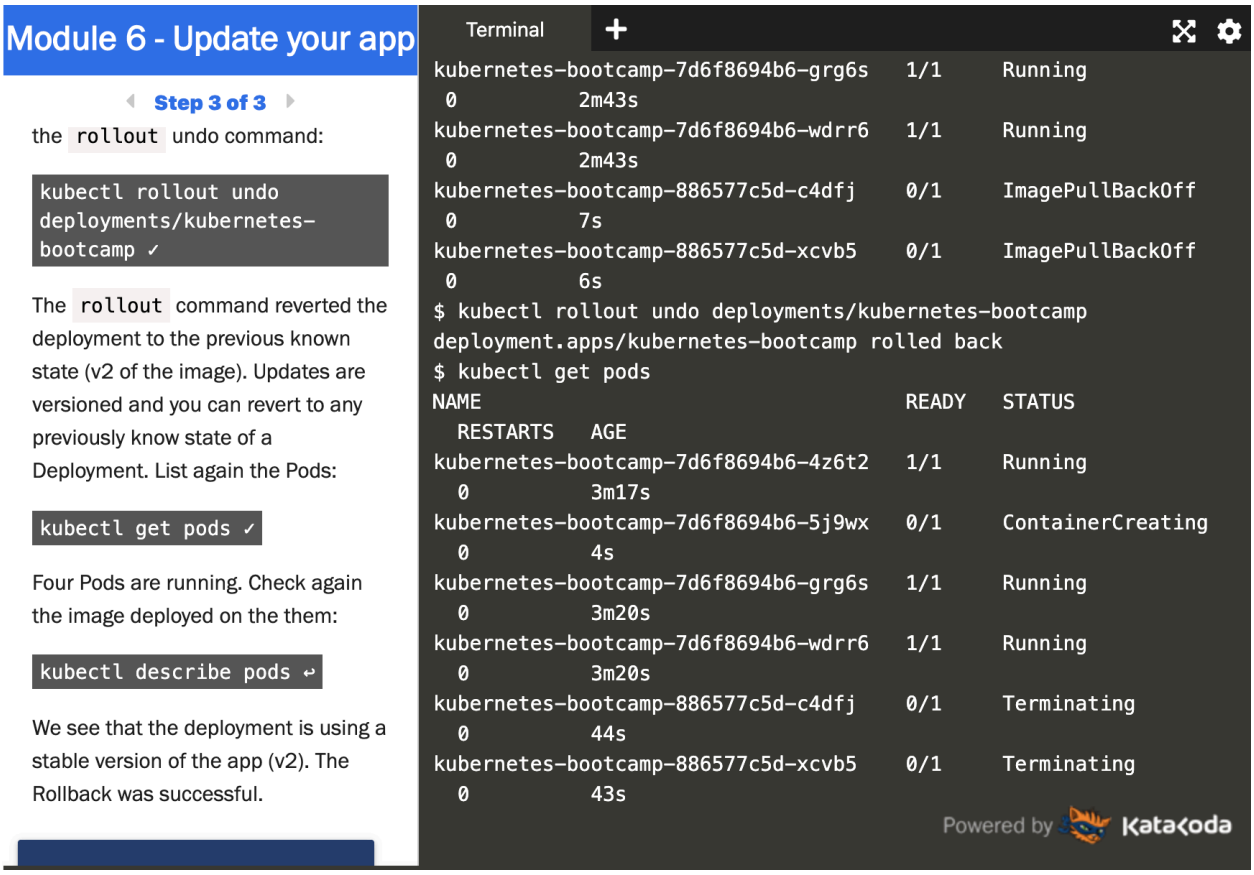


- **Rollback an update**

We now try to update it to a new version v10 and use get deployments to check the status of the deployment. We can see we do not have the desired number of pods as we did previously, and we check the pods details. This happens because the repository has not image called v10 and hence the rollout did not happen properly. We need to use the rollout undo command now.



The rollout command will revert the deployment to its previous state which was v2. Now the pods are properly running, and the deployment is using a stable version of the app, so our rollout was successful.


Module 6 - Update your app
Step 3 of 3
the rollout undo command:
kubectl rollout undo deployments/kubernetes-bootcamp ✓
The rollout command reverted the deployment to the previous known state (v2 of the image). Updates are versioned and you can revert to any previously know state of a Deployment. List again the Pods:
kubectl get pods ✓
Four Pods are running. Check again the image deployed on the them:
kubectl describe pods ↵
We see that the deployment is using a stable version of the app (v2). The Rollback was successful.
Terminal
kubernetes-bootcamp-7d6f8694b6-grg6s 1/1 Running 0 2m43s
kubernetes-bootcamp-7d6f8694b6-wdrr6 1/1 Running 0 2m43s
kubernetes-bootcamp-886577c5d-c4dfj 0/1 ImagePullBackOff 0 7s
kubernetes-bootcamp-886577c5d-xcvb5 0/1 ImagePullBackOff 0 6s
$ kubectl rollout undo deployments/kubernetes-bootcamp
deployment.apps/kubernetes-bootcamp rolled back
$ kubectl get pods
NAME READY STATUS RESTARTS AGE
kubernetes-bootcamp-7d6f8694b6-4z6t2 1/1 Running 0 3m17s
kubernetes-bootcamp-7d6f8694b6-5j9wx 0/1 ContainerCreating 0 4s
kubernetes-bootcamp-7d6f8694b6-grg6s 1/1 Running 0 3m20s
kubernetes-bootcamp-7d6f8694b6-wdrr6 1/1 Running 0 3m20s
kubernetes-bootcamp-886577c5d-c4dfj 0/1 Terminating 0 44s
kubernetes-bootcamp-886577c5d-xcvb5 0/1 Terminating 0 43s
Powered by Katacoda