

**Kubernetes** is an open-source platform from Goggle designed for the containerized applications to run when and where we want with the ease to update the applications for new features and bugs fixes while allowing these applications to run 24/7. Thus, Kubernetes allows the containerized applications to run as loosely coupled applications on a cluster of machines which are highly available and automates the distribution and scheduling of the application containers running on the cluster efficiently.

Kubernetes Cluster follows a Master Slave Architecture where master controls all the nodes, nodes are the worker nodes assigned tasks to be completed by the master.

## Module 1 - Create a Kubernetes Cluster

We check if Minikube is running properly or not. Minikube is a lightweight Kubernetes implementation that creates a VM and deploys a simple cluster containing only one node. Kubernetes abstracts the deployment of applications in a cluster without tying them specifically to a machine in a cluster creating a loosely coupled connection. Cluster is started using the “minikube start” command. To be able to communicate with Kubernetes Cluster, we use kubectl which is kind of same as normal command line interface like putty or Windows Powershell we would use to interact with the applications for running the commands. Kubectl version command is used to check if this CLI is up and running or not.

### Cluster up and running

The screenshot displays the Kubernetes.io documentation page for 'Module 1 - Create a Kubernetes cluster'. The page is titled 'Cluster up and running' and is part of a tutorial series. It includes a sidebar with navigation links such as 'Home', 'Getting started', 'Concepts', 'Tasks', 'Tutorials', 'Hello Minikube', 'Learn Kubernetes Basics', 'Create a Cluster', 'Interactive Tutorial - Creating a Cluster', 'Deploy an App', 'Explore Your App', 'Expose Your App Publicly', 'Scale Your App', 'Update Your App', 'Configuration', and 'Stateless'. The main content area shows a terminal window with the output of the 'minikube start' command. The terminal output indicates that minikube v1.8.1 is installed on Ubuntu 18.04, using the none driver, and is running on localhost. It also shows the preparation of Kubernetes v1.17.3 on Docker 19.03.6 and the launch of the cluster. The page includes a sidebar with navigation links and a 'Continue to Module 2' button.

### Cluster Version

The screenshot shows the Kubernetes documentation page for 'Module 1 - Create a Kubernetes cluster'. The page is titled 'Module 1 - Create a Kubernetes cluster' and is part of a 'Tutorial - Creating a Cluster'. The main content area shows the 'Cluster up and running' section, which includes instructions on how to start the cluster using minikube. A terminal window is open, showing the following commands and output:

```
$ 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=145651MB) ...
* 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", GitVersion:"v1.17.3", GitCommit:"06ad960b6fd03b39c8
310aaf92d1e7c12ce618213", 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:"06ad960b6fd03b39c8
310aaf92d1e7c12ce618213", GitTreeState:"clean", BuildDate:"2020-02-11T18:07:13Z", GoVersion:"go1.13.6"
, Compiler:"gc", Platform:"linux/amd64"}
$
```

Kubectl is configured on both the master and the nodes. kubectl version is the client version whereas Kubernetes version is the master installed on the master node. “kubectl cluster-info” command shows the cluster details indicating the master is running and the location at which the master is running. “kubectl get nodes” command is used to show all the nodes in the cluster to be used for deployment. We only have 1 node which is having Status as “Ready” meaning it is ready for deployment and it is a Master Node as indicated by the Roles.

### Cluster details

The screenshot shows the 'Cluster details' section of the Kubernetes documentation. The terminal window displays the following commands and output:

```
$ kubectl cluster-info
Kubernetes master is running at https://172.17.0.49:8443
KubeDNS is running at https://172.17.0.49:8443/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   4m    v1.17.3
$
```

## Module 2 - Deploy an App

Once the Kubernetes Cluster is up and running, we would deploy our containerized application on top of this cluster. This deployment instructs Kubernetes on how to create and update the instances of the application.

### kubectl basics

“kubectl version” command is used to check if the kubectl CLI is up and running for us to interact with the cluster to deploy our application along with checking the nodes in the cluster using the “kubectl get nodes”.

The screenshot shows the Kubernetes documentation page for 'Module 2 - Deploy an app'. The page is titled 'Module 2 - Deploy an app' and is part of the 'kubernetes.io/docs/tutorials/kubernetes-basics/deploy-app/deploy-interactive/' series. The left sidebar contains a navigation menu with links to Home, Getting started, Concepts, Tasks, Tutorials, Hello Minikube, Learn Kubernetes Basics, Create a Cluster, Deploy an App, Using kubectl to Create a Deployment, Interactive Tutorial - Deploying an App, Explore Your App, Expose Your App Publicly, Scale Your App, Update Your App, and Configuration. The main content area is titled 'Module 2 - Deploy an app' and includes a 'Step 1 of 3' indicator. The 'kubectl basics' section explains that like minikube, kubectl comes installed in the online terminal. It provides the command 'kubectl get nodes' and shows the output in a terminal window. The terminal output shows the client and server versions, and the output of 'kubectl get nodes' which shows a single node named 'minikube' in a 'Ready' state.

Module 2 - Deploy an app

Step 1 of 3

kubectl basics

Like minikube, kubectl comes installed in the online terminal. Type kubectl in the terminal to see its usage. The common format of a kubectl command is: kubectl action resource. This performs the specified action (like create, describe) on the specified resource (like node, container). You can use --help after the command to get additional info about possible parameters ( kubectl get nodes --

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", GitVersion:"v1.17.3", GitCommit:"06ad960b6fd83b39c8310aaf92d1e7c12ce618213", 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:"06ad960b6fd83b39c8310aaf92d1e7c12ce618213", 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   27s   v1.17.3
```

Powered by Katacoda

Deploying the application using the “kubectl create deployment” command with Kubernetes-bootcamp being the name of the deployment and image location of the base image using the image parameter. The Application deployment is successful using the image file specified which can be checked by using “kubectl get deployments” showing the deployment running on the single instance.

## Deploy our app

The screenshot shows the Kubernetes documentation page for 'Deploy our app'. The left sidebar contains a navigation menu with links to Explore Your App, Expose Your App Publicly, Scale Your App, Update Your App, and Configuration. The main content area is titled 'Deploy our app' and includes a 'command is:' section. It provides the command 'kubectl create deployment kubernetes-bootcamp --image=gcr.io/google-samples/kubernetes-bootcamp:v1' and shows the output in a terminal window. The terminal output shows the deployment created and the output of 'kubectl get deployments' which shows a single deployment named 'kubernetes-bootcamp' in a 'Ready' state.

command is:

kubectl action resource. This performs the specified action (like create, describe) on the

```
$ 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           16s
```

Powered by Katacoda

## View our app

We create proxy to communicate with the internal private and isolated network which cannot be accessed from the outside. When using kubectl we are interacting through API endpoint to interact with the application deployed on this private Kubernetes cluster. So, this proxy created would forward the communications to this internal private network for us to interact with the application from outside enabling direct access to all the API from the terminal. Terminal 2 opens up when starting the proxy.

Now, we are getting the POD Name and storing it in the environment variable `POD_NAME` for later use to connect to the containerized application running on that pod.

When we did the Deployment, Kubernetes by default created a Pod to host our application instance. A Pod is a Kubernetes abstraction that represents a group of one or more application containers and some shared resources for those containers which include shared storage,

networking etc. So, when create deployment on Kubernetes, Kubernetes by default creates a Pod which has containers in it. Each Pod is associated with a particular Node.

“kubectl get pods” commands shows the pods running whereas “kubectl describe pods” would give detailed description of the pods including the containers information which are located inside the pod, IP address, ports being used etc.

The screenshot shows the Kubernetes documentation page for 'Module 3 - Explore your app'. The terminal output displays the result of the command `kubectl describe pods` for a pod named `kubernetes-bootcamp-765bf4c7b4-9kf6g`. The output includes details such as Name, Namespace, Priority, Node, Start Time, Labels, Annotations, Status, IP, and Containers.

```
$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
kubernetes-bootcamp-765bf4c7b4-9kf6g 1/1     Running   0           58s

$ kubectl describe pods
Name:         kubernetes-bootcamp-765bf4c7b4-9kf6g
Namespace:    default
Priority:      0
Node:         minikube/172.17.0.10
Start Time:   Fri, 16 Oct 2020 03:48:30 +0000
Labels:       pod-template-hash=765bf4c7b4
              run=kubernetes-bootcamp
Annotations:  <none>
Status:       Running
IP:           172.18.0.4
IPs:          172.18.0.4
Controlled By: ReplicaSet/kubernetes-bootcamp-765bf4c7b4
Containers:
  kubernetes-bootcamp:
    Container ID:  docker://9a059ee928af09cf9b291efb7b02ebc23e0ead231df42d074524f554eb88f0
    Image:         gcr.io/google-samples/kubernetes-bootcamp:v1
    Image ID:      docker-pullable://jocatalin/kubernetes-bootcamp@sha256:0d6b8ee3bb57c5f5b6156f446b3bc3b3c143d233037f3azf0be279c8fcc04af
    Port:          8080/TCP
    Host Port:     0/TCP
```

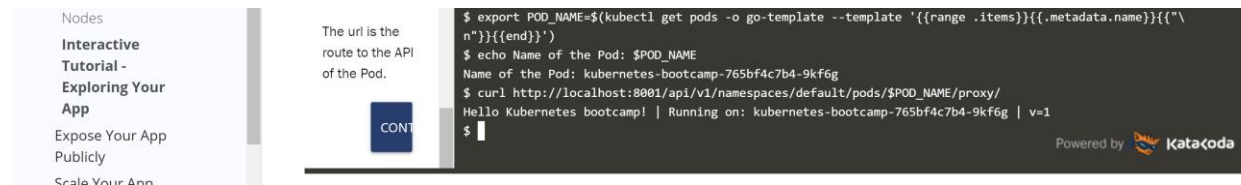
The screenshot shows the Kubernetes documentation page for 'Module 3 - Explore your app'. The terminal output displays the result of the command `kubectl describe pods` for a pod named `kubernetes-bootcamp-765bf4c7b4-9kf6g`. The output includes details such as ContainersReady, PodScheduled, Volumes, QoS Class, Node-Selectors, Tolerations, and Events.

```
ContainersReady: True
PodScheduled:    True
Volumes:
  default-token-fgdcg:
    Type:      Secret (a volume populated by a Secret)
    SecretName: default-token-fgdcg
    Optional:  false
QoS Class:     BestEffort
Node-Selectors: <none>
Tolerations:   node.kubernetes.io/not-ready:NoExecute for 300s
               node.kubernetes.io/unreachable:NoExecute for 300s

Events:
Type      Reason      Age      From      Message
----      -
Warning   FailedScheduling 67s (x2 over 67s) default-scheduler 0/1 nodes are available: 1 node(s) had taints that the pod didn't tolerate.
Normal    Scheduled    61s      default-scheduler Successfully assigned default/kubernetes-bootcamp-765bf4c7b4-9kf6g to minikube
Normal    Pulled       58s      kubelet, minikube Container image "gcr.io/google-samples/kubernetes-bootcamp:v1" already present on machine
Normal    Created      58s      kubelet, minikube Created container kubernetes-bootcamp
Normal    Started      58s      kubelet, minikube Started container kubernetes-bootcamp
```

Show the app in the terminal

We are getting the POD Name and querying the POD directly through the use of proxy. In order to be able to see the output of the application, we use curl request running on port 8001 with the url being the way to the API of the POD.



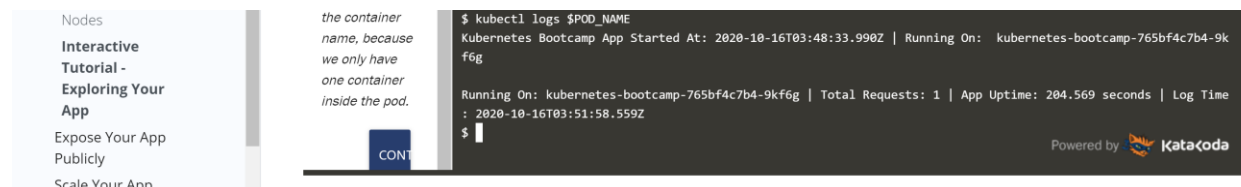
The screenshot shows the Katacoda interface with a sidebar on the left containing navigation links: Nodes, Interactive Tutorial - Exploring Your App, Expose Your App Publicly, and Scale Your App. The main area has a text box stating "The url is the route to the API of the Pod." with a "CONT" button below it. The terminal window displays the following commands and output:

```
$ 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-765bf4c7b4-9kf6g
$ curl http://localhost:8001/api/v1/namespaces/default/pods/$POD_NAME/proxy/
Hello Kubernetes bootcamp! | Running on: kubernetes-bootcamp-765bf4c7b4-9kf6g | v=1
$
```

Powered by Katacoda

View the container logs

“kubectl logs \$POD\_NAME” command is used to get/retrieve the logs of the container inside the POD. Since we have only 1 POD we are not explicitly specifying the POD Name.



The screenshot shows the Katacoda interface with a sidebar on the left containing navigation links: Nodes, Interactive Tutorial - Exploring Your App, Expose Your App Publicly, and Scale Your App. The main area has a text box stating "the container name, because we only have one container inside the pod." with a "CONT" button below it. The terminal window displays the following commands and output:

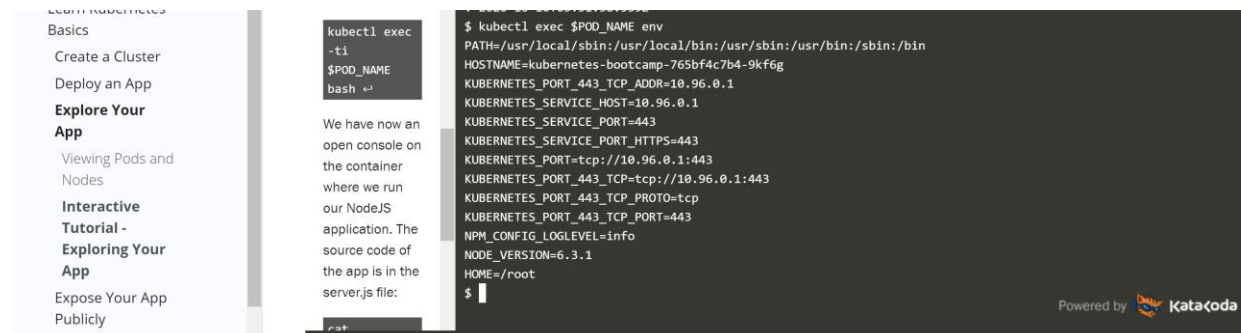
```
$ kubectl logs $POD_NAME
Kubernetes Bootcamp App Started At: 2020-10-16T03:48:33.996Z | Running On: kubernetes-bootcamp-765bf4c7b4-9kf6g

Running On: kubernetes-bootcamp-765bf4c7b4-9kf6g | Total Requests: 1 | App Uptime: 204.569 seconds | Log Time
: 2020-10-16T03:51:58.559Z
$
```

Powered by Katacoda

Executing command on the container

Executing the commands directly on the container contained in the pod once the pod is up and running. To execute the commands, we use “exec” command.



The screenshot shows the Katacoda interface with a sidebar on the left containing navigation links: Basics, Create a Cluster, Deploy an App, Explore Your App, Viewing Pods and Nodes, Interactive Tutorial - Exploring Your App, Expose Your App Publicly, and Scale Your App. The main area has a text box stating "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:" with a "CONT" button below it. The terminal window displays the following commands and output:

```
$ kubectl exec $POD_NAME env
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOSTNAME=kubernetes-bootcamp-765bf4c7b4-9kf6g
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

Running the Nodejs application using the open console on the container after starting a bash session. The source code the application is in server.js file and this application can be checked to be up and running using the curl command.



The screenshot shows the Kubernetes documentation page for 'Module 3 - Explore your app'. The page is titled 'Module 3 - Explore your app' and is part of the 'Interactive Tutorial - Exploring Your App'. The left sidebar contains a search bar and a list of navigation links: Home, Getting started, Concepts, Tasks, Tutorials, Hello Minikube, Learn Kubernetes Basics, Create a Cluster, Deploy an App, Explore Your App, Viewing Pods and Nodes, Interactive Tutorial - Exploring Your App, Expose Your App Publicly, and Scale Your App. The main content area shows a terminal window with the following commands and output:

```
$ kubectl exec -ti $POD_NAME bash
root@kubernetes-bootcamp-765bf4c7b4-9kf6g:/# 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("\n | 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" );
});
root@kubernetes-bootcamp-765bf4c7b4-9kf6g:/# curl localhost:8080
Hello Kubernetes bootcamp! | Running on: kubernetes-bootcamp-765bf4c7b4-9kf6g | v=1
root@kubernetes-bootcamp-765bf4c7b4-9kf6g:/#
```

The terminal output shows the application is running on the pod 'kubernetes-bootcamp-765bf4c7b4-9kf6g' and is listening on port 8080. The application is a simple HTTP server that responds with 'Hello Kubernetes bootcamp! | Running on: kubernetes-bootcamp-765bf4c7b4-9kf6g | v=1'.

## Module 4 - Expose Your App Publicly

Service in Kubernetes is an abstraction to define logical set of pods and a policy through which we can access this logical set of pods enabling the loose coupling between the dependent pods. Services route the traffic across a set of dependent pods.

Create a new service

We check for existing pods using the “kubectl get pods” command. “kubectl get services” gives the list of current services from the cluster. We have a Kubernetes service running which is created by default when minikube starts the cluster.

To create a new service and expose it to the external traffic we use “kubectl expose” command with NodePort as a parameter. We now have successfully created this new service.

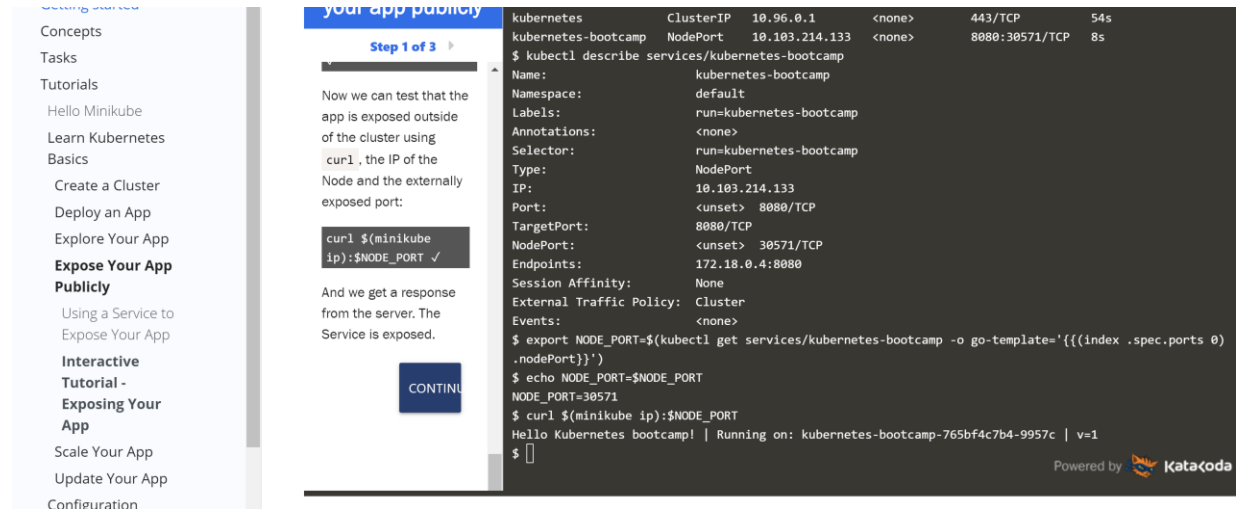
The screenshot shows the Kubernetes documentation page for 'Module 4 - Expose your app publicly'. The page is titled 'Module 4 - Expose your app publicly' and is part of the 'Interactive Tutorial - Exposing Your App'. The left sidebar contains a search bar and a list of navigation links: Home, Getting started, Concepts, Tasks, Tutorials, Hello Minikube, Learn Kubernetes Basics, Create a Cluster, Deploy an App, Expose Your App Publicly, Using a Service to Expose Your App, Interactive Tutorial - Exposing Your App, and Scale Your App. The main content area shows a terminal window with the following commands and output:

```
sleep 1; launch.sh
$
$ sleep 1; launch.sh
Starting Kubernetes. This is expected to take less than a minute..
.....
Kubernetes Started
$
$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
kubernetes-bootcamp-765bf4c7b4-9957c  0/1     ContainerCreating  0          16s
$ kubectl get services
NAME    TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes  ClusterIP   10.96.0.1    <none>         443/TCP   37s
$ kubectl expose deployment/kubernetes-bootcamp --type="NodePort" --port 8080
service/kubernetes-bootcamp exposed
$ kubectl get services
NAME    TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes  ClusterIP   10.96.0.1    <none>         443/TCP   54s
kubernetes-bootcamp  NodePort    10.103.214.133  <none>         8080:30571/TCP  8s
```

The terminal output shows the application is running on the pod 'kubernetes-bootcamp-765bf4c7b4-9957c' and is listening on port 8080. The application is a simple HTTP server that responds with 'Hello Kubernetes bootcamp! | Running on: kubernetes-bootcamp-765bf4c7b4-9kf6g | v=1'.

We describe the service using “kubectl describe service” command to get more details about the service such as the port on which this service is running, IP address etc.

To check if the application is exposed to external traffic, we use the UP address and the port mentioned in the describe command output to access the application using the curl command.



your app publicly

Step 1 of 3

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.

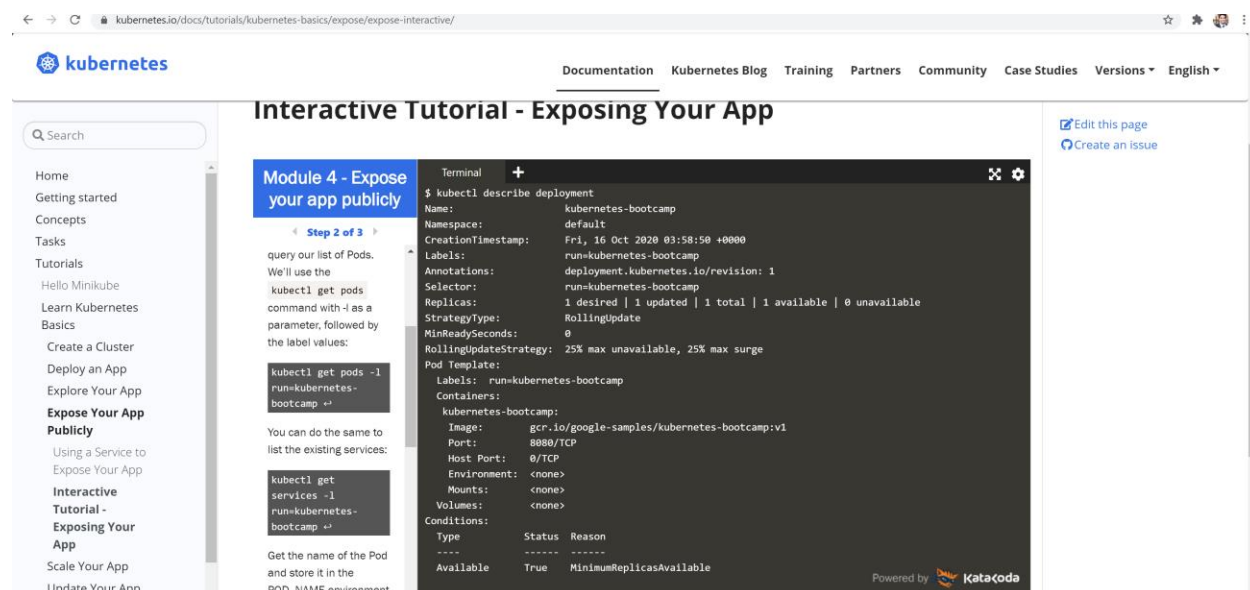
CONTINUE

```
kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 54s
kubernetes-bootcamp NodePort 10.103.214.133 <none> 8080:30571/TCP 8s
$ kubectl describe services/kubernetes-bootcamp
Name: kubernetes-bootcamp
Namespace: default
Labels: run=kubernetes-bootcamp
Annotations: <none>
Selector: run=kubernetes-bootcamp
Type: NodePort
IP: 10.103.214.133
Port: <unset> 8080/TCP
TargetPort: 8080/TCP
NodePort: <unset> 30571/TCP
Endpoints: 172.18.0.4:8080
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=30571
$ curl $(minikube ip):$NODE_PORT
Hello Kubernetes bootcamp! | Running on: kubernetes-bootcamp-765bf4c7b4-9957c | v=1
$
```

Powered by KataCoda

## Using labels

When the deployment was created, the pod was automatically given a label which can be viewed using the “kubectl describe deployment” command.



Interactive Tutorial - Exposing Your App

Module 4 - Expose your app publicly

Step 2 of 3

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
```

You can do the same to list the existing services:

```
kubectl get services -l run=kubernetes-bootcamp
```

Get the name of the Pod and store it in the `POD_NAME` environment

```
$ kubectl describe deployment
Name: kubernetes-bootcamp
Namespace: default
CreationTimestamp: Fri, 16 Oct 2020 03:58:50 +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
      Port: 8080/TCP
      Host Port: 0/TCP
      Environment: <none>
      Mounts: <none>
      Volumes: <none>
  Conditions:
    Type Status Reason
    ----
    Available True MinimumReplicasAvailable
```

Powered by KataCoda

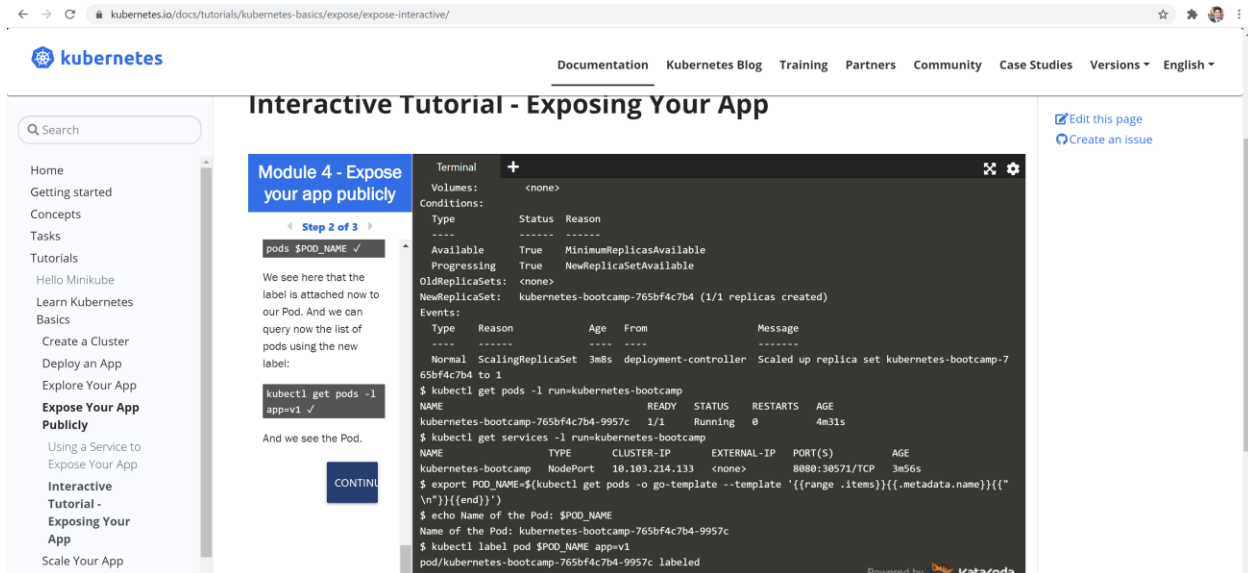
Using the pod label, we got from the above to query the list of pods using the parameter `-l` to specify the pod label to get its details. To get the service details, same “kubectl get” command is used. Also, storing the pod name in the environment variable `POD_NAME`.



We can also change the name of the label using the following command:

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

New label is assigned to the current POD Name stored in the POD\_NAME variable.

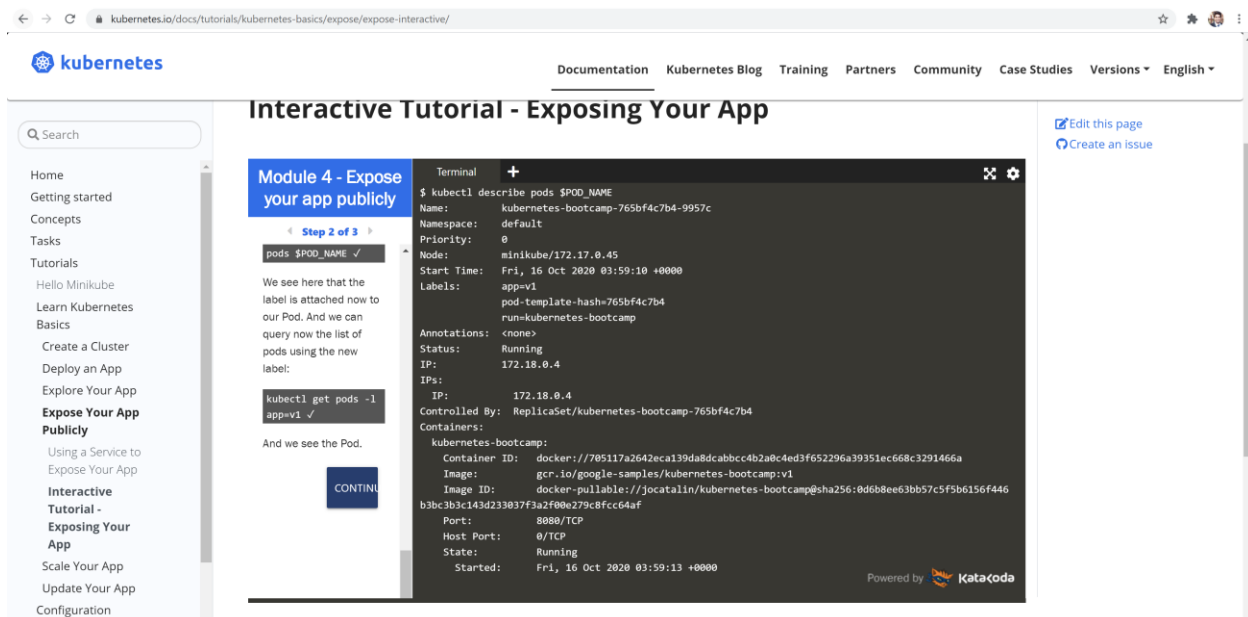


The screenshot shows the Kubernetes Interactive Tutorial interface. On the left, a sidebar lists navigation options like Home, Getting started, Concepts, Tasks, Tutorials, and a list of tutorial steps. The main content area is titled 'Interactive Tutorial - Exposing Your App' and 'Module 4 - Expose your app publicly'. It shows 'Step 2 of 3' with a 'CONTINUE' button. The terminal window displays the following commands and output:

```
$ kubectl get pods -l app=v1 ✓
And we see the Pod.
CONTINUE
```

```
Terminal
Volumes: <none>
Conditions:
Type      Status Reason
-----
Available True   MinimumReplicasAvailable
Progressing True  NewReplicaSetAvailable
OldReplicaSets: <none>
NewReplicaSet: kubernetes-bootcamp-765bf4c7b4 (1/1 replicas created)
Events:
Type      Reason      Age      From          Message
-----
Normal    ScalingReplicaSet  3m8s    deployment-controller    Scaled up replica set kubernetes-bootcamp-765bf4c7b4 to 1
$ kubectl get pods -l run=kubernetes-bootcamp
NAME                                READY STATUS RESTARTS AGE
kubernetes-bootcamp-765bf4c7b4-9957c 1/1    Running 0       4m31s
$ kubectl get services -l run=kubernetes-bootcamp
NAME      TYPE      CLUSTER-IP    EXTERNAL-IP  PORT(S)    AGE
kubernetes-bootcamp NodePort  10.103.214.133 <none>       8080:30571/TCP 3m56s
$ 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-765bf4c7b4-9957c
$ kubectl label pod $POD_NAME app=v1
pod/kubernetes-bootcamp-765bf4c7b4-9957c labeled
```

Now describing the pod again, we can see that new label is added to the pod while the old label has not been deleted yet.



The screenshot shows the Kubernetes Interactive Tutorial interface. On the left, a sidebar lists navigation options like Home, Getting started, Concepts, Tasks, Tutorials, and a list of tutorial steps. The main content area is titled 'Interactive Tutorial - Exposing Your App' and 'Module 4 - Expose your app publicly'. It shows 'Step 2 of 3' with a 'CONTINUE' button. The terminal window displays the following commands and output:

```
$ kubectl describe pod $POD_NAME
Name: kubernetes-bootcamp-765bf4c7b4-9957c
Namespace: default
Priority: 0
Node: minikube/172.17.0.45
Start Time: Fri, 16 Oct 2020 03:59:10 +0000
Labels: app=v1
pod-template-hash=765bf4c7b4
run=kubernetes-bootcamp
Annotations: <none>
Status: Running
IP: 172.18.0.4
IPs: 172.18.0.4
Controlled By: ReplicaSet/kubernetes-bootcamp-765bf4c7b4
Containers:
  kubernetes-bootcamp:
    Container ID: docker://705117a2642eca139da8dcabbc4b2a8c4ed3f652296a39351ec668c3291466a
    Image: gcr.io/google-samples/kubernetes-bootcamp:v1
    Image ID: docker-pullable://jocatalin/kubernetes-bootcamp@sha256:8d6b8ee63bb57c5f5b6156f446b3bc3b3c143d233037f3a2f00e279c8fcc64af
    Port: 8080/TCP
    Host Port: 0/TCP
    State: Running
    Started: Fri, 16 Oct 2020 03:59:13 +0000
```

**Module 4 - Expose your app publicly**

**Step 2 of 3**

Pods: \$POD\_NAME ✓

We see here that the label is attached now to our Pod. And we can query now the list of pods using the new label:

```
kubectl get pods -l app=v1 ✓
```

And we see the Pod.

**CONTINUE**

**Terminal**

```
default-token-kxbgw:
Type: Secret (a volume populated by a Secret)
SecretName: default-token-kxbgw
Optional: false
QoS Class: BestEffort
Node-Selectors: <none>
Tolerations: node.kubernetes.io/not-ready:NoExecute for 300s
              node.kubernetes.io/unreachable:NoExecute for 300s

Events:
Type Reason Age From Message
----
Warning FailedScheduling 4m36s (x3 over 4m44s) default-scheduler 0/1 nodes are available: 1 node(s) had taints that the pod didn't tolerate.
Normal Scheduled 4m32s default-scheduler Successfully assigned default/kubernetes-bootcamp-765bf4c7b4-9957c to minikube
Normal Pulled 4m29s kubelet, minikube Container image "gcr.io/google-samples/kubernetes-bootcamp:v1" already present on machine
Normal Created 4m28s kubelet, minikube Created container kubernetes-bootcamp
Normal Started 4m28s kubelet, minikube Started container kubernetes-bootcamp

$ kubectl get pods -l app=v1
NAME                                READY STATUS RESTARTS AGE
kubernetes-bootcamp-765bf4c7b4-9957c 1/1 Running 0 4m47s
```

Powered by **Katacoda**

## Deleting a service

We now delete the service using the “kubectl delete service” command and if the service is deleted or not using the “kubectl get services” command.

Now to check if the app is exposed to the external traffic which was achieved by using the service which was deleted, we used “curl” command which shows it is not accessible from the outside. But it can be accessed internally using the kubectl with the app being up and running.

**Module 5 - Scale Your App**

**Expose Your App Publicly**

Using a Service to Expose Your App

**Interactive Tutorial - Exposing Your App**

Scale Your App

Update Your App

Configuration

**localhost:8080 ✓**

We see here that the application is up. This is because the Deployment is managing the application. To shut down the application, you would need to delete the Deployment as well.

**CONTINUE**

**Terminal**

```
kubernetes-bootcamp-765bf4c7b4-9957c 1/1 Running 0 4m47s
$ kubectl get pods -l app=v1
NAME                                READY STATUS RESTARTS AGE
kubernetes-bootcamp-765bf4c7b4-9957c 1/1 Running 0 7m24s
$ 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 7m43s
$ curl $(minikube ip):$NODE_PORT
curl: (7) Failed to connect to 172.17.0.45 port 30571: Connection refused
$ kubectl exec -ti $POD_NAME curl localhost:8080
Hello Kubernetes bootcamp! | Running on: kubernetes-bootcamp-765bf4c7b4-9957c | v=1
$
```

Powered by **Katacoda**

## Module 5 - Scale Your App

### Scaling a deployment

After deploying the app and creating service to expose the app to the outside world, we only have 1 pod. With the growing needs of the business and when the traffic increases, we would need to scale our application to accept more number of requests than before causing us to scale up the application.

Scaling up means creating new pods as and when required. Multiple instances of an application can be run by distributing the load among these pods and we can perform Rolling updates without downtime as the other pods would be working to cater to the business without downtime.

We first check the deployments which shows only 1 pod running. We scale up the deployment by increasing the replicas in the deployment to now 4 up and running instances. These 4 instances along with their different IP addresses is visible by using “kubectl get pods -o wide” command. “kubectl get rs” command gives the ReplicaSet information now there is only 1 replica available.

**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**

```
$ 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           20s
$ kubectl get rs
NAME          DESIRED   CURRENT   READY   AGE
kubernetes-bootcamp-765bf4c7b4  1         1         1       23s
$ kubectl scale deployments/kubernetes-bootcamp --replicas=4
deployment.apps/kubernetes-bootcamp scaled
$ kubectl get deployments
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
kubernetes-bootcamp  4/4     4            4           47s
$ kubectl get pods -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP              NODE
NOMINATED NODE   READINESS GATES
kubernetes-bootcamp-765bf4c7b4-6hj92  1/1     Running   0          37s   172.18.0.9      minikube
<none>              <none>
kubernetes-bootcamp-765bf4c7b4-j8rpd  1/1     Running   0          37s   172.18.0.8      minikube
<none>              <none>
kubernetes-bootcamp-765bf4c7b4-rw8vr  1/1     Running   0          37s   ...             ...
Powered by Katacoda
```

Describe command shows changes in the deployments events log

**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:

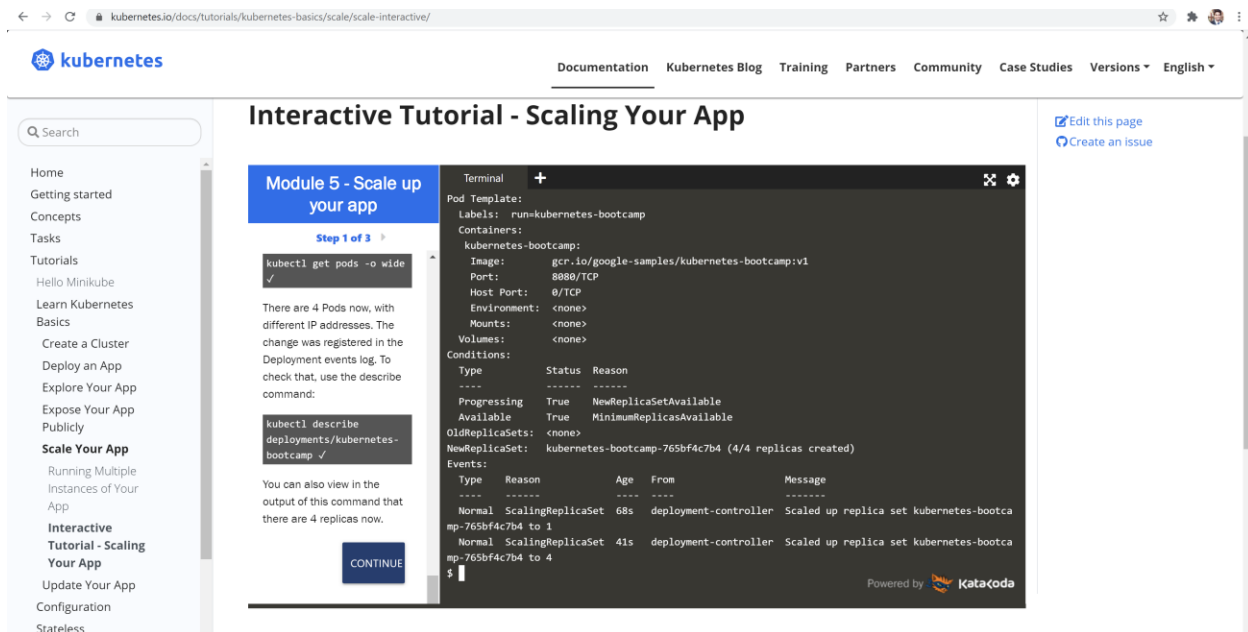
**kubectl describe deployments/kubernetes-bootcamp**

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:  Fri, 16 Oct 2020 04:08:52 +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:  8/TCP
      Environment:
      Mounts:     <none>
      Volumes:    <none>
  Conditions:
    Type           Status  Reason
    ----           -
    Progressing    True   NewReplicaSetAvailable
Powered by Katacoda
```



Interactive Tutorial - Scaling Your App

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

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
Progressing	True	NewReplicaSetAvailable
Available	True	MinimumReplicasAvailable

OldReplicaSets: <none>

NewReplicaSet: kubernetes-bootcamp-765bf4c7b4 (4/4 replicas created)

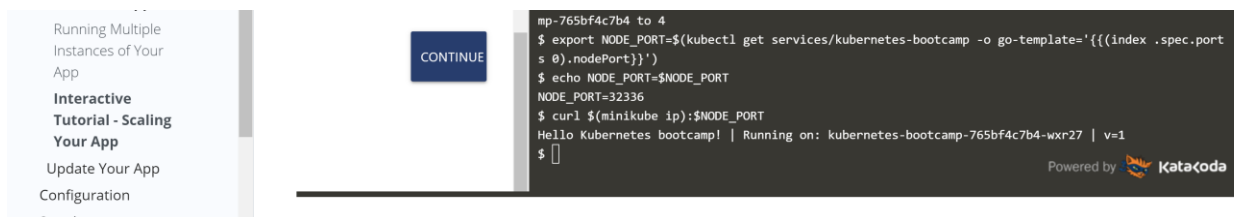
Events:

Type	Reason	Age	From	Message
Normal	ScalingReplicaSet	68s	deployment-controller	Scaled up replica set kubernetes-bootcamp-765bf4c7b4 to 4
Normal	ScalingReplicaSet	41s	deployment-controller	Scaled up replica set kubernetes-bootcamp-765bf4c7b4 to 4

Powered by KataCoda

## Load Balancing

Created an environment variable for storing port of the node to access the node using the curl command. Curl command reaches/hits the different pods with different IP addresses when trying to access the application showing the Load Balancing is working perfectly.



Running Multiple Instances of Your App

Interactive Tutorial - Scaling Your App

`CONTINUE`

`mp-765bf4c7b4 to 4`

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

`$ echo NODE_PORT=$NODE_PORT`

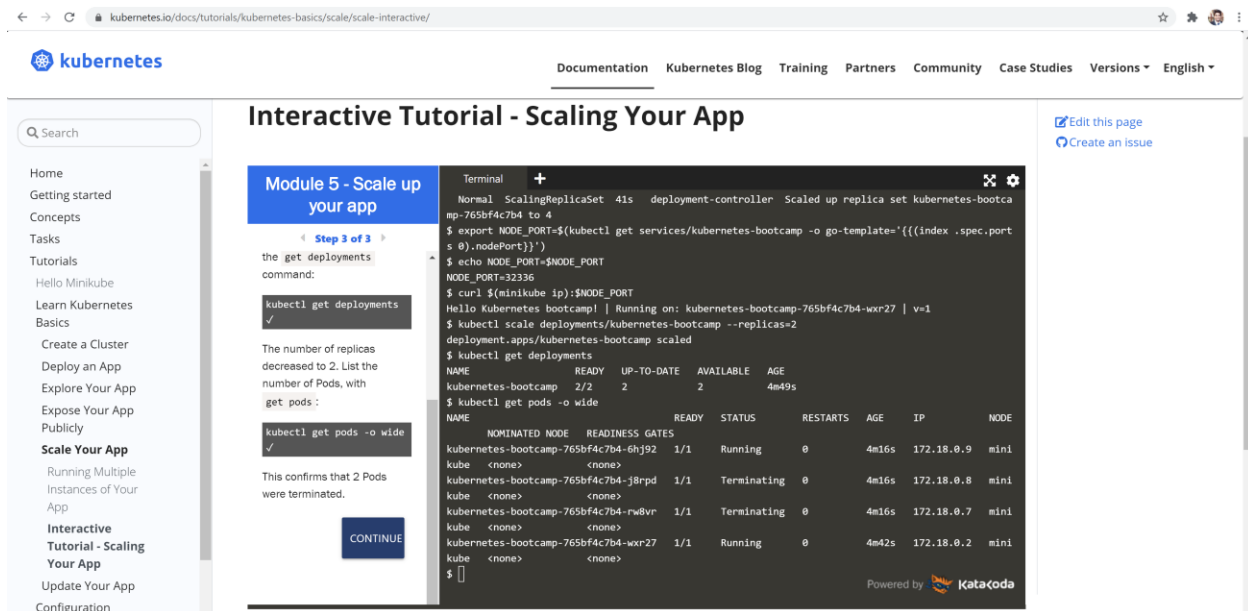
`NODE_PORT=32336`

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

Hello Kubernetes bootcamp! | Running on: kubernetes-bootcamp-765bf4c7b4-wxr27 | v=1

Powered by KataCoda

As we scaled up the application when the number of requests increased, we can scale down the application by reducing the deployment replicas when the hits can be handled by smaller number of pods. We have scaled down the application to 2 replicas from 4 and both of them are up and running as shown below.



**Module 5 - Scale up your app**

**Step 3 of 3**

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.

**CONTINUE**

**Terminal**

```
Normal ScalingReplicaSet 41s deployment-controller Scaled up replica set kubernetes-bootcamp-765bf4c7b4 to 4
$ export NODE_PORT=$(kubectl get services/kubernetes-bootcamp -o go-template='{{{(index .spec.port 0).nodePort}}}')
$ echo NODE_PORT=$NODE_PORT
NODE_PORT=32336
$ curl $(minikube ip):$NODE_PORT
Hello Kubernetes bootcamp! | Running on: kubernetes-bootcamp-765bf4c7b4-wxr27 | 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           4m49s
$ kubectl get pods -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP            NODE
kubernetes-bootcamp-765bf4c7b4-6hj92  1/1     Running   0           4m16s  172.18.0.9    mini
kubernetes-bootcamp-765bf4c7b4-j8rpd  1/1     Terminating 0           4m16s  172.18.0.8    mini
kubernetes-bootcamp-765bf4c7b4-rw8vr  1/1     Terminating 0           4m16s  172.18.0.7    mini
kubernetes-bootcamp-765bf4c7b4-wxr27  1/1     Running   0           4m42s  172.18.0.2    mini
```

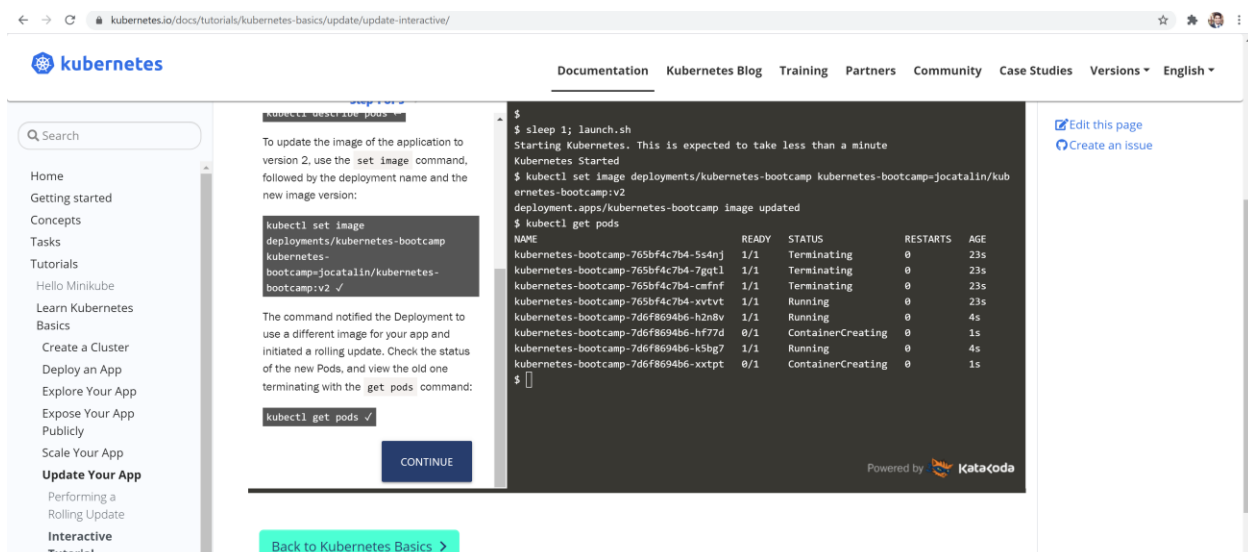
Powered by **Katacoda**

## Module 6 - Update Your App

Business users want the application to be running 24/7 along with developers enhance the user experience by updating the applications to new version to fix bugs or have new functionalities without downtime. Rolling updates allow to update the pods one after other to maintain zero downtime.

Update the version of the app

We are updating the image of the application to the newer version using the “`kubectl set image`” command. This command is followed by name of the deployment and the new image version in this case to v2 from v1. To check the status of the new pods running and the old ones being terminated we use “`get pods`” command.



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 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**

```
$
$ sleep 1; launch.sh
Starting Kubernetes. This is expected to take less than a minute
Kubernetes Started
$ 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-765bf4c7b4-5s4nj  1/1     Terminating 0           23s
kubernetes-bootcamp-765bf4c7b4-7gqt1  1/1     Terminating 0           23s
kubernetes-bootcamp-765bf4c7b4-cmfxf  1/1     Terminating 0           23s
kubernetes-bootcamp-765bf4c7b4-xvtvt  1/1     Running     0           4s
kubernetes-bootcamp-7d6f8694b6-h2n8v  1/1     Running     0           1s
kubernetes-bootcamp-7d6f8694b6-hf77d  0/1     ContainerCreating 0           4s
kubernetes-bootcamp-7d6f8694b6-k5bg7  1/1     Running     0           1s
kubernetes-bootcamp-7d6f8694b6-xttpt  0/1     ContainerCreating 0           1s
```

Powered by **Katacoda**

[Back to Kubernetes Basics >](#)

We store the node port in a variable and get the IP address, using the curl command to access the app to verify its running status. We can see that the pod is running with the newer version 2. To check the status of the successful rollout of the update, we use the command “`kubectl rollout status Name of the deployment`”. Rolling update is done successfully as can be seen below.

[illegible]

## Rollback an update

We now perform another rolling update to the newer version v10. But by using the `get deployments` command we are not able to see the desired number of pods. Also, by checking the pods, we can see there is no update rolled out as the image for the newer update to version v10 is not found in the repository. So, rollout is not done properly as indicated.

Documentation

Kubernetes Blog

Training

Partners

Community

Case Studies

Versions

English

Search

Home

Getting started

Concepts

Tasks

Tutorials

Hello Minikube

Learn Kubernetes Basics

Create a Cluster

Deploy an App

Explore Your App

Expose Your App

Publicly

Kubernetes Documentation

Tutorials

Learn Kubernetes Basics

Update Your App

Interactive Tutorial - Updating Your App

Module 6 - Update your app

Step 3 of 3

repository. Let's roll back to our previously working version. We'll use 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

Terminal

```
$ kubectl set image deployments/kubernetes-bootcamp kubernetes-bootcamp=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           3m18s
$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
kubernetes-bootcamp-7d6f8694b6-h2n8v 1/1     Running   0           2m52s
kubernetes-bootcamp-7d6f8694b6-hf77d 1/1     Running   0           2m49s
kubernetes-bootcamp-7d6f8694b6-k5bg7 1/1     Running   0           2m52s
kubernetes-bootcamp-7d6f8694b6-xxtp1 1/1     Terminating 0           2m49s
kubernetes-bootcamp-886577c5d-psg7f 0/1     ErrImagePull 0           13s
kubernetes-bootcamp-886577c5d-qblq9 0/1     ErrImagePull 0           13s
```

Edit this page

Create an issue



We use the “rollout undo” command to go back to the previous version and not do this rollout update to the version v10. After the rollout undo is done to go back to the previous state, all the pods are running successfully as we can see.

Scale Your App

**Update Your App**

Performing a Rolling Update

Interactive Tutorial - Updating Your App

Configuration

Stateless

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.

```
$ kubectl rollout undo deployments/kubernetes-bootcamp
deployment.apps/kubernetes-bootcamp rolled back
$ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
kubernetes-bootcamp-7d6f8694b6-h2n8v	1/1	Running	0	3m4s
kubernetes-bootcamp-7d6f8694b6-hf77d	1/1	Running	0	3m1s
kubernetes-bootcamp-7d6f8694b6-hrmg6	0/1	ContainerCreating	0	1s
kubernetes-bootcamp-7d6f8694b6-k5bg7	1/1	Running	0	3m4s
kubernetes-bootcamp-7d6f8694b6-xxtpf	1/1	Terminating	0	3m1s
kubernetes-bootcamp-886577c5d-psg7f	0/1	Terminating	0	25s
kubernetes-bootcamp-886577c5d-qblq9	0/1	Terminating	-	--

Powered by  Katacoda

## References:

<https://kubernetes.io/docs/tutorials/kubernetes-basics/> - Kubernetes Basics Tutorial