

## 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, Kubernetes 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.

The screenshot shows a terminal window titled "Kubernetes Bootcamp Terminal". The window has a blue header bar with the title and a small "+" icon. Below the header is a dark gray terminal area. On the left side of the terminal, there is a sidebar with a blue header "Module 1 - Create a Kubernetes cluster" and a section titled "Step 2 of 3". This section contains text explaining how to interact with Kubernetes using the command line interface (kubectl) and provides a code block for running "kubectl version". The main terminal area displays the output of several commands:

```
$  
$ minikube version  
minikube version: v1.8.1  
commit: cbda04cf6bbe65e987ae52bb393c10099ab62014  
$ minikube start  
* minikube v1.8.1 on Ubuntu 18.04  
* Using the none driver based on user configuration  
* Running on localhost (CPUs=2, Memory=2460MB, Disk=145651  
MB) ...  
* OS release is Ubuntu 18.04.4 LTS  
* Preparing Kubernetes v1.17.3 on Docker 19.03.6 ...  
- kubelet.resolv-conf=/run/systemd/resolve/resolv.conf  
* Launching Kubernetes ...  
* Enabling addons: default-storageclass, storage-provision  
er  
* Configuring local host environment ...  
* Waiting for cluster to come online ...  
* Done! kubectl is now configured to use "minikube"  
$
```

At the bottom right of the terminal window, there is a watermark that says "Powered by Katacoda".

- **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.

## Module 1 - Create a Kubernetes cluster

◀ Step 2 of 3 ▶

To interact with Kubernetes during this bootcamp we'll use the command line interface, kubectl. We'll explain kubectl in detail in the next modules, but for now, we're just going to look at some cluster information. To check if kubectl is installed you can run the *kubectl version* command:

```
kubectl version ✓
```

OK, kubectl is configured and we can see both the version of the client and as well as the server. The client version is the kubectl version; the server version is the Kubernetes version installed on the master. You can also see details about the build.

Terminal +

```
* minikube v1.8.1 on Ubuntu 18.04
* Using the none driver based on user configuration
* Running on localhost (CPUs=2, Memory=2460MB, Disk=145651
MB) ...
* OS release is Ubuntu 18.04.4 LTS
* Preparing Kubernetes v1.17.3 on Docker 19.03.6 ...
  - kubelet.resolv-conf=/run/systemd/resolve/resolv.conf
* Launching Kubernetes ...
* Enabling addons: default-storageclass, storage-provisioner
* Configuring local host environment ...
* Waiting for cluster to come online ...
* Done! kubectl is now configured to use "minikube"
$ kubectl version
Client Version: version.Info{Major:"1", Minor:"17", GitVer
  sion:"v1.17.3", GitCommit:"06ad960bfd03b39c8310aaf92d1e7c1
  2ce618213", GitTreeState:"clean", BuildDate:"2020-02-11T18
  :14:22Z", GoVersion:"go1.13.6", Compiler:"gc", Platform:"l
  inux/amd64"}
Server Version: version.Info{Major:"1", Minor:"17", GitVer
  sion:"v1.17.3", GitCommit:"06ad960bfd03b39c8310aaf92d1e7c1
  2ce618213", GitTreeState:"clean", BuildDate:"2020-02-11T18
  :07:13Z", GoVersion:"go1.13.6", Compiler:"gc", Platform:"l
  inux/amd64"}
$ [ ]
```

Powered by  Katacoda

- 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 1 - Create a Kubernetes cluster

### ◀ Step 3 of 3 ▶

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

```
kubectl get nodes ✓
```

This command shows all nodes that can be used to host our applications. Now we have only one node, and we can see that its status is ready (it is ready to accept applications for deployment).

CONTINUE

```
Terminal + X ⚙
* Configuring local host environment ...
* Waiting for cluster to come online ...
* Done! kubectl is now configured to use "minikube"
$ kubectl version
Client Version: version.Info{Major:"1", Minor:"17", GitVersion:"v1.17.3", GitCommit:"06ad960bfd03b39c8310aaf92d1e7c12ce618213", GitTreeState:"clean", BuildDate:"2020-02-11T18:14:22Z", GoVersion:"go1.13.6", Compiler:"gc", Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"17", GitVersion:"v1.17.3", GitCommit:"06ad960bfd03b39c8310aaf92d1e7c12ce618213", GitTreeState:"clean", BuildDate:"2020-02-11T18:07:13Z", GoVersion:"go1.13.6", Compiler:"gc", Platform:"linux/amd64"}
$ kubectl cluster-info
Kubernetes master is running at https://172.17.0.35:8443
KubeDNS is running at https://172.17.0.35:8443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
$ kubectl get nodes
NAME      STATUS   ROLES     AGE    VERSION
minikube  Ready    master    93s   v1.17.3
$ [REDACTED]
```

Powered by  Katacoda

## 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.

The screenshot shows a terminal window titled "Kubernetes Bootcamp Terminal". The terminal displays the following output:

```
$ sleep 1; launch.sh
Starting Kubernetes.....expected to take less than a minute
Kubernetes Started
$ kubectl version
Client Version: version.Info{Major:"1", Minor:"17", GitVersion:"v1.17.3", GitCommit:"06ad960bfd03b39c8310aaf92d1e7c12ce618213", GitTreeState:"clean", BuildDate:"2020-02-11T18:14:22Z", GoVersion:"go1.13.6", Compiler:"gc", Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"17", GitVersion:"v1.17.3", GitCommit:"06ad960bfd03b39c8310aaf92d1e7c12ce618213", GitTreeState:"clean", BuildDate:"2020-02-11T18:07:13Z", GoVersion:"go1.13.6", Compiler:"gc", Platform:"linux/amd64"}
$ kubectl get nodes
NAME STATUS ROLES AGE VERSION
minikube Ready master 28s v1.17.3
$ []
```

On the left side of the terminal window, there is a sidebar with the following content:

- Step 1 of 3**
- You can use `--help` after the command to get additional info about possible parameters (`kubectl get nodes --help`).
- Check that kubectl is configured to talk to your cluster, by running the `kubectl version` command:  
`kubectl version ✓`
- OK, kubectl is installed and you can see both the client and the server versions.
- To view the nodes in the cluster, run the `kubectl get nodes` command:  
`kubectl get nodes ✓`
- Here we see the available nodes (1 in our case). Kubernetes will choose where to deploy our application based on Node available resources.

At the bottom right of the terminal window, it says "Powered by Katacoda".

- **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.

## Module 2 - Deploy an app

◀ Step 2 of 3 ▶

Performed a few steps for you.

- searched for a suitable node where an instance of the application could be run (we have only 1 available node)
- scheduled the application to run on that Node
- configured the cluster to reschedule the instance on a new Node when needed

To list your deployments use the `get deployments` command:

```
kubectl get deployments ✓
```

We see that there is 1 deployment running a single instance of your app. The instance is running inside a Docker container on your node.

[CONTINUE](#)

Terminal + Kubernetes Bootcamp Terminal

```
$ sleep 1; launch.sh
Starting Kubernetes.....expected to take less than a minute
Kubernetes Started
$ kubectl version
Client Version: version.Info{Major:"1", Minor:"17", GitVersion:"v1.17.3", GitCommit:"06ad960bfd03b39c8310aaf92d1e7c12ce618213", GitTreeState:"clean", BuildDate:"2020-02-11T18:14:22Z", GoVersion:"go1.13.6", Compiler:"gc", Platform:"linux/amd64"}
Server Version: version.Info{Major:"1", Minor:"17", GitVersion:"v1.17.3", GitCommit:"06ad960bfd03b39c8310aaf92d1e7c12ce618213", GitTreeState:"clean", BuildDate:"2020-02-11T18:07:13Z", GoVersion:"go1.13.6", Compiler:"gc", Platform:"linux/amd64"}
$ kubectl get nodes
NAME      STATUS    ROLES     AGE   VERSION
minikube  Ready     master    28s   v1.17.3
$ kubectl create deployment kubernetes-bootcamp --image=gcr.io/google-samples/kubernetes-bootcamp:v1
deployment.apps/kubernetes-bootcamp created
$ kubectl get deployments
NAME                  READY   UP-TO-DATE   AVAILABLE   AGE
kubernetes-bootcamp  1/1     1           1           11s
$
```

Powered by  Katacoda

- **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.

## Module 2 - Deploy an app

### ◀ Step 2 of 3 ▶

- have only 1 available node)
- scheduled the application to run on that Node
  - configured the cluster to reschedule the instance on a new Node when needed

To list your deployments use the `get deployments` command:

`kubectl get deployments ✓`

We see that there is 1 deployment running a single instance of your app. The instance is running inside a Docker container on your node.

CONTINUE

```
Terminal +  
$ sleep 1; launch.sh  
Starting Kubernetes. This is expected to take less than a  
minute.....  
Kubernetes Started  
$ kubectl version  
Client Version: version.Info{Major:"1", Minor:"17", GitVer  
sion:"v1.17.3", GitCommit:"06ad960bfd03b39c8310aaf92d1e7c1  
2ce618213", GitTreeState:"clean", BuildDate:"2020-02-11T18  
:14:22Z", GoVersion:"go1.13.6", Compiler:"gc", Platform:"l  
inux/amd64"}  
Server Version: version.Info{Major:"1", Minor:"17", GitVer  
sion:"v1.17.3", GitCommit:"06ad960bfd03b39c8310aaf92d1e7c1  
2ce618213", GitTreeState:"clean", BuildDate:"2020-02-11T18  
:07:13Z", GoVersion:"go1.13.6", Compiler:"gc", Platform:"l  
inux/amd64"}  
$ kubectl get nodes  
NAME STATUS ROLES AGE VERSION  
minikube NotReady master 14s v1.17.3  
$ kubectl create deployment kubernetes-bootcamp --image=gc  
r.io/google-samples/kubernetes-bootcamp:v1  
deployment.apps/kubernetes-bootcamp created  
$ kubectl get deployments  
NAME READY UP-TO-DATE AVAILABLE AGE  
kubernetes-bootcamp 1/1 1 1 12s  
$ [ ]
```

Powered by  Katacoda

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.

## Module 2 - Deploy an app

◀ Step 3 of 3 ▶

If Port 8001 is not accessible, ensure that the `kubectl proxy` started above is running.

The API server will automatically create an endpoint for each pod, based on the pod name, that is also accessible through the proxy.

First we need to get the Pod name, and we'll store in the environment variable `POD_NAME`:

```
export POD_NAME=$(kubectl get pods -o go-template --template '{{range .items}}{{.metadata.name}}\n{{end}}')  
echo Name of the Pod:  
$POD_NAME ↵
```

*Note: Check the top of the terminal.  
The proxy was run in a new tab*

Terminal Terminal 2 +

```
ersion:"v1.17.3", GitCommit:"06ad960bfd03b39c8310aaf92d1e7c12ce618213", GitTreeState:"clean", BuildDate:"2020-02-11T18:07:13Z", GoVersion:"go1.13.6", Compiler:"gc", Platform:"linux/amd64"}  
$ kubectl get nodes  
NAME STATUS ROLES AGE VERSION  
minikube NotReady master 14s v1.17.3  
$ kubectl create deployment kubernetes-bootcamp --image=gcr.io/google-samples/kubernetes-bootcamp:v1  
deployment.apps/kubernetes-bootcamp created  
$ kubectl get deployments  
NAME READY UP-TO-DATE AVAILABLE AGE  
kubernetes-bootcamp 1/1 1 1 12s  
$ curl http://localhost:8001/version  
{  
  "major": "1",  
  "minor": "17",  
  "gitVersion": "v1.17.3",  
  "gitCommit": "06ad960bfd03b39c8310aaf92d1e7c12ce618213",  
  "gitTreeState": "clean",  
  "buildDate": "2020-02-11T18:07:13Z",  
  "goVersion": "go1.13.6",  
  "compiler": "gc",  
  "platform": "linux/amd64"  
} $
```

Powered by  Katacoda

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 2 - Deploy an app

### ◀ Step 3 of 3 ▶

*Note: Check the top or the terminal.*

*The proxy was run in a new tab (Terminal 2), and the recent commands were executed the original tab (Terminal 1). The proxy still runs in the second tab, and this allowed our curl command to work using localhost:8001 .*

In order for the new deployment to be accessible without using the Proxy, a Service is required which will be explained in the next modules.

CONTINUE

```
Terminal Terminal 2 +  
$ kubectl get nodes  
NAME STATUS ROLES AGE VERSION  
minikube NotReady master 14s v1.17.3  
$ kubectl create deployment kubernetes-bootcamp --image=gcr.io/google-samples/kubernetes-bootcamp:v1  
deployment.apps/kubernetes-bootcamp created  
$ kubectl get deployments  
NAME READY UP-TO-DATE AVAILABLE AGE  
kubernetes-bootcamp 1/1 1 1 12s  
$ curl http://localhost:8001/version  
{  
  "major": "1",  
  "minor": "17",  
  "gitVersion": "v1.17.3",  
  "gitCommit": "06ad960bfd03b39c8310aaf92d1e7c12ce618213",  
  "gitTreeState": "clean",  
  "buildDate": "2020-02-11T18:07:13Z",  
  "goVersion": "go1.13.6",  
  "compiler": "gc",  
  "platform": "linux/amd64"  
}$ 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-69fbc6f4cf-jmw2l  
$
```

Powered by  Katacoda

## 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.

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

```
kubectl get pods ✓
```

If no pods are running then it means the interactive environment is still reloading it's previous state. Please wait a couple of seconds and list the Pods again. You can continue once you see the one Pod running.

[Next to view what containers are](#)

```
$ kubectl get pods
NAME                               READY   STATUS    REST
ARTS     AGE
kubernetes-bootcamp-765bf4c7b4-ksfgr   1/1     Running   0
11s
$ kubectl describe pods
Name:           kubernetes-bootcamp-765bf4c7b4-ksfgr
Namespace:      default
Priority:       0
Node:          minikube/172.17.0.28
Start Time:    Sat, 17 Oct 2020 22:46:22 +0000
Labels:         pod-template-hash=765bf4c7b4
                run=kubernetes-bootcamp
Annotations:   <none>
Status:        Running
IP:            172.18.0.4
```

Powered by  Katacoda

- 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.

To see the output of our application, run a `curl` request.

```
curl
http://localhost:8001/api/v1/✓
```

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

**CONTINUE**

```
Normal    Started   2m20s   Kubetcl,
minikube  Started container kubernetes-bootcamp
$ 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-ksfgr
$ curl http://localhost:8001/api/v1/namespaces/default/pods/$POD_NAME/proxy/
Hello Kubernetes bootcamp! | Running on: kubernetes-bootcamp-765bf4c7b4-ksfgr | v=1
$ [ ]
```

Powered by  Katacoda

- 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.

```
kubectl logs $POD_NAME ✓
```

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

CONTINUE

```
Hello Kubernetes bootcamp! | Running On: kubernetes-bootcamp-765bf4c7b4-ksfgr | v=1
$ kubectl logs $POD_NAME
Kubernetes Bootcamp App Started At: 2020-10-17T22:46:24.735Z
| Running On: kubernetes-bootcamp-765bf4c7b4-ksfgr

Running On: kubernetes-bootcamp-765bf4c7b4-ksfgr | Total Requests: 1 | App Uptime: 167.386 seconds | Log Time: 2020-10-17T22:49:12.121Z
$ [ ]
```

Powered by  Katacoda

- 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.

## Module 3 - Explore your app

◀ Step 4 of 4 ▶

variables.

```
kubectl exec $POD_NAME env
✓
```

Again, worth mentioning that the name of the container itself can be omitted since we only have a single container in the Pod.

Next let's start a bash session in the Pod's container:

```
kubectl exec -ti $POD_NAME
bash ↵
```

We have now an open console on the container where we run our NodeJS application. The source code of the app is in the server.js file:

```
cat server.js ↵
```

```
Terminal Terminal 2 + X ⚙
Hello Kubernetes bootcamp! | Running on: kubernetes-bootcamp-765bf4c7b4-ksfgr | v=1
$ kubectl logs $POD_NAME
Kubernetes Bootcamp App Started At: 2020-10-17T22:46:24.735Z
| Running On: kubernetes-bootcamp-765bf4c7b4-ksfgr

Running On: kubernetes-bootcamp-765bf4c7b4-ksfgr | Total Requests: 1 | App Uptime: 167.386 seconds | Log Time: 2020-10-17T22:49:12.121Z
$ kubectl exec $POD_NAME env
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOSTNAME=kubernetes-bootcamp-765bf4c7b4-ksfgr
KUBERNETES_PORT_443_TCP_ADDR=10.96.0.1
KUBERNETES_SERVICE_HOST=10.96.0.1
KUBERNETES_SERVICE_PORT=443
KUBERNETES_SERVICE_PORT_HTTPS=443
KUBERNETES_PORT=tcp://10.96.0.1:443
KUBERNETES_PORT_443_TCP=tcp://10.96.0.1:443
KUBERNETES_PORT_443_TCP_PROTO=tcp
KUBERNETES_PORT_443_TCP_PORT=443
NPM_CONFIG_LOGLEVEL=info
NODE_VERSION=6.3.1
HOME=/root
$ [ ]
```

Powered by  Katacoda

## Module 3 - Explore your app

### Step 4 of 4

omitted since we only have a single container in the Pod.

Next let's start a bash session in the Pod's container:

```
kubectl exec -ti $POD_NAME  
bash ✓
```

We have now an open console on the container where we run our NodeJS application. The source code of the app is in the server.js file:

```
cat server.js ✓
```

You can check that the application is up by running a curl command:

```
curl localhost:8080 ↵
```

*Note: here we used localhost*

```
Terminal Terminal 2 +  
$ kubectl exec -ti $POD_NAME bash  
root@kubernetes-bootcamp-765bf4c7b4-ksfgr:/# cat server.js  
var http = require('http');  
var requests=0;  
var podname= process.env.HOSTNAME;  
var startTime;  
var host;  
var handleRequest = function(request, response) {  
    response.setHeader('Content-Type', 'text/plain');  
    response.writeHead(200);  
    response.write("Hello Kubernetes bootcamp! | Running on: ");  
    response.write(host);  
    response.end(" | v=1\n");  
    console.log("Running On:", host, "| Total Requests:", ++requests, "| App Uptime:", (new Date() - startTime)/1000, "seconds", "| Log Time:", new Date());  
}  
var www = http.createServer(handleRequest);  
www.listen(8080,function () {  
    startTime = new Date();  
    host = process.env.HOSTNAME;  
    console.log ("Kubernetes Bootcamp App Started At:",startTime, "| Running On: ",host, "\n" );  
});
```

Powered by  Katacoda

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

To close your container connection type `exit ✓`.

CONTINUE

```
});  
root@kubernetes-bootcamp-765bf4c7b4-ksfgr:/# curl localhost:8080  
Hello Kubernetes bootcamp! | Running on: kubernetes-bootcamp-765bf4c7b4-ksfgr | v=1  
root@kubernetes-bootcamp-765bf4c7b4-ksfgr:/# exit  
exit  
$ []
```

Powered by  Katacoda

## 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 expose 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.

The screenshot shows a terminal window with the following content:

```
Module 4 - Expose your app publicly
Step 1 of 3
--port 8080 ✓

Let's run again the get services command:
kubectl get services ✓

We have now a running Service called kubernetes-bootcamp. Here we see that the Service received a unique cluster-IP, an internal port and an external-IP (the IP of the Node).

To find out what port was opened externally (by the NodePort option) we'll run the describe service command:
kubectl describe services/kubernetes-bootcamp ↵

Terminal + x 🔍
ute.....
Kubernetes Started
$ kubectl get pods
No resources found in default namespace.
$ kubectl get pods
NAME                                     READY   STATUS    REST
ARTS   AGE
kubernetes-bootcamp-765bf4c7b4-hkbvx   0/1     Pending   0
      0s
$ kubectl get services
NAME          TYPE        CLUSTER-IP      EXTERNAL-IP      PORT(S)
AGE
kubernetes   ClusterIP   10.96.0.1      <none>           443/TCP
      15s
$ kubectl expose deployment/kubernetes-bootcamp --type="NodePort" --port 8080
service/kubernetes-bootcamp exposed
$ kubectl get services
NAME          TYPE        CLUSTER-IP      EXTERNAL-IP      PORT(S)
PORT(S)        AGE
kubernetes   ClusterIP   10.96.0.1      <none>
      443/TCP      33s
kubernetes-bootcamp   NodePort      10.110.120.87   <none>
      8080:32524/TCP  3s
$ [ ]
```

Powered by Katacoda

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.

## Module 4 - Expose your app publicly

### Step 1 of 3 ▶

```
template= '{{(index
.spec.ports
0).nodePort}}')
echo NODE_PORT=$NODE_PORT
✓
```

Now we can test that the app is exposed outside of the cluster using `curl`, the IP of the Node and the externally exposed port:

```
curl $(minikube
ip):$NODE_PORT ✓
```

And we get a response from the server. The Service is exposed.

CONTINUE

```
Terminal + kubernetes-bootcamp  NodePort  10.110.120.87  <none>
          8080:32524/TCP  3s
$ kubectl describe services/kubernetes-bootcamp
Name:           kubernetes-bootcamp
Namespace:      default
Labels:         run=kubernetes-bootcamp
Annotations:   <none>
Selector:       run=kubernetes-bootcamp
Type:          NodePort
IP:            10.110.120.87
Port:          <unset>  8080/TCP
TargetPort:    8080/TCP
NodePort:      <unset>  32524/TCP
Endpoints:    172.18.0.3:8080
Session Affinity: None
External Traffic Policy: Cluster
Events:        <none>
$ export NODE_PORT=$(kubectl get services/kubernetes-bootcamp
-o go-template='{{(index .spec.ports 0).nodePort}}')
$ echo NODE_PORT=$NODE_PORT
NODE_PORT=32524
$ curl $(minikube ip):$NODE_PORT
Hello Kubernetes bootcamp! | Running on: kubernetes-bootcamp-
765bf4c7b4-hkbvx | v=1
$ [
```

Powered by  Katacoda

- **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 available | 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:v1
  1
    Port:        8080/TCP
    Host Port:   0/TCP
    Environment: <none>
    Mounts:      <none>
    Volumes:     <none>
  Conditions:
    Type        Status  Reason
    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-hkbvx   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-hkbvx
$ kubectl label pod $POD_NAME app=v1
pod/kubernetes-bootcamp-765bf4c7b4-hkbvx labeled
$ [
```

Powered by Katacoda

```
$ kubectl get pods -l app=v1
NAME          READY   STATUS    RESTA
RTS   AGE
kubernetes-bootcamp-765bf4c7b4-hkbvx  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-hkbvx  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-hkbvx | 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.

## Module 5 - Scale up your app

### Step 1 of 3 ▶

get deployments command:

```
kubectl get deployments ✓
```

The output should be similar to:

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
kubernetes-bootcamp	1/1	1/1	1	39s
1	1	11m		

We should have 1 Pod. If not, run the command again. This shows:

- *NAME* lists the names of the Deployments in the cluster.
- *READY* shows the ratio of

Terminal + Kubernetes Bootcamp Terminal

```
sleep 1; launch.sh
$ 
$ sleep 1; launch.sh
Starting Kubernetes. This is expected to take less than a minute..
.....
Kubernetes Started
$ kubectl get deployments
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
kubernetes-bootcamp   1/1     1           1          39s
$ kubectl get rs
NAME          DESIRED   CURRENT   READY   AGE
kubernetes-bootcamp-765bf4c7b4   1         1         1       67s
$ kubectl scale deployments/kubernetes-bootcamp --replicas=4
deployment.apps/kubernetes-bootcamp scaled
$ kubectl get deployments
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
kubernetes-bootcamp   1/4     4           1          89s
```

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.

- Deployments in the cluster.
- *READY* shows the ratio of CURRENT/DESIRED replicas
  - *UP-TO-DATE* displays the number of replicas that have been updated to achieve the desired state.
  - *AVAILABLE* displays how many replicas of the application are

```
$ kubectl describe deployments/kubernetes-bootcamp
Name:                 kubernetes-bootcamp
Namespace:            default
CreationTimestamp:    Mon, 19 Oct 2020 14:23:47 +0000
Labels:               run=kubernetes-bootcamp
Annotations:          deployment.kubernetes.io/revision: 1
Selector:             run=kubernetes-bootcamp
Replicas:             4 desired | 4 updated | 4 total | 4 available
Powered by Katacoda
```

## Module 5 - Scale up your app

Step 1 of 3 ▶

```
kubectl get pods -o wide ✓
```

There are 4 Pods now, with different IP addresses. The change was registered in the Deployment events log. To check that, use the describe command:

```
kubectl describe deployments/kubernetes-bootcamp ✓
```

You can also view in the output of this command that there are 4 replicas now.

CONTINUE

Terminal +

```
$ kubectl describe deployments/kubernetes-bootcamp
Name:           kubernetes-bootcamp
Namespace:      default
CreationTimestamp: Sat, 17 Oct 2020 23:05:26 +0000
Labels:          run=kubernetes-bootcamp
Annotations:    deployment.kubernetes.io/revision: 1
Selector:        run=kubernetes-bootcamp
Replicas:       4 desired | 4 updated | 4 total | 4 available | 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:v1
      Port:       8080/TCP
      Host Port:  0/TCP
      Environment: <none>
      Mounts:     <none>
      Volumes:    <none>
  Conditions:
    Type     Status  Reason
1
```

Powered by  Katacoda

- **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.

## Module 5 - Scale up your app

◀ Step 2 of 3 ▶

bootcamp >

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

```
export
NODE_PORT=$(kubectl get
services/kubernetes-
bootcamp -o go-
template='{{(index
.spec.ports
0).nodePort}}')
echo NODE_PORT=$NODE_PORT
✓
```

Next, we'll do a curl to the exposed IP and port. Execute the command multiple times:

```
curl $(minikube
ip):$NODE_PORT
✓
```

We hit a different Pod with every

Terminal

+

✖ ⚙

```
led up replica set kubernetes-bootcamp-765bf4c7b4 to 4
$ kubectl describe services/kubernetes-bootcamp
Name:           kubernetes-bootcamp
Namespace:      default
Labels:         run=kubernetes-bootcamp
Annotations:   <none>
Selector:       run=kubernetes-bootcamp
Type:          NodePort
IP:            10.96.22.57
Port:          <unset>  8080/TCP
TargetPort:    8080/TCP
NodePort:      <unset>  32288/TCP
Endpoints:    172.18.0.2:8080,172.18.0.5:8080,172
               .18.0.6:8080 + 1 more...
Session Affinity: None
External Traffic Policy: Cluster
Events:        <none>
$ export NODE_PORT=$(kubectl get services/kubernetes-bootcamp
-o go-template='{{(index .spec.ports 0).nodePort}}')
$ echo NODE_PORT=$NODE_PORT
NODE_PORT=32288
$ curl $(minikube ip):$NODE_PORT
Hello Kubernetes bootcamp! | Running on: kubernetes-bootcamp-
765bf4c7b4-xk9hv | v=1
$
```

Powered by  Katacoda

- **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.

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

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

```
kubectl get deployments  
✓
```

The number of replicas decreased to 2. List the number of Pods, with `get pods`:

```
kubectl get pods -o wide  
✓
```

This confirms that 2 Pods were terminated.

```
/05014c7b4-xk9hv | v=1  
$ kubectl scale deployments/kubernetes-bootcamp --replicas=2  
deployment.apps/kubernetes-bootcamp scaled  
$ kubectl get deployments  
NAME READY UP-TO-DATE AVAILABLE AGE  
kubernetes-bootcamp 2/2 2 2 6m3s  
$ kubectl get pods -o wide  
NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES  
kubernetes-bootcamp-765bf4c7b4-4fsbb 1/1 Terminating 5m57s 172.18.0.9 minikube <none> <none>  
kubernetes-bootcamp-765bf4c7b4-867vh 1/1 Terminating 5m50s 172.18.0.5 minikube <none> <none>  
kubernetes-bootcamp-765bf4c7b4-tj948 1/1 Running 5m50s 172.18.0.6 minikube <none> <none>  
kubernetes-bootcamp-765bf4c7b4-xk9hv 1/1 Running 5m50s 172.18.0.2 minikube <none> <none>  
$ [ ]
```

Powered by  Katacoda

## 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 we 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-bootcamp=kube-node:v2
```

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

**Module 6 - Update your app**

**Step 1 of 3**

```
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 +

```
Successfully assigned default/kubernetes-bootcamp-765bf4c7b4-jmtm8 to minikube
$ kubectl set image deployments/kubernetes-bootcamp kubernetes-bootcamp=jocatalin/kubernetes-bootcamp:v2
deployment.apps/kubernetes-bootcamp image updated
$ kubectl get pods
NAME                                READY   STATUS    AGE
ARTS      AGE
kubernetes-bootcamp-765bf4c7b4-226rf  1/1    Terminating   56s
kubernetes-bootcamp-765bf4c7b4-2zl9h  1/1    Terminating   56s
kubernetes-bootcamp-765bf4c7b4-6l2th  1/1    Terminating   56s
kubernetes-bootcamp-765bf4c7b4-jmtm8  1/1    Terminating   56s
kubernetes-bootcamp-7d6f8694b6-chhbh  1/1    Running     6s
kubernetes-bootcamp-7d6f8694b6-cvsg2  1/1    Running     5s
kubernetes-bootcamp-7d6f8694b6-tbt4s  1/1    Running     9s
kubernetes-bootcamp-7d6f8694b6-vbnpj  1/1    Running     9s
$ []
```

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.

Module 6 - Update your app

◀ Step 2 of 3 ▶

Exposed IP and port

```
curl $(minikube ip):$NODE_PORT ✓
```

We hit a different Pod with every request and we see that all Pods are running the latest version (v2).

The update can be confirmed also by running a rollout status command:

```
kubectl rollout status deployments/kubernetes-bootcamp ✓
```

To view the current image version of the app, run a describe command against the Pods:

```
kubectl describe pods ✓
```

We run now version 2 of the app (look at the Image field)

Terminal +

Name:	kubernetes-bootcamp
Namespace:	default
Labels:	run=kubernetes-bootcamp
Annotations:	<none>
Selector:	run=kubernetes-bootcamp
Type:	NodePort
IP:	10.110.29.46
Port:	<unset> 8080/TCP
TargetPort:	8080/TCP
NodePort:	<unset> 31252/TCP
Endpoints:	172.18.0.10:8080,172.18.0.11:8080,172.1... 8.0.12:8080 + 1 more...
Session Affinity:	None
External Traffic Policy:	Cluster
Events:	<none>

```
$ export NODE_PORT=$(kubectl get services/kubernetes-bootcamp -o go-template='{{(index .spec.ports 0).nodePort}}')  
$ echo NODE_PORT=$NODE_PORT  
NODE_PORT=31252  
$ curl $(minikube ip):$NODE_PORT  
Hello Kubernetes bootcamp! | Running on: kubernetes-bootcamp-7d6f  
8694b6-wdrr6 | v=2  
$ kubectl rollout status deployments/kubernetes-bootcamp  
deployment "kubernetes-bootcamp" successfully rolled out  
$
```

Powered by  Katacoda

- 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.

## Module 6 - Update your app

### ◀ Step 3 of 3 ▶

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

Use `get deployments` to see the status of the deployment:

```
kubectl get deployments ✓
```

And something is wrong... We do not have the desired number of Pods available. List the Pods again:

```
kubectl get pods ✓
```

A `describe` command on the Pods should give more insights:

```
kubectl describe pods ↵
```

There is no image called `v10` in the repository. Let's roll back to our

Terminal +

```
kubernetes-bootcamp  
Normal Started 2m16s kubelet, minikube Started container  
kubernetes-bootcamp  
$ kubectl set image deployments/kubernetes-bootcamp kubernetes-bo  
otcamp=gcr.io/google-samples/kubernetes-bootcamp:v10  
deployment.apps/kubernetes-bootcamp image updated  
$ kubectl get deployments  
NAME READY UP-TO-DATE AVAILABLE AGE  
kubernetes-bootcamp 3/4 2 3 3m10s  
$ kubectl get pods  
NAME READY STATUS  
RESTARTS AGE  
kubernetes-bootcamp-7d6f8694b6-4z6t2 1/1 Running  
0 2m40s  
kubernetes-bootcamp-7d6f8694b6-f78p2 1/1 Terminating  
0 2m40s  
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  
$ [ ]
```

Powered by  Katacoda

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 known 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 +

NAME	READY	STATUS
kubernetes-bootcamp-7d6f8694b6-4z6t2	1/1	Running
kubernetes-bootcamp-7d6f8694b6-5j9wx	0/1	ContainerCreating
kubernetes-bootcamp-7d6f8694b6-grg6s	1/1	Running
kubernetes-bootcamp-7d6f8694b6-wdrr6	1/1	Running
kubernetes-bootcamp-886577c5d-c4dfj	0/1	Terminating
kubernetes-bootcamp-886577c5d-xcvb5	0/1	Terminating

\$ kubectl rollout undo deployments/kubernetes-bootcamp  
deployment.apps/kubernetes-bootcamp rolled back

\$ kubectl get pods

NAME	READY	STATUS
kubernetes-bootcamp-7d6f8694b6-4z6t2	1/1	Running
kubernetes-bootcamp-7d6f8694b6-5j9wx	0/1	ContainerCreating
kubernetes-bootcamp-7d6f8694b6-grg6s	1/1	Running
kubernetes-bootcamp-7d6f8694b6-wdrr6	1/1	Running
kubernetes-bootcamp-886577c5d-c4dfj	0/1	Terminating
kubernetes-bootcamp-886577c5d-xcvb5	0/1	Terminating

Powered by  Katacoda