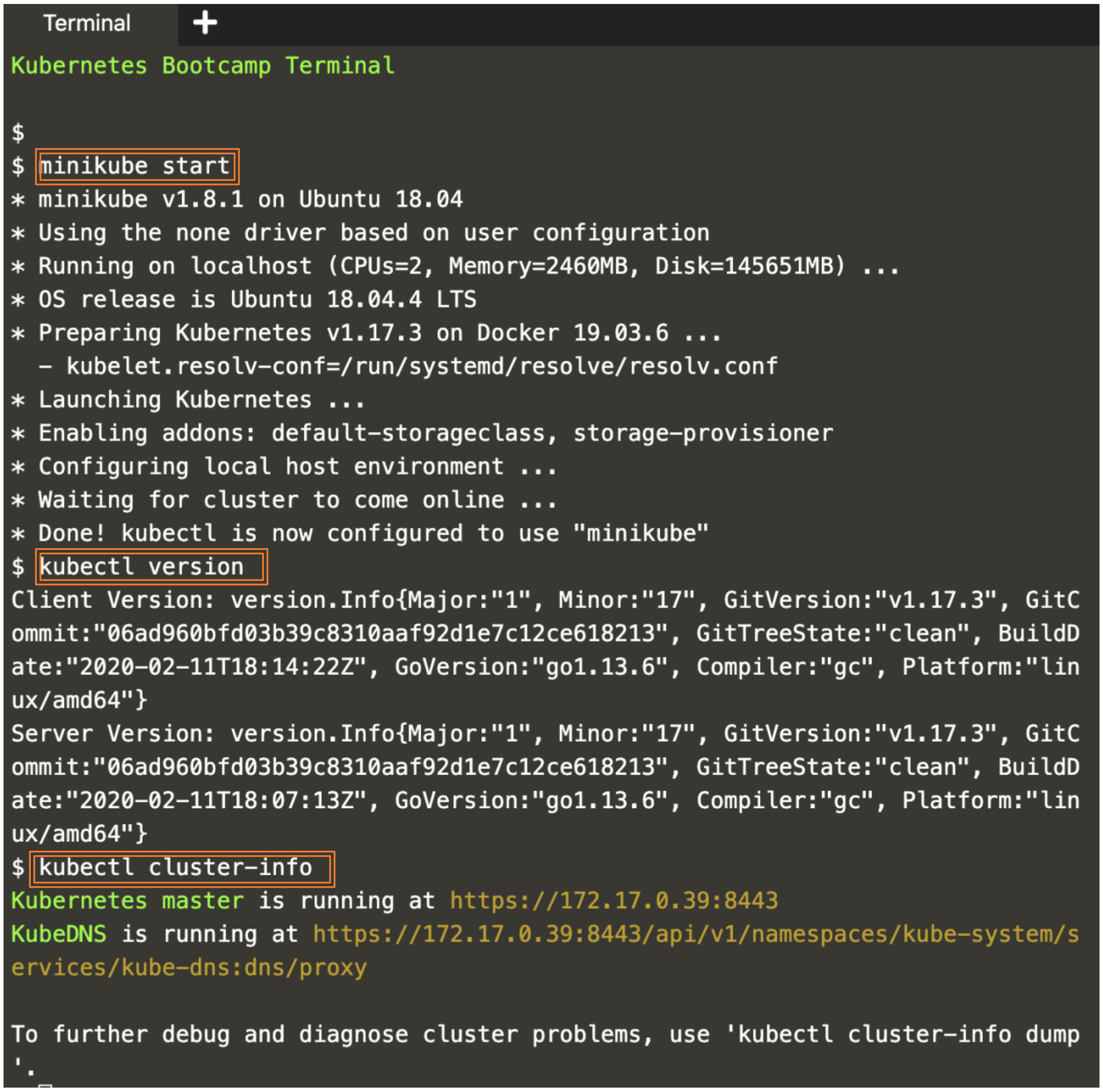
Akshay Bhala
abhala@syr.edu

IST 615

Module 1: Creating a Kubernetes Cluster

Minikube start command starts a VM for us and a Kubernetes cluster is now running in that VM.
1. But what is Kubernetes Cluster?
- A **Kubernetes cluster** is a set of nodes that run containerized applications.
To run and interact with Kubernetes we require command line interface. This command line interface is called **kubectl.**
Kubectl Version command simply inform us that kubectl is installed and we can see two versions:

a. Client version is kubectl version.
b. Server version is Kubernetes version which is installed on Master.



```
Kubernetes Bootcamp Terminal

$
$ 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", GitC
ommit:"06ad960bfd03b39c8310aaf92d1e7c12ce618213", GitTreeState:"clean", BuildD
ate:"2020-02-11T18:14:22Z", GoVersion:"go1.13.6", Compiler:"gc", Platform:"lin
ux/amd64"}
Server Version: version.Info{Major:"1", Minor:"17", GitVersion:"v1.17.3", GitC
ommit:"06ad960bfd03b39c8310aaf92d1e7c12ce618213", GitTreeState:"clean", BuildD
ate:"2020-02-11T18:07:13Z", GoVersion:"go1.13.6", Compiler:"gc", Platform:"lin
ux/amd64"}
$ kubectl cluster-info
Kubernetes master is running at https://172.17.0.39:8443
KubeDNS is running at https://172.17.0.39:8443/api/v1/namespaces/kube-system/s
ervices/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump
'.
```

Akshay Bhala
abhala@syr.edu

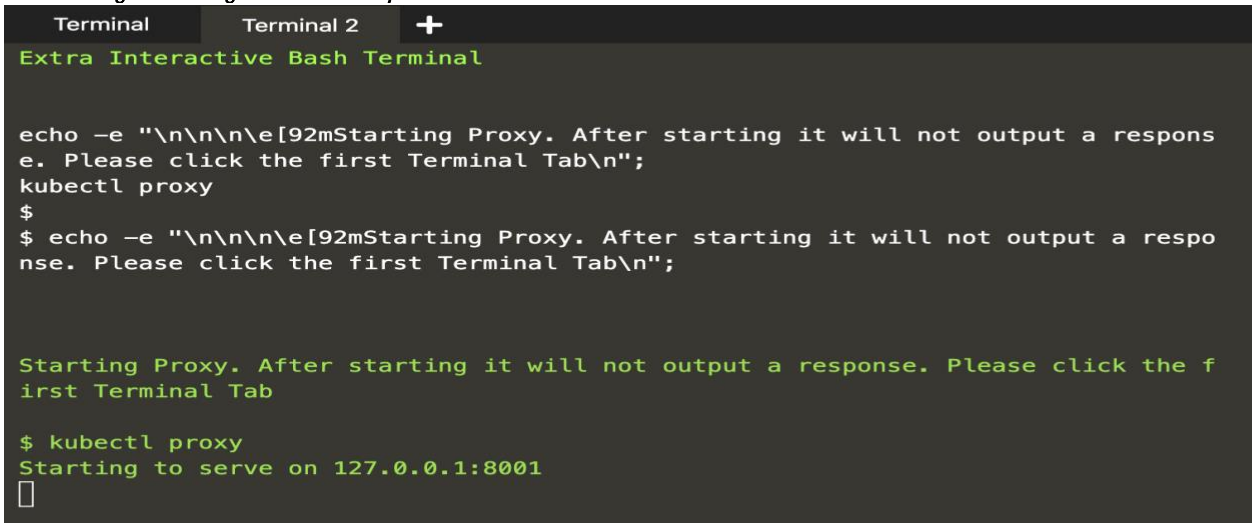
IST 615

Module 2: Deploy an App

- The Goal of this module is to deploy our app on k=Kubernetes using Kubectl.
- We first check whether our kubectl is configured to talk to our custer by running kubectl version command.
- Nodes are the workers that run applications. Nodes use Kubernetes API to communicate with the master. **kubectl get nodes** command informed us about the nodes available to host an application and in our scenario we can see only one node 'minikube'. **One thing to observe is the role of the node is stated as Master why?** In Kubernetes world, having a master and a worker/minion node creates a cluster, so you can either create a cluster at the initiation of the master node or Kubernetes will by **default create a default cluster for the new master node** – every cluster must have at least one master and exactly one active master node running the management of the cluster/worker nodes
- Deployments represent a set of multiple, identical Pods with no unique identities. A Deployment runs multiple replicas of your application and automatically replaces any instances that fail or become unresponsive. In this way, Deployments help ensure that one or more instances of your application are available to serve user requests. Deployments are managed by the Kubernetes Deployment controller.
- By providing an app image location and name from the repository hosted outside Docker hub using **kubectl create deployment** command we have created the deployment and Kubernetes schedules the app to run on the available node and also configures the cluster to reschedule an instance on the new node when needed.
- To check our deployment, we use **kubectl get deployments** command which shows the instance is running by 1 deployment and the instance runs inside a Docker container of our node.
- Pods represent the processes running on a cluster and it runs inside a private, isolated network
- When a pod is created it is assigned its own unique IP address. If there are multiple containers within the pod, they can communicate between each other simply by using localhost. Communications outside of the pod is achieved by exposing a port. Communications between pods in a cluster takes advantage of the fact that Kubernetes assigns a cluster-private IP address to every pid in a cluster, eliminating the need to either explicitly create links between pods or to map container ports to host ports. In this way, every pod in a cluster can see each other without the need for NAT. Therefore, kubectl creates a proxy that forwards communications into the cluster wide private network and provides direct access to the API from these terminals. We open a separate Terminal to run the echo command to establish the communication between the terminal and Kubernetes Cluster.
- Curl command is version check directly through API. API server automatically create an endpoint for each pod, based on the pod name, that is also accessible through the proxy
- All the commands are below in the screenshot.

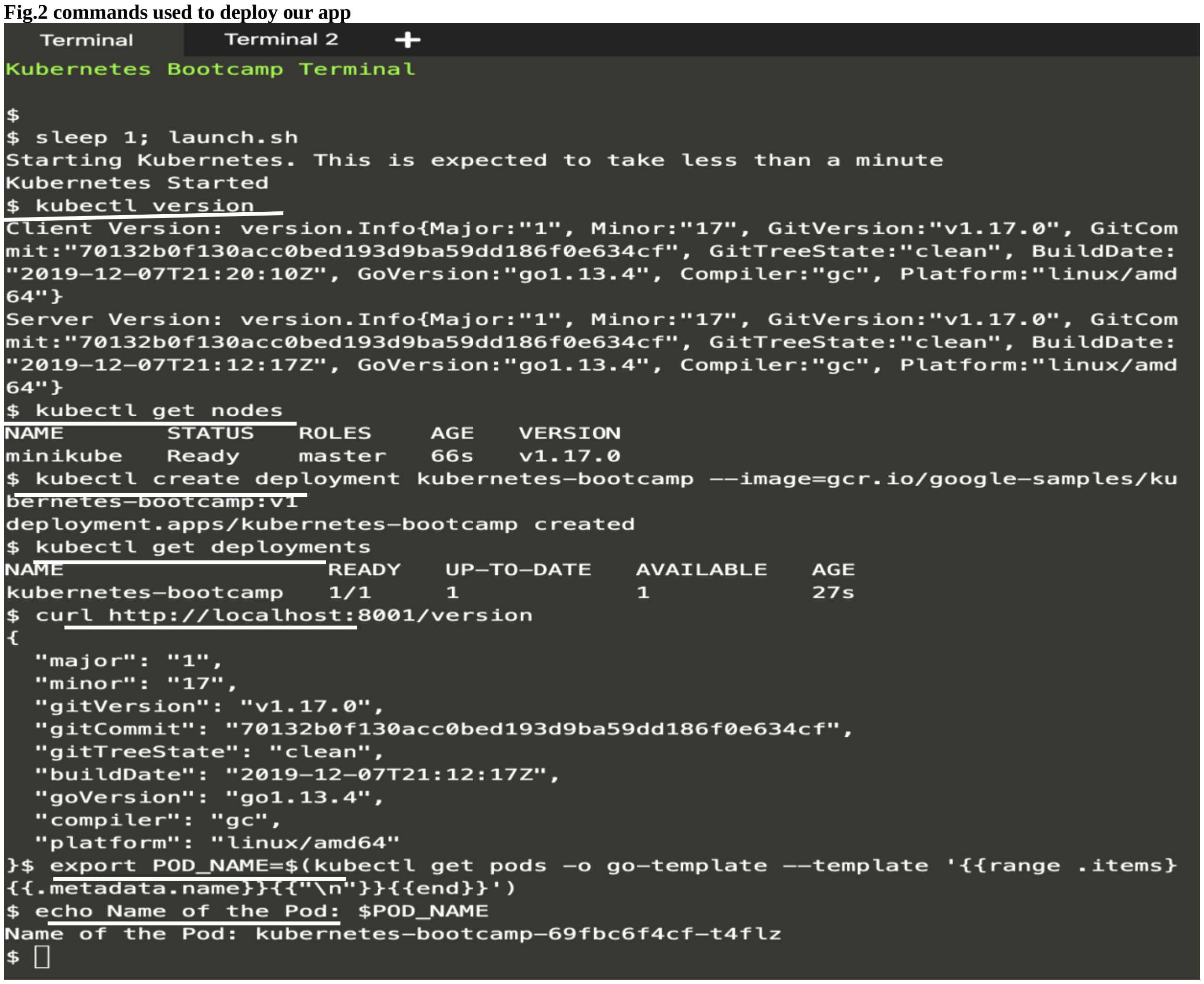
Note Fig1 is placed first because of the space

Fig.1 This image shows the Proxy Terminal



```
Extra Interactive Bash Terminal


echo -e "\n\n\n\e[92mStarting Proxy. After starting it will not output a respons
e. Please click the first Terminal Tab\n";
kubectl proxy
$
$ echo -e "\n\n\n\e[92mStarting Proxy. After starting it will not output a respo
nse. Please click the first Terminal Tab\n";



Starting Proxy. After starting it will not output a response. Please click the f
irst Terminal Tab

$ kubectl proxy
Starting to serve on 127.0.0.1:8001
```

Fig.2 commands used to deploy our app

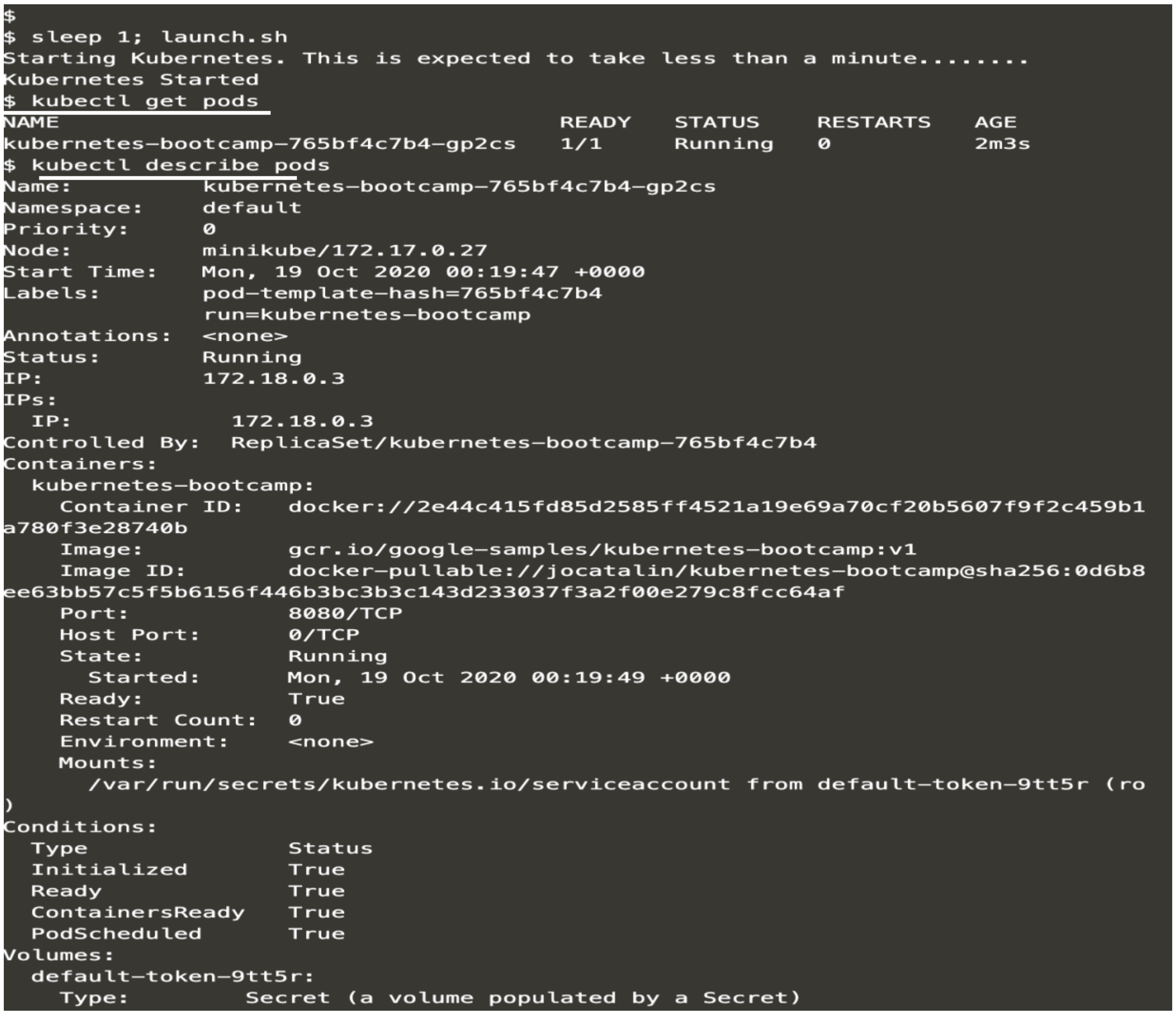


```
Kubernetes Bootcamp Terminal

$
$ 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:"v1.17.0", GitCom
mit:"70132b0f130acc0bed193d9ba59dd186f0e634cf", GitTreeState:"clean", BuildDate:
"2019-12-07T21:20:10Z", GoVersion:"go1.13.4", Compiler:"gc", Platform:"linux/amd
64"}
Server Version: version.Info{Major:"1", Minor:"17", GitVersion:"v1.17.0", GitCom
mit:"70132b0f130acc0bed193d9ba59dd186f0e634cf", GitTreeState:"clean", BuildDate:
"2019-12-07T21:12:17Z", GoVersion:"go1.13.4", Compiler:"gc", Platform:"linux/amd
64"}
$ kubectl get nodes
NAME       STATUS   ROLES    AGE   VERSION
minikube   Ready    master   66s   v1.17.0
$ kubectl create deployment kubernetes-bootcamp --image=gcr.io/google-samples/ku
bernetes-bootcamp:v1
deployment.apps/kubernetes-bootcamp created
$ kubectl get deployments
NAME                  READY   UP-TO-DATE   AVAILABLE   AGE
kubernetes-bootcamp   1/1     1            1           27s
$ curl http://localhost:8001/version
{
  "major": "1",
  "minor": "17",
  "gitVersion": "v1.17.0",
  "gitCommit": "70132b0f130acc0bed193d9ba59dd186f0e634cf",
  "gitTreeState": "clean",
  "buildDate": "2019-12-07T21:12:17Z",
  "goVersion": "go1.13.4",
  "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-t4flz
$ 
```

Module 3: Explore our App

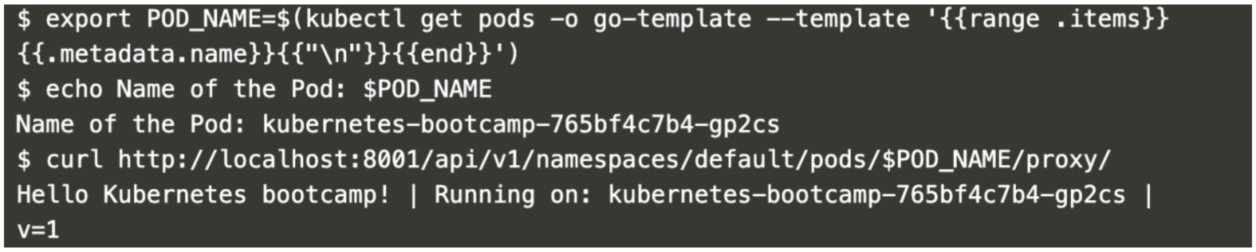
- Pods are the atomic unit on the Kubernetes platform. When we create a Deployment on Kubernetes, that Deployment creates Pods with containers inside them (as opposed to creating containers directly). Each Pod is tied to the Node where it is scheduled and remains there until termination (according to restart policy) or deletion. In case of a Node failure, identical Pods are scheduled on other available Nodes in the cluster.
- **Kubectl get pods** command is a way to verify our that our application we deployed is running. As you can see in the image, we get the name of our application image and the status as running.
- Now moving further to know about the containers inside the pod and to check the images used to build those containers we run the **kubectl describe pods** command. We see here details about the Pod’s container: IP address, the ports used, and a list of events related to the lifecycle of the Pod. Containers that belong to the Pod can communicate with one another using localhost

Akshay Bhala
abhala@syr.edu



```
$
$ 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-gp2cs  1/1     Running   0          2m3s
$ kubectl describe pods
Name:         kubernetes-bootcamp-765bf4c7b4-gp2cs
Namespace:    default
Priority:     0
Node:         minikube/172.17.0.27
Start Time:   Mon, 19 Oct 2020 00:19:47 +0000
Labels:       pod-template-hash=765bf4c7b4
              run=kubernetes-bootcamp
Annotations:  <none>
Status:       Running
IP:           172.18.0.3
IPs:
  IP:           172.18.0.3
Controlled By:  ReplicaSet/kubernetes-bootcamp-765bf4c7b4
Containers:
  kubernetes-bootcamp:
    Container ID:   docker://2e44c415fd85d2585ff4521a19e69a70cf20b5607f9f2c459b1
a780f3e28740b
    Image:          gcr.io/google-samples/kubernetes-bootcamp:v1
    Image ID:       docker-pullable://jocatalin/kubernetes-bootcamp@sha256:0d6b8
ee63bb57c5f5b6156f446b3bc3b3c143d233037f3a2f00e279c8fcc64af
    Port:           8080/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Mon, 19 Oct 2020 00:19:49 +0000
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-9tt5r (ro
)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  default-token-9tt5r:
    Type:        Secret (a volume populated by a Secret)
```

- Pods are running in an isolated, private network - so we need to proxy access to them so we can debug and interact with them. Follow the commands shown in the image below which will open a new terminal and we can directly query the pod through the proxy to get the pod name and store it in the POD_NAME environment variable.
- Curl request will show the output of our application and the url is the route to the API of the pod



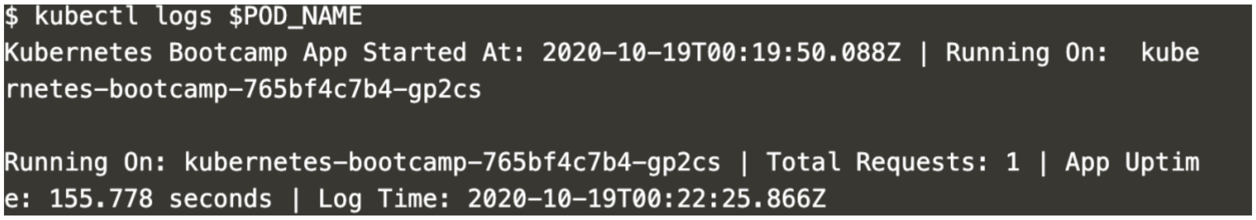
```
$ export POD_NAME=$(kubectl get pods -o go-template --template '{{range .items}}
{{.metadata.name}}{{"\n"}}{{end}}')
$ echo Name of the Pod: $POD_NAME
Name of the Pod: kubernetes-bootcamp-765bf4c7b4-gp2cs
$ curl http://localhost:8001/api/v1/namespaces/default/pods/$POD_NAME/proxy/
Hello Kubernetes bootcamp! | Running on: kubernetes-bootcamp-765bf4c7b4-gp2cs |
v=1
```

- Application logs can help you understand what is happening inside your application, debugging and to monitor cluster activity

Akshay Bhala
abhala@syr.edu

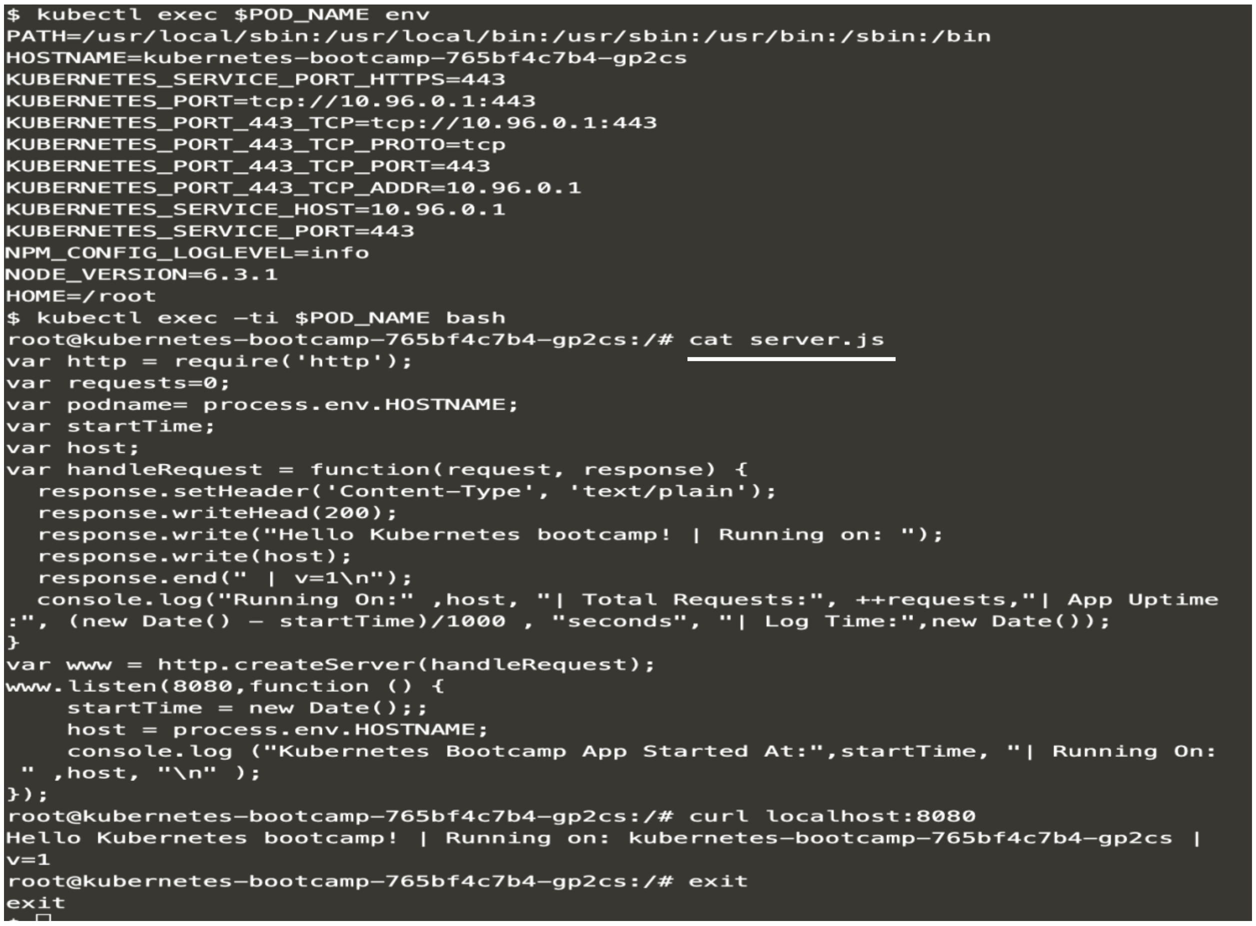
IST 615

- To view the container logs, we need to keep an eye out for anything that our applications sends to **STDOUT(Standard output).** To retrieve them, we use the **kubectl logs command**. Since we only have one container, we did not need to specify the container name.



```
$ kubectl logs $POD_NAME
Kubernetes Bootcamp App Started At: 2020-10-19T00:19:50.088Z | Running On:  kube
rnetes-bootcamp-765bf4c7b4-gp2cs

Running On: kubernetes-bootcamp-765bf4c7b4-gp2cs | Total Requests: 1 | App Uptim
e: 155.778 seconds | Log Time: 2020-10-19T00:22:25.866Z
```

- As our pod is running, we can now directly execute the command on our container. As we just have one container, we don't need to mention the container name and kubectl exec $Pod_Name env will list all the environment variables. These Environment variables are used to inject Data Into Applications or to expose pod information to containers.
- Kubectl exec -ti $POD_NAME bash command is to get a Shell in a Container, use the -t -i options to get a tty and attach STDIN
- We now have a console running on our container and hence we run our nodeJS application. Cat server.js includes the source code of our application. Using curl command to verify our application is up and running.
- We used localhost because we executed the command inside the NodeJS Pod

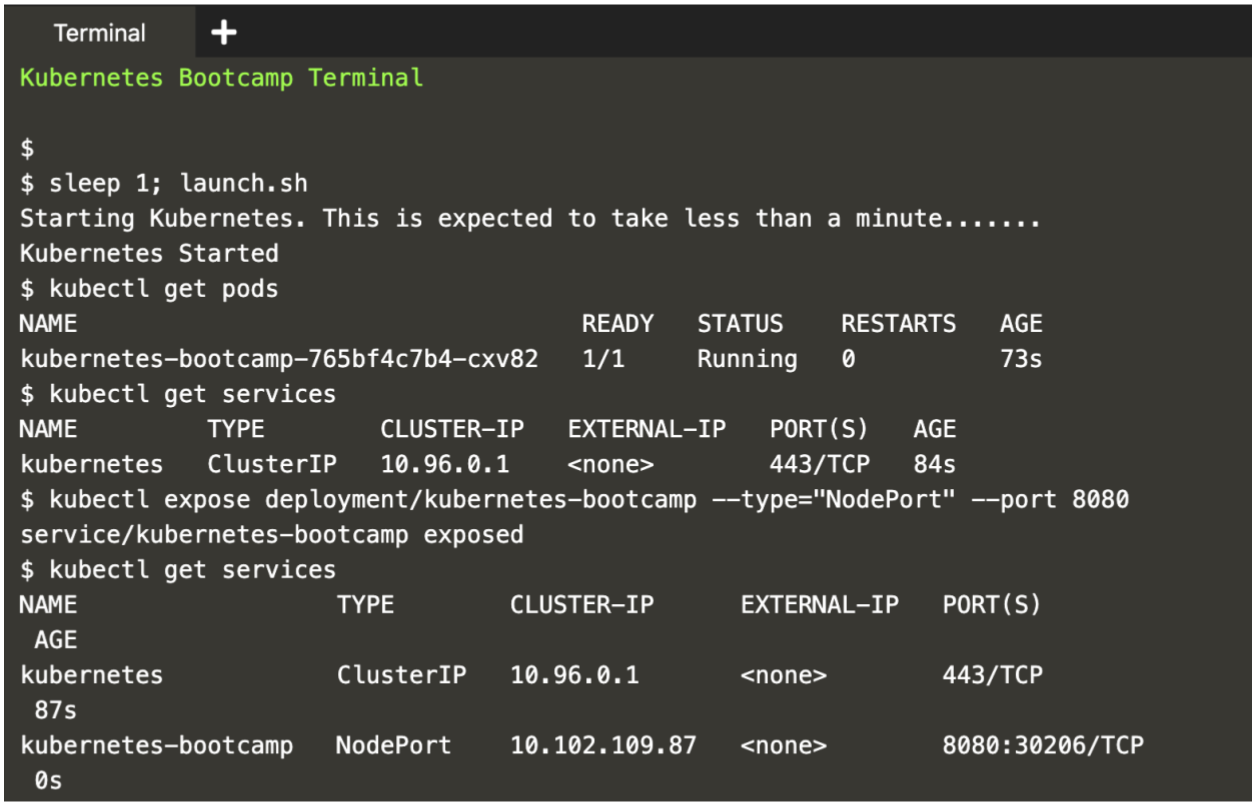


```
$ kubectl exec $POD_NAME env
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOSTNAME=kubernetes-bootcamp-765bf4c7b4-gp2cs
KUBERNETES_SERVICE_PORT_HTTPS=443
KUBERNETES_PORT=tcp://10.96.0.1:443
KUBERNETES_PORT_443_TCP=tcp://10.96.0.1:443
KUBERNETES_PORT_443_TCP_PROTO=tcp
KUBERNETES_PORT_443_TCP_PORT=443
KUBERNETES_PORT_443_TCP_ADDR=10.96.0.1
KUBERNETES_SERVICE_HOST=10.96.0.1
KUBERNETES_SERVICE_PORT=443
NPM_CONFIG_LOGLEVEL=info
NODE_VERSION=6.3.1
HOME=/root
$ kubectl exec -ti $POD_NAME bash
root@kubernetes-bootcamp-765bf4c7b4-gp2cs:/# 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" );
});
root@kubernetes-bootcamp-765bf4c7b4-gp2cs:/# curl localhost:8080
Hello Kubernetes bootcamp! | Running on: kubernetes-bootcamp-765bf4c7b4-gp2cs |
v=1
root@kubernetes-bootcamp-765bf4c7b4-gp2cs:/# exit
exit
```

Akshay Bhala
abhala@syr.edu

Module 4: Expose your App Publicly

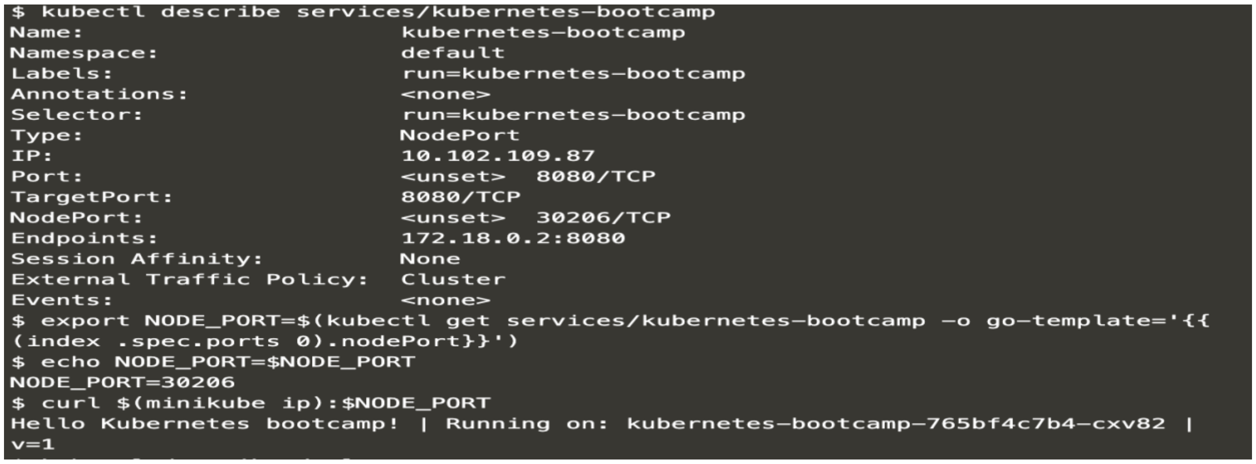
- A Service in Kubernetes is an abstraction which defines a logical set of Pods and a policy by which to access them.
- To list the current services our cluster is using, we use the kubectl get service command. Service called kubernetes that is created by default when minikube starts the cluster.
- We create a new service and expose it to external traffic by using the expose command with NodePort as parameter. We have now a running Service called kubernetes-bootcamp.
- If you set the type field to NodePort, the Kubernetes control plane allocates a port from a range specified by --service-node-port-range. Each node proxies that port (the same port number on every Node) into your Service. Your Service reports the allocated port in its .spec.ports[*].nodePort field.
- 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)



```
Kubernetes Bootcamp Terminal

$
$ 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-cxv82   1/1     Running   0          73s
$ kubectl get services
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   84s
$ 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
 87s
kubernetes-bootcamp   NodePort    10.102.109.87   <none>        8080:30206/TCP
 0s
```

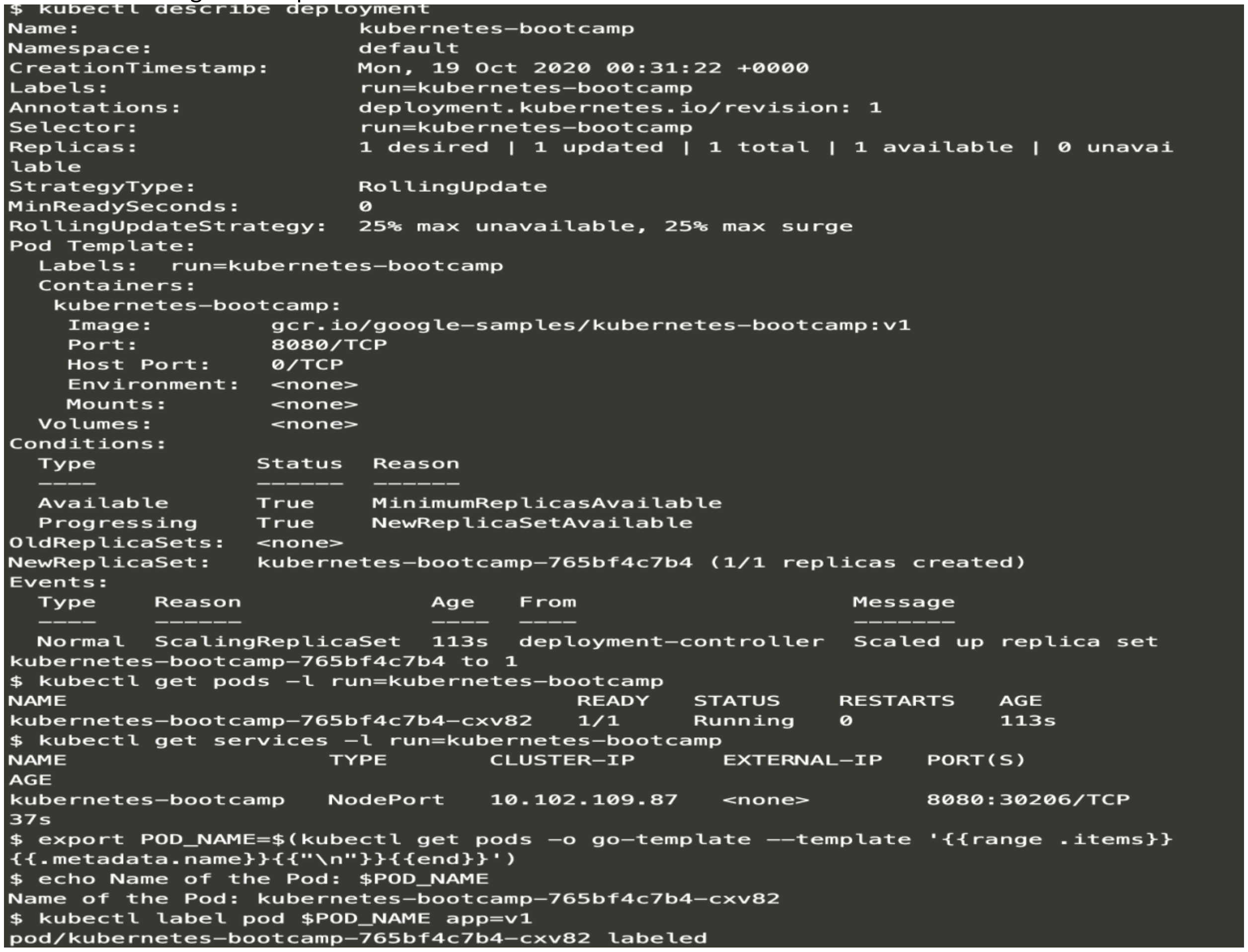
- To find out what port was opened externally (by the NodePort option) we'll run the describe service command
- Once that is done, we create a new environment variable called NODE_PORT which also has the value to the port assigned.
- We then test our application using the curl command. (IP of the node and externally exposed port) and as shown in the image below we get a response from the server. The Service is exposed

Akshay Bhala
abhala@syr.edu



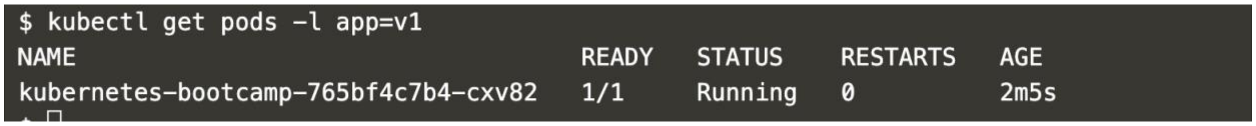
```
$ kubectl describe services/kubernetes-bootcamp
Name:                     kubernetes-bootcamp
Namespace:                default
Labels:                   run=kubernetes-bootcamp
Annotations:              <none>
Selector:                 run=kubernetes-bootcamp
Type:                     NodePort
IP:                       10.102.109.87
Port:                     <unset>  8080/TCP
TargetPort:               8080/TCP
NodePort:                 <unset>  30206/TCP
Endpoints:                172.18.0.2:8080
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=30206
$ curl $(minikube ip):$NODE_PORT
Hello Kubernetes bootcamp! | Running on: kubernetes-bootcamp-765bf4c7b4-cxv82 |
v=1
```

- The deployment automatically created a label for our pod. Labels are key/value pairs that are attached to Kubernetes objects, such as pods. Use the **describe deployment** command to know the label of our pod.
- We used the label name to get the pods and the services linked to our service “Kubernetes-bootcamp”.
- We then created an environment variable to store the name of the Pod (POD_NAME). We then apply a new label by using the **label** command followed by object type, object name (we get this as we created an environment variable) and the new label. We also pinned the application version to the POD.
- Using describe command we see that the label is attached to the pod and we can now query the pod using new label to get list of pods.



```
$ kubectl describe deployment
Name:                   kubernetes-bootcamp
Namespace:              default
CreationTimestamp:      Mon, 19 Oct 2020 00:31:22 +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 unavai
lable
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
  Progressing    True    NewReplicaSetAvailable
OldReplicaSets:  <none>
NewReplicaSet:   kubernetes-bootcamp-765bf4c7b4 (1/1 replicas created)
Events:
  Type    Reason             Age   From                   Message
  ----    ------             ----  ----                   -------
  Normal  ScalingReplicaSet  113s  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-cxv82   1/1     Running   0          113s
$ kubectl get services -l run=kubernetes-bootcamp
NAME                  TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)
AGE
kubernetes-bootcamp   NodePort   10.102.109.87   <none>        8080:30206/TCP
37s
$ 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-cxv82
$ kubectl label pod $POD_NAME app=v1
pod/kubernetes-bootcamp-765bf4c7b4-cxv82 labeled
```

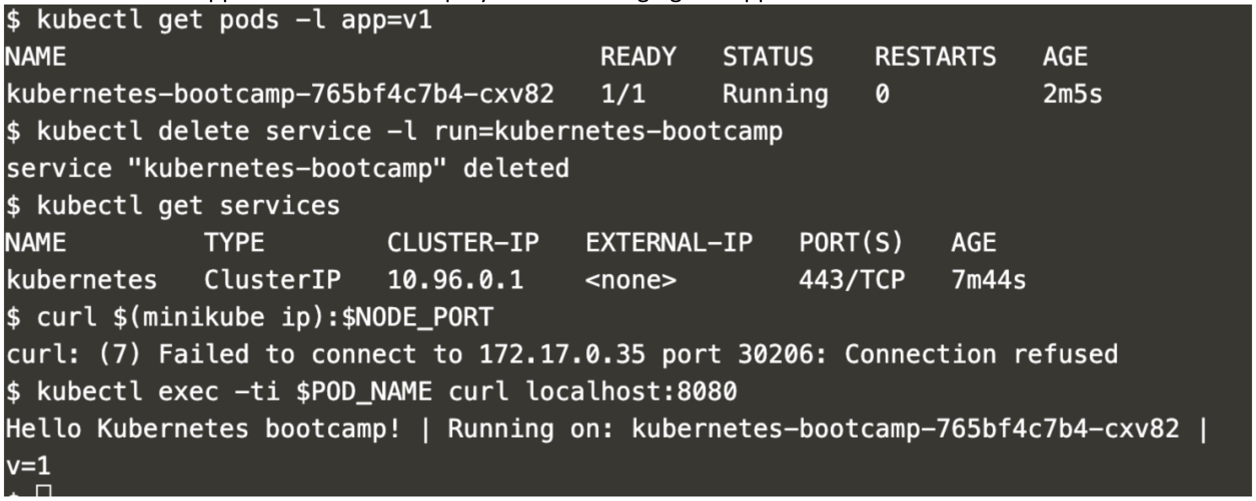
- We can see the Pod here



```
$ kubectl get pods -l app=v1
NAME                                  READY   STATUS    RESTARTS   AGE
kubernetes-bootcamp-765bf4c7b4-cxv82   1/1     Running   0          2m5s
```

Deleting a service

- Use the **delete service** for deleting the service.
- **get services** command verify whether the service is deleted or not. As seen in the image below, it gets confirmed that the service is deleted.
- We use the curl command to check whether the route is exposed or not. As seen below, the error confirms that the app is no longer reachable from outside the cluster.
- We can still check whether the app is reachable from inside the cluster by using the command as shown below. This happens because the deployment is managing the application



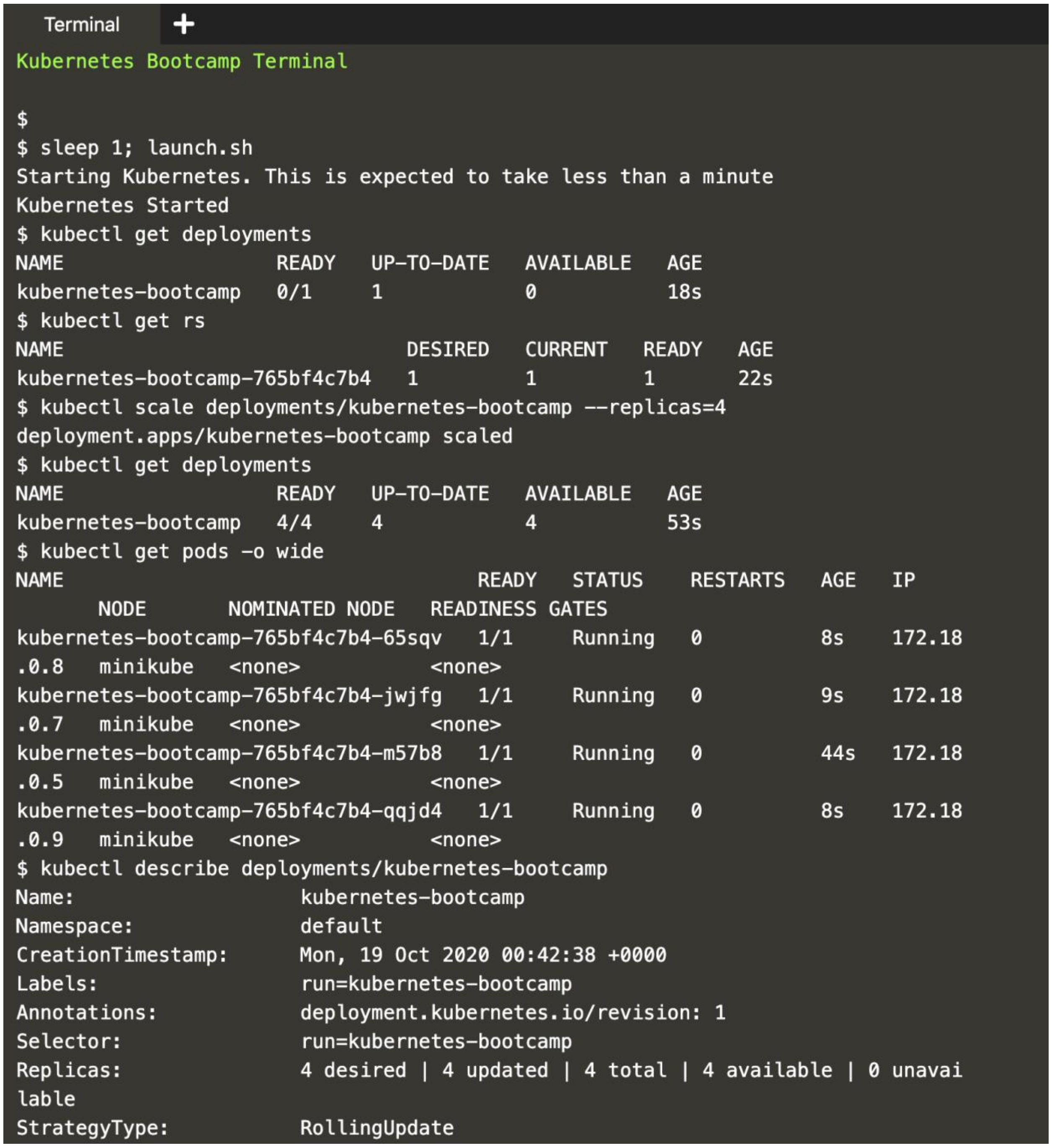
```
$ kubectl get pods -l app=v1
NAME                                  READY   STATUS    RESTARTS   AGE
kubernetes-bootcamp-765bf4c7b4-cxv82   1/1     Running   0          2m5s
$ kubectl delete service -l run=kubernetes-bootcamp
service "kubernetes-bootcamp" deleted
$ kubectl get services
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   7m44s
$ curl $(minikube ip):$NODE_PORT
curl: (7) Failed to connect to 172.17.0.35 port 30206: Connection refused
$ kubectl exec -ti $POD_NAME curl localhost:8080
Hello Kubernetes bootcamp! | Running on: kubernetes-bootcamp-765bf4c7b4-cxv82 |
v=1
```

Module 5: Scale Your App

- In previous modules we created a Deployment, and then exposed it publicly via a Service. The Deployment created only one Pod for running our application. When traffic increases, we will need to scale the application to keep up with user demand. **Scaling** is accomplished by changing the number of replicas in a Deployment.
- Scaling out a Deployment will ensure new Pods are created and scheduled to Nodes with available resources. Scaling will increase the number of Pods to the new desired state.
- Scaling to zero is also possible, and it will terminate all Pods of the specified Deployment.
- **get deployments** command gives us NAME: lists the names of the 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 available to your users. AGE: displays the amount of time that the application has been running.
- To see the ReplicaSet created by the Deployment, run **kubectl get rs**
- Notice that the name of the ReplicaSet is always formatted as [DEPLOYMENT-NAME]-[RANDOM-STRING]. The random string is randomly generated and uses the pod-template-hash as a seed
- Two important columns of this command are: DESIRED displays the desired number of replicas of the application, which you define when you create the Deployment. This is the desired state. CURRENT displays how many replicas are currently running

Akshay Bhala
abhala@syr.edu

- Using **kubectl scale** command we are scaling our deployment to 4 replicas. We now check our deployments and we can rightly see that we have 4 instances of the application available
- kubectl get pods -o wide   # List all pods in the current namespace, with more details. There are 4 Pods now, with different IP addresses. The change was registered in the Deployment events log
- We use Describe command to check 4 replicas are avialabe or not.

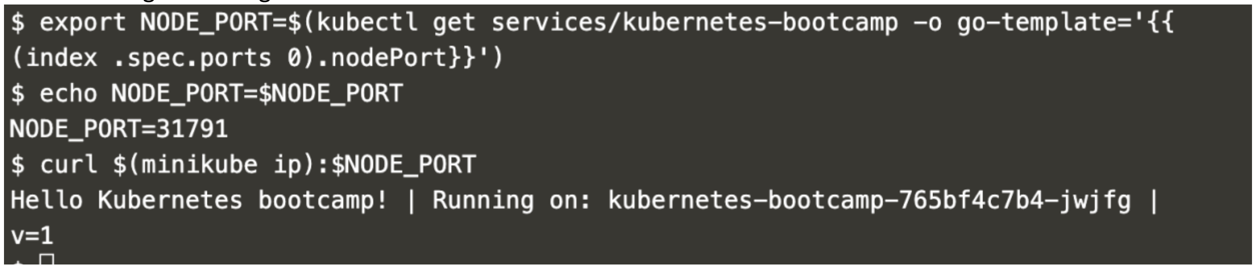


```
Kubernetes Bootcamp Terminal

$
$ 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   0/1     1            0           18s
$ kubectl get rs
NAME                             DESIRED   CURRENT   READY   AGE
kubernetes-bootcamp-765bf4c7b4   1         1         1       22s
$ 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           53s
$ kubectl get pods -o wide
NAME                                   READY   STATUS    RESTARTS   AGE   IP
       NODE       NOMINATED NODE   READINESS GATES
kubernetes-bootcamp-765bf4c7b4-65sqv   1/1     Running   0          8s    172.18
.0.8   minikube   <none>           <none>
kubernetes-bootcamp-765bf4c7b4-jwjfg   1/1     Running   0          9s    172.18
.0.7   minikube   <none>           <none>
kubernetes-bootcamp-765bf4c7b4-m57b8   1/1     Running   0          44s   172.18
.0.5   minikube   <none>           <none>
kubernetes-bootcamp-765bf4c7b4-qqjd4   1/1     Running   0          8s    172.18
.0.9   minikube   <none>           <none>
$ kubectl describe deployments/kubernetes-bootcamp
Name:                   kubernetes-bootcamp
Namespace:              default
CreationTimestamp:      Mon, 19 Oct 2020 00:42:38 +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 unavai
lable
StrategyType:           RollingUpdate
```

Akshay Bhala
abhala@syr.edu

Load Balancing:

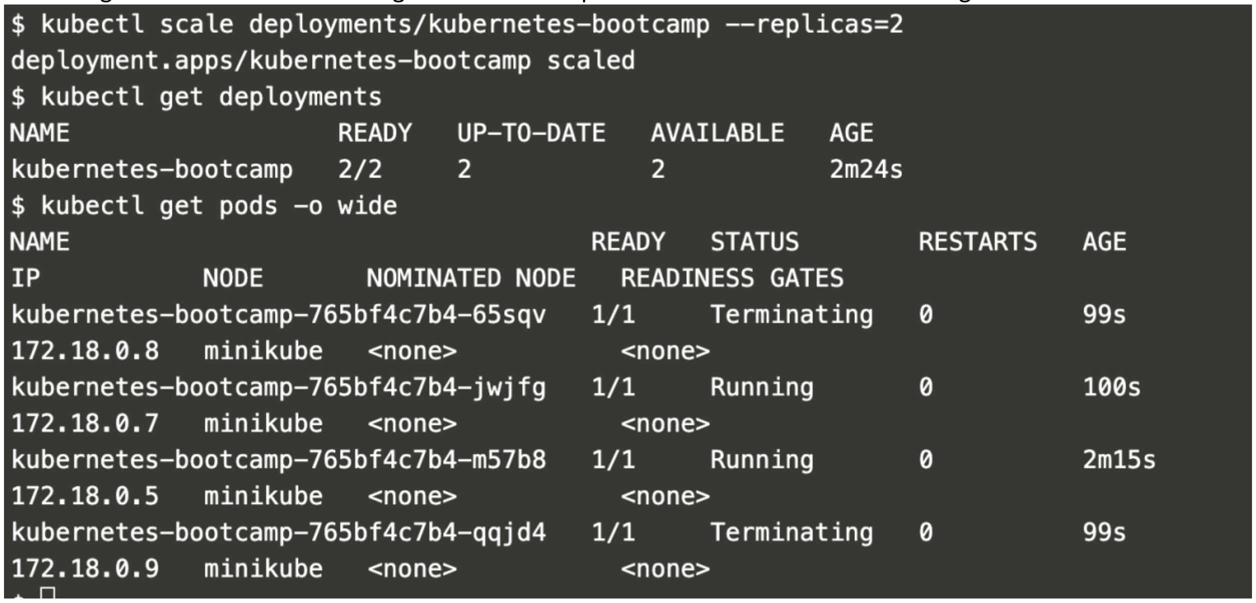
- Running multiple instances of an application will require a way to distribute the traffic to all of them. Services have an integrated load-balancer that will distribute network traffic to all Pods of an exposed Deployment. Services will monitor continuously the running Pods using endpoints, to ensure the traffic is sent only to available Pods.
- We create an environment variable and use the curl command to the exposed IP and port. Execute the command multiple times: We hit a different Pod with every request. This demonstrates that the load-balancing is working.



```
$ export NODE_PORT=$(kubectl get services/kubernetes-bootcamp -o go-template='{{
(index .spec.ports 0).nodePort}}')
$ echo NODE_PORT=$NODE_PORT
NODE_PORT=31791
$ curl $(minikube ip):$NODE_PORT
Hello Kubernetes bootcamp! | Running on: kubernetes-bootcamp-765bf4c7b4-jwjfg |
v=1
```

Scale Down:

- To scale down we follow the same commands as above. We can also slay down to 2 replicas from 4 using the scale command. To ensure that, we can use the get deployment command to check if the change was applied. Using the o wide command we get details of the pods. We can see that 2 are running and 2 are terminated



```
$ 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           2m24s
$ kubectl get pods -o wide
NAME                                   READY   STATUS        RESTARTS   AGE
IP           NODE       NOMINATED NODE   READINESS GATES
kubernetes-bootcamp-765bf4c7b4-65sqv   1/1     Terminating   0          99s
172.18.0.8   minikube   <none>           <none>
kubernetes-bootcamp-765bf4c7b4-jwjfg   1/1     Running       0          100s
172.18.0.7   minikube   <none>           <none>
kubernetes-bootcamp-765bf4c7b4-m57b8   1/1     Running       0          2m15s
172.18.0.5   minikube   <none>           <none>
kubernetes-bootcamp-765bf4c7b4-qqjd4   1/1     Terminating   0          99s
172.18.0.9   minikube   <none>           <none>
```

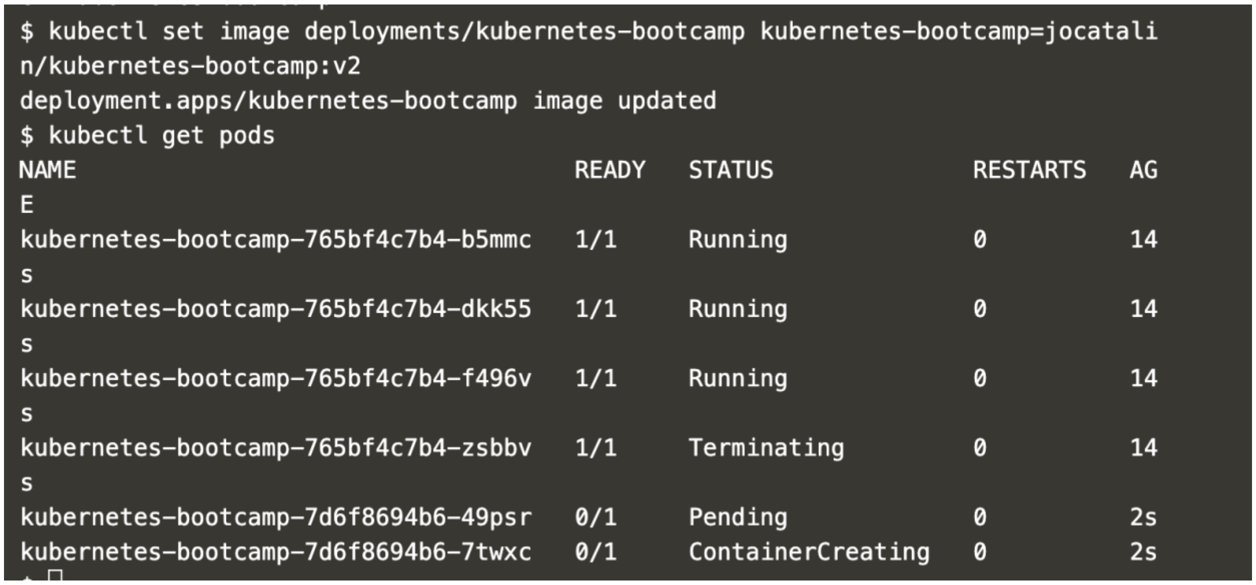
Module 6: Update Your App

- Rolling updates allow Deployments' update to take place with zero downtime by incrementally updating Pods instances with new ones. The new Pods will be scheduled on Nodes with available resources.
- Scaling is a requirement for performing updates without affecting application availability. By default, the maximum number of Pods that can be unavailable during the update and the maximum number of new Pods that can be created, is one. Both options can be configured to either numbers or percentages (of Pods). In Kubernetes, updates are versioned, and any Deployment update can be reverted to a previous (stable) version.
- Rolling updates allow the following actions:
    - Promote an application from one environment to another (via container image updates)
    - Rollback to previous versions

- Continuous Integration and Continuous Delivery of applications with zero downtime

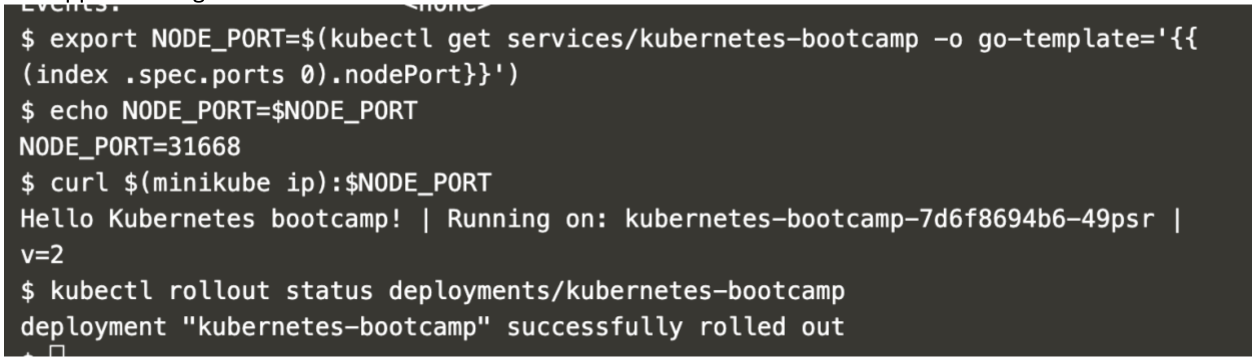
UPDATE YOUR APP:

- 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. The command notified the Deployment to use a different image for your app and initiated a rolling update
- Verify the change by **get pods** command and we see that the new one is running and old one is terminated



```
$ kubectl set image deployments/kubernetes-bootcamp kubernetes-bootcamp=jocatali
n/kubernetes-bootcamp:v2
deployment.apps/kubernetes-bootcamp image updated
$ kubectl get pods
NAME                                   READY   STATUS              RESTARTS   AG
E
kubernetes-bootcamp-765bf4c7b4-b5mmc   1/1     Running             0          14
s
kubernetes-bootcamp-765bf4c7b4-dkk55   1/1     Running             0          14
s
kubernetes-bootcamp-765bf4c7b4-f496v   1/1     Running             0          14
s
kubernetes-bootcamp-765bf4c7b4-zsbbv   1/1     Terminating         0          14
s
kubernetes-bootcamp-7d6f8694b6-49psr   0/1     Pending             0          2s
kubernetes-bootcamp-7d6f8694b6-7twxc   0/1     ContainerCreating   0          2s
```

- Follow the same steps as done in previous modules by creating a new environment variable and using curl command to verify an update.
- As we have scaled with 2 replicas, we hit a different Pod with every request and we see that all Pods are running the latest version (v2).
- The update can also be confirmed also by running a rollout status command and to get the current image version of the app run a describe command against the pods. As shown in the image our version 2 of the app is running

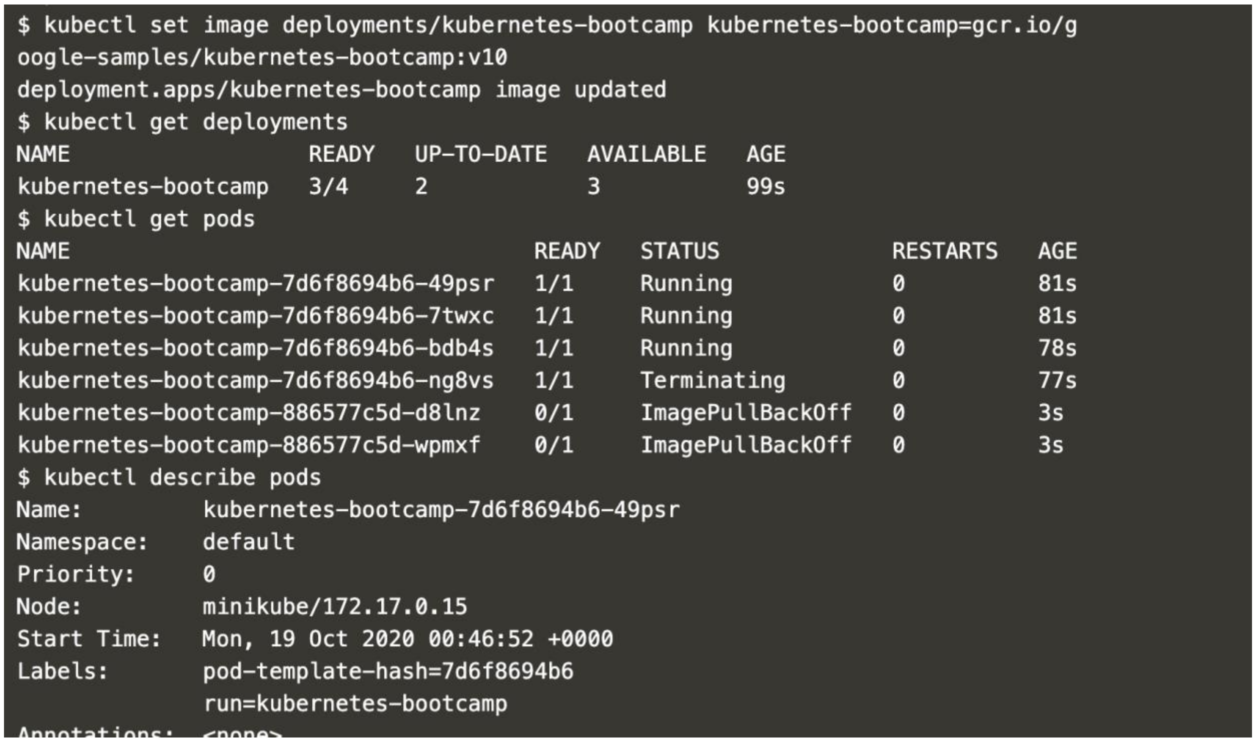


```
$ export NODE_PORT=$(kubectl get services/kubernetes-bootcamp -o go-template='{{
(index .spec.ports 0).nodePort}}')
$ echo NODE_PORT=$NODE_PORT
NODE_PORT=31668
$ curl $(minikube ip):$NODE_PORT
Hello Kubernetes bootcamp! | Running on: kubernetes-bootcamp-7d6f8694b6-49psr |
v=2
$ kubectl rollout status deployments/kubernetes-bootcamp
deployment "kubernetes-bootcamp" successfully rolled out
```

ROLLBACK AN UPDATE:

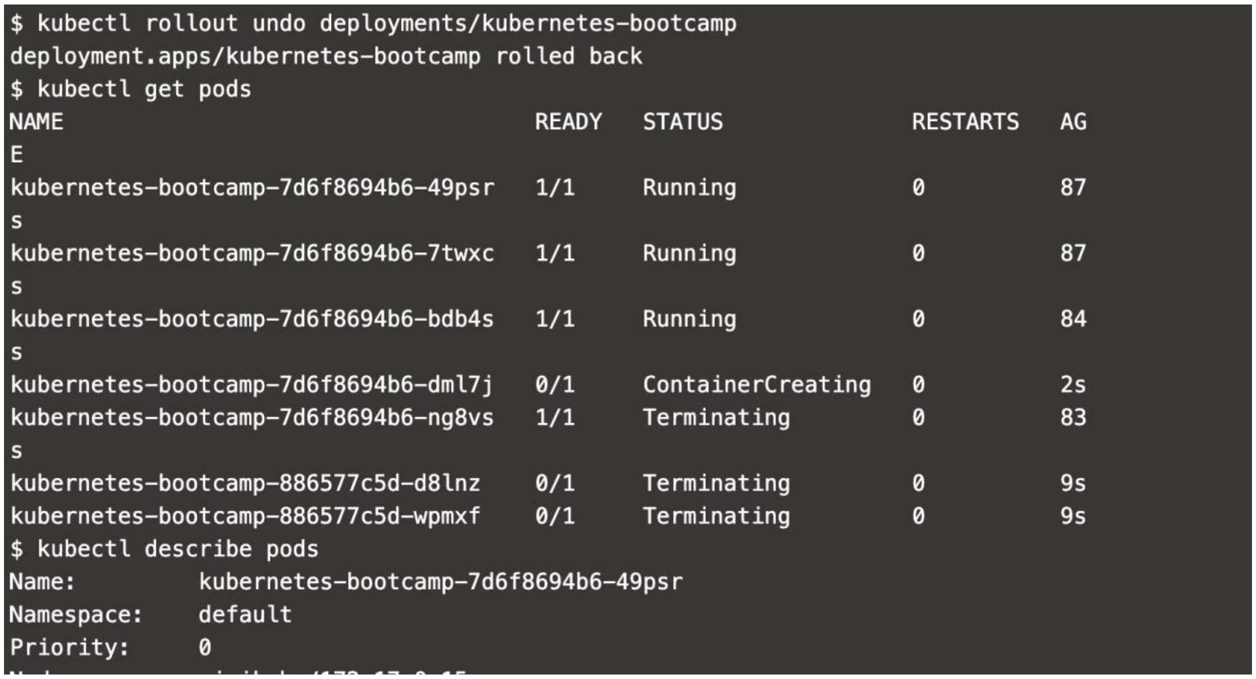
- In this we are trying to undo an update or how to retrieve back to original/ previous version if an update fails.
- Following the same commands as above we are trying to perform another update and deploy the image tagged as V10 using set image command.
- As we try to verify our updates we observe that something went wrong, we do not have the desired number of Pods available (1 out of 4 is not available) and also there is no image v10 in our repository.

Akshay Bhala
abhala@syr.edu



```
$ kubectl set image deployments/kubernetes-bootcamp kubernetes-bootcamp=gcr.io/g
oogle-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           99s
$ kubectl get pods
NAME                                   READY   STATUS             RESTARTS   AGE
kubernetes-bootcamp-7d6f8694b6-49psr   1/1     Running            0          81s
kubernetes-bootcamp-7d6f8694b6-7twxc   1/1     Running            0          81s
kubernetes-bootcamp-7d6f8694b6-bdb4s   1/1     Running            0          78s
kubernetes-bootcamp-7d6f8694b6-ng8vs   1/1     Terminating        0          77s
kubernetes-bootcamp-886577c5d-d8lnz    0/1     ImagePullBackOff   0          3s
kubernetes-bootcamp-886577c5d-wpmxf    0/1     ImagePullBackOff   0          3s
$ kubectl describe pods
Name:         kubernetes-bootcamp-7d6f8694b6-49psr
Namespace:    default
Priority:     0
Node:         minikube/172.17.0.15
Start Time:   Mon, 19 Oct 2020 00:46:52 +0000
Labels:       pod-template-hash=7d6f8694b6
              run=kubernetes-bootcamp
Annotations:  <none>
```

- We then rollback to our previous version**. Rollout undo** command reverted the deployment to the previous known state (v2 of the image). Updates are versioned and you can revert to any previously know state of a Deployment.
- After listing the pods, we see that our 4 pods are running now, and 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   AG
E
kubernetes-bootcamp-7d6f8694b6-49psr   1/1     Running             0          87
s
kubernetes-bootcamp-7d6f8694b6-7twxc   1/1     Running             0          87
s
kubernetes-bootcamp-7d6f8694b6-bdb4s   1/1     Running             0          84
s
kubernetes-bootcamp-7d6f8694b6-dml7j   0/1     ContainerCreating   0          2s
kubernetes-bootcamp-7d6f8694b6-ng8vs   1/1     Terminating         0          83
s
kubernetes-bootcamp-886577c5d-d8lnz    0/1     Terminating         0          9s
kubernetes-bootcamp-886577c5d-wpmxf    0/1     Terminating         0          9s
$ kubectl describe pods
Name:         kubernetes-bootcamp-7d6f8694b6-49psr
Namespace:    default
Priority:     0
```