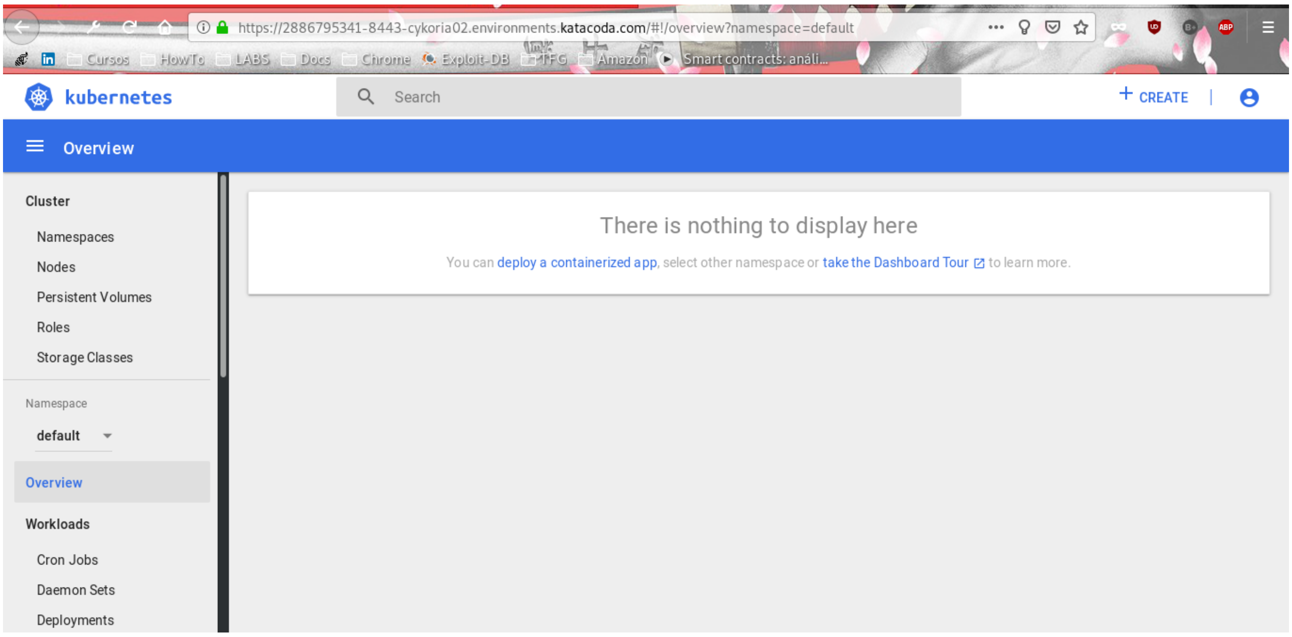
Práctica 7
CPD

Por: Rubén Calvo Villazán

II.a) Captura del Dashboard.
II.b) Captura donde aparece el puerto modificado.
II.c) Captura donde se muestran los Deployment y Services.
II.d) Captura donde aparece personalizado el NodePort.
II.e) Captura donde personalizamos el enrutamiento de nivel 4.
II.f) Captura donde personalizamos el mensaje de salida en el almacenamiento persistente.
II.g) Captura de la ejecución de la automatización con forge

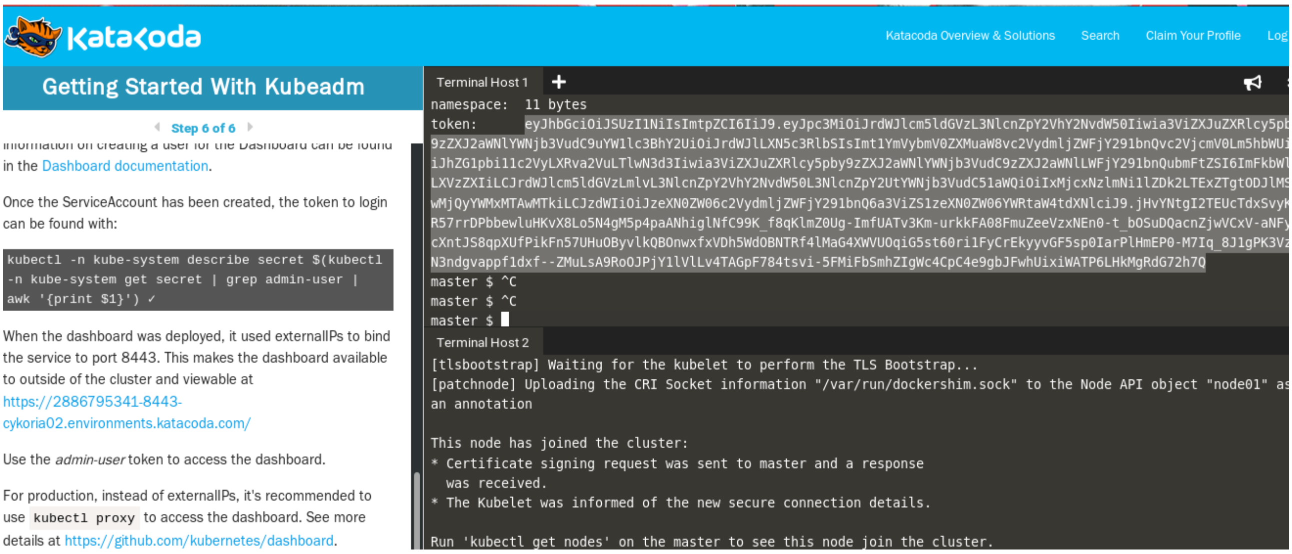
Launch a multi-node cluster using Kubeadm

II.a) Captura del Dashboard.



There is nothing to display here

You can deploy a containerized app, select other namespace or take the Dashboard Tour to learn more.



information on creating a user for the Dashboard can be found in the Dashboard documentation.

Once the ServiceAccount has been created, the token to login can be found with:

```
kubectl -n kube-system describe secret $(kubectl
-n kube-system get secret | grep admin-user |
awk '{print $1}') ✓
```

When the dashboard was deployed, it used externalIPs to bind the service to port 8443. This makes the dashboard available to outside of the cluster and viewable at https://2886795341-8443-cykoria02.environments.katacoda.com/

Use the *admin-user* token to access the dashboard.

For production, instead of externalIPs, it's recommended to use `kubectl proxy` to access the dashboard. See more details at https://github.com/kubernetes/dashboard.

```
namespace:  11 bytes
token:      eyJhbGciOiJSUzI1NiIsImtpZCI6IiJ9.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50Iiwia3ViZXJuZXRlcy5pb
9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJrdWJlLXN5c3RlbSIsImt1YmVybmV0ZXMuaW8vc2VydmljZWFjY291bnQvc2VjcmV0Lm5hbWUi
iJhZG1pbi11c2VyLXRva2VuLTlwN3d3Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9zZXJ2aWNlLWFjY291bnQubmFtZSI6ImFkbWl
LXVzZXIiLCJrdWJlcm5ldGVzLmlvL3NlcnZpY2VhY2NvdW50L3NlcnZpY2UtYWNjb3VudC51aWQiOiIxMjcxNzlmNi1lZDk2LTExZTgtODJlMS
wMjQyYWMxMTAwMTkiLCJzdWIiOiJzeXN0ZW06c2VydmljZWFjY291bnQ6a3ViZS1zeXN0ZW06YWRtaW4tdXNlciJ9.jHvYNtgI2TEUcTdxSvyK
R57rrDPbbewluHKvX8Lo5N4gM5p4paANhiglNfC99K_f8qKlmZ0Ug-ImfUATv3Km-urkkFA08FmuZeeVzxNEn0-t_bOSuDQacnZjwVCxV-aNFy
cXntJS8qpXUfPikFn57UHuOByvlkQBOnwxfxVDh5WdOBNTRf4lMaG4XWVUOqiG5st60ri1FyCrEkyyvGF5sp0IarPlHmEP0-M7Iq_8J1gPK3Vz
N3ndgvappf1dxf--ZMuLsA9RoOJPjY1lVlLv4TAGpF784tsvi-5FMiFbSmhZIgWc4CpC4e9gbJFwhUixiWATP6LHkMgRdG72h7Q
master $ ^C
master $ ^C
master $
```

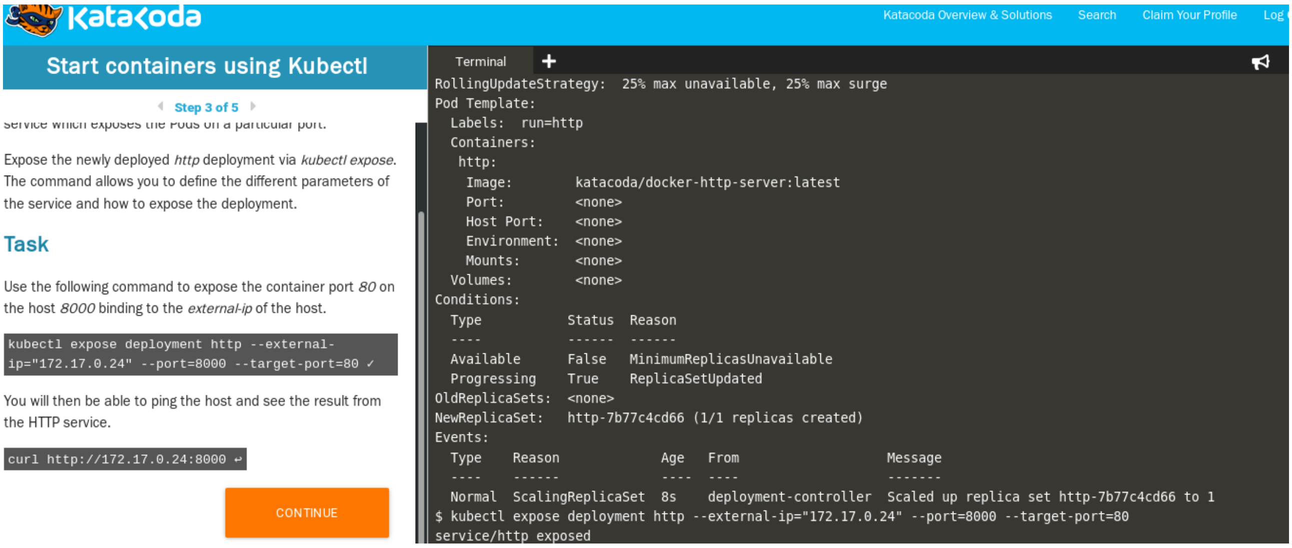
```
[tlsbootstrap] Waiting for the kubelet to perform the TLS Bootstrap...
[patchnode] Uploading the CRI Socket information "/var/run/dockershim.sock" to the Node API object "node01" as
an annotation

This node has joined the cluster:
* Certificate signing request was sent to master and a response
  was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the master to see this node join the cluster.
```

Deploy Containers Using Kubectl

II.b) Captura donde aparece el puerto modificado.



service which exposes the Pods on a particular port.

Expose the newly deployed *http* deployment via *kubectl expose*. The command allows you to define the different parameters of the service and how to expose the deployment.

Task

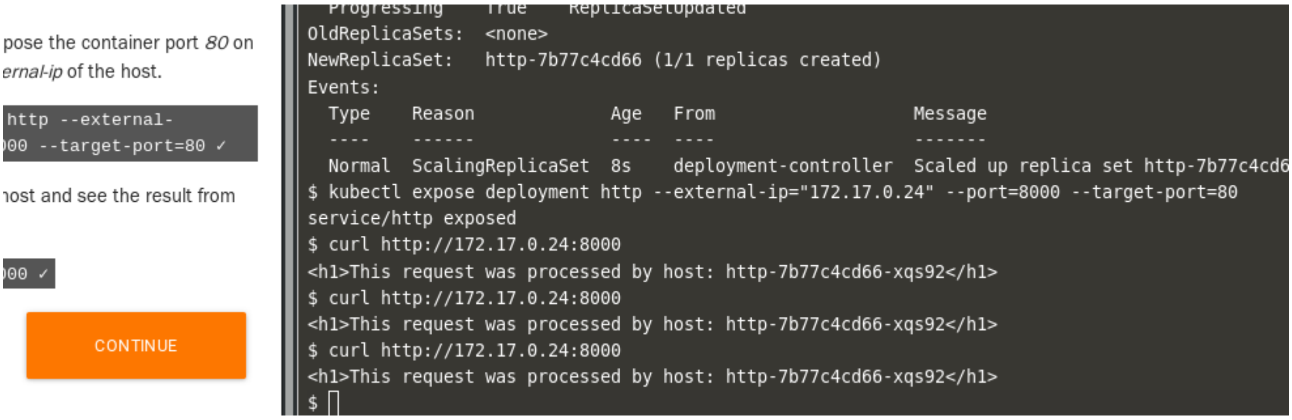
Use the following command to expose the container port *80* on the host *8000* binding to the *external-ip* of the host.

```
kubectl expose deployment http --external-ip="172.17.0.24" --port=8000 --target-port=80 ✓
```

You will then be able to ping the host and see the result from the HTTP service.

```
curl http://172.17.0.24:8000 ↩
```

```
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  run=http
  Containers:
   http:
    Image:        katacoda/docker-http-server:latest
    Port:         <none>
    Host Port:    <none>
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      False   MinimumReplicasUnavailable
  Progressing    True    ReplicaSetUpdated
OldReplicaSets:  <none>
NewReplicaSet:   http-7b77c4cd66 (1/1 replicas created)
Events:
  Type    Reason             Age   From                   Message
  ----    ------             ----  ----                   -------
  Normal  ScalingReplicaSet  8s    deployment-controller  Scaled up replica set http-7b77c4cd66 to 1
$ kubectl expose deployment http --external-ip="172.17.0.24" --port=8000 --target-port=80
service/http exposed
```



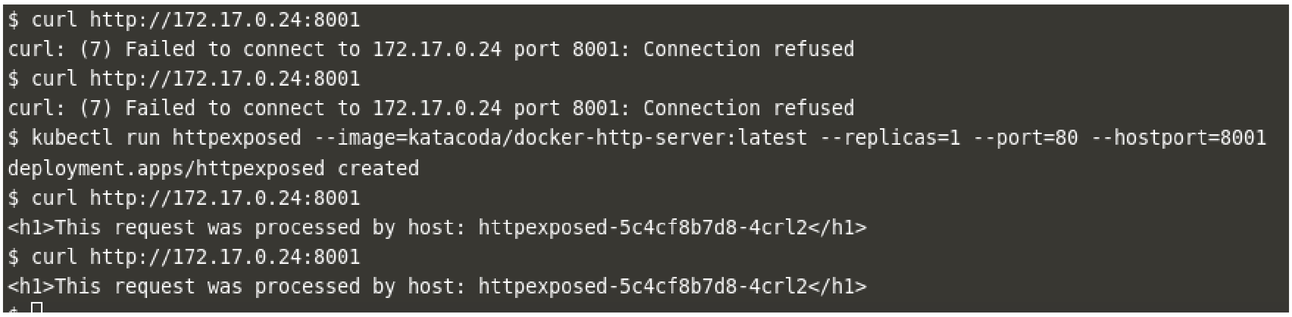
pose the container port *80* on
ernal-ip of the host.

```
http --external-
)00 --target-port=80 ✓
```

ıost and see the result from

```
)00 ✓
```

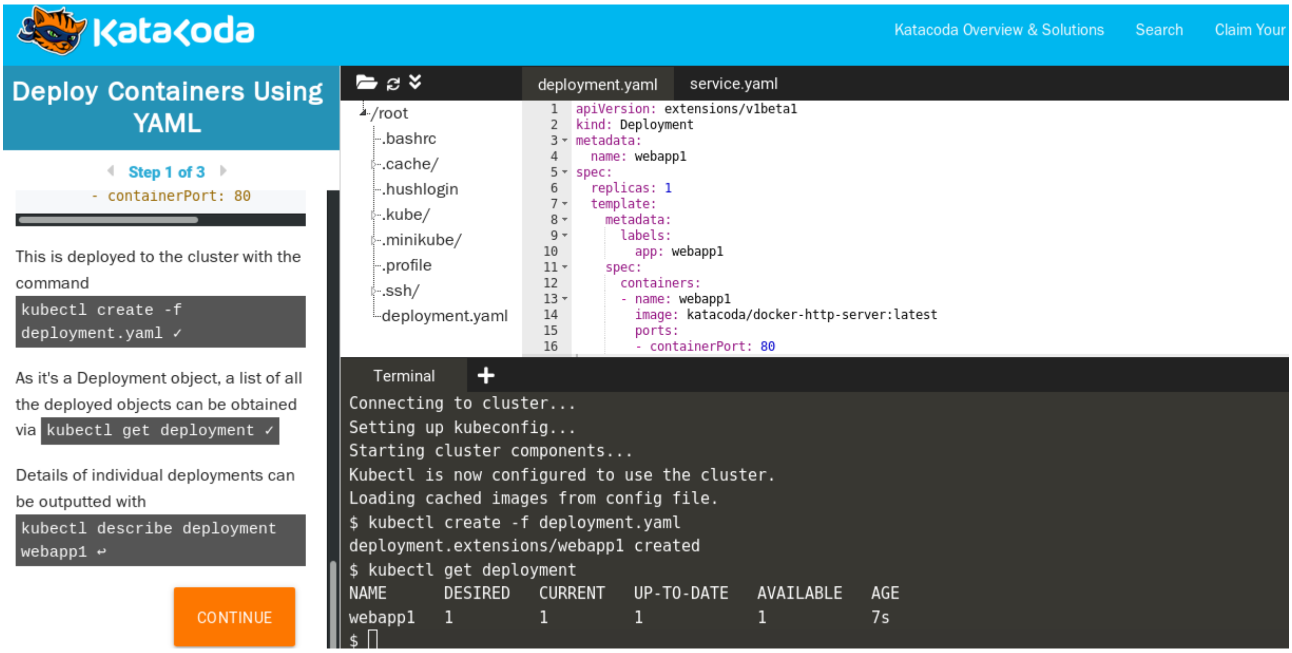
```
  Progressing    True    ReplicaSetUpdated
OldReplicaSets:  <none>
NewReplicaSet:   http-7b77c4cd66 (1/1 replicas created)
Events:
  Type    Reason             Age   From                   Message
  ----    ------             ----  ----                   -------
  Normal  ScalingReplicaSet  8s    deployment-controller  Scaled up replica set http-7b77c4cd6
$ kubectl expose deployment http --external-ip="172.17.0.24" --port=8000 --target-port=80
service/http exposed
$ curl http://172.17.0.24:8000
<h1>This request was processed by host: http-7b77c4cd66-xqs92</h1>
$ curl http://172.17.0.24:8000
<h1>This request was processed by host: http-7b77c4cd66-xqs92</h1>
$ curl http://172.17.0.24:8000
<h1>This request was processed by host: http-7b77c4cd66-xqs92</h1>
$ 
```



```
$ curl http://172.17.0.24:8001
curl: (7) Failed to connect to 172.17.0.24 port 8001: Connection refused
$ curl http://172.17.0.24:8001
curl: (7) Failed to connect to 172.17.0.24 port 8001: Connection refused
$ kubectl run httpexposed --image=katacoda/docker-http-server:latest --replicas=1 --port=80 --hostport=8001
deployment.apps/httpexposed created
$ curl http://172.17.0.24:8001
<h1>This request was processed by host: httpexposed-5c4cf8b7d8-4crl2</h1>
$ curl http://172.17.0.24:8001
<h1>This request was processed by host: httpexposed-5c4cf8b7d8-4crl2</h1>
$ 
```

Deploy Containers Using YAML

II.c) Captura donde se muestran los Deployment y Services.



```
- containerPort: 80
```

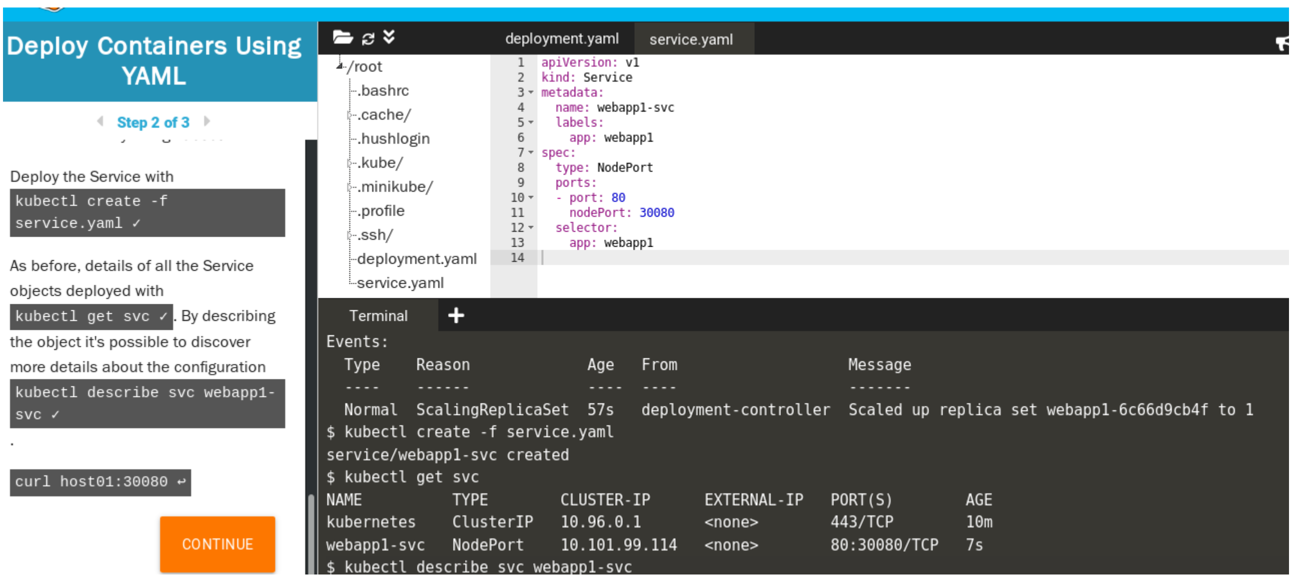
This is deployed to the cluster with the command `kubectl create -f deployment.yaml ✓`

As it's a Deployment object, a list of all the deployed objects can be obtained via `kubectl get deployment ✓`

Details of individual deployments can be outputted with `kubectl describe deployment webapp1 ↩`

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: webapp1
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: webapp1
    spec:
      containers:
      - name: webapp1
        image: katacoda/docker-http-server:latest
        ports:
        - containerPort: 80
```

```
Connecting to cluster...
Setting up kubeconfig...
Starting cluster components...
Kubectl is now configured to use the cluster.
Loading cached images from config file.
$ kubectl create -f deployment.yaml
deployment.extensions/webapp1 created
$ kubectl get deployment
NAME       DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
webapp1    1         1         1            1           7s
$ 
```



Deploy the Service with `kubectl create -f service.yaml ✓`

As before, details of all the Service objects deployed with `kubectl get svc ✓`. By describing the object it's possible to discover more details about the configuration `kubectl describe svc webapp1-svc ✓`

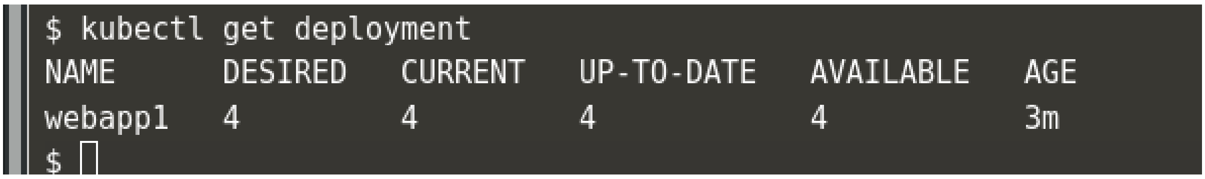
.

`curl host01:30080 ↩`

```
apiVersion: v1
kind: Service
metadata:
  name: webapp1-svc
  labels:
    app: webapp1
spec:
  type: NodePort
  ports:
  - port: 80
    nodePort: 30080
  selector:
    app: webapp1
```

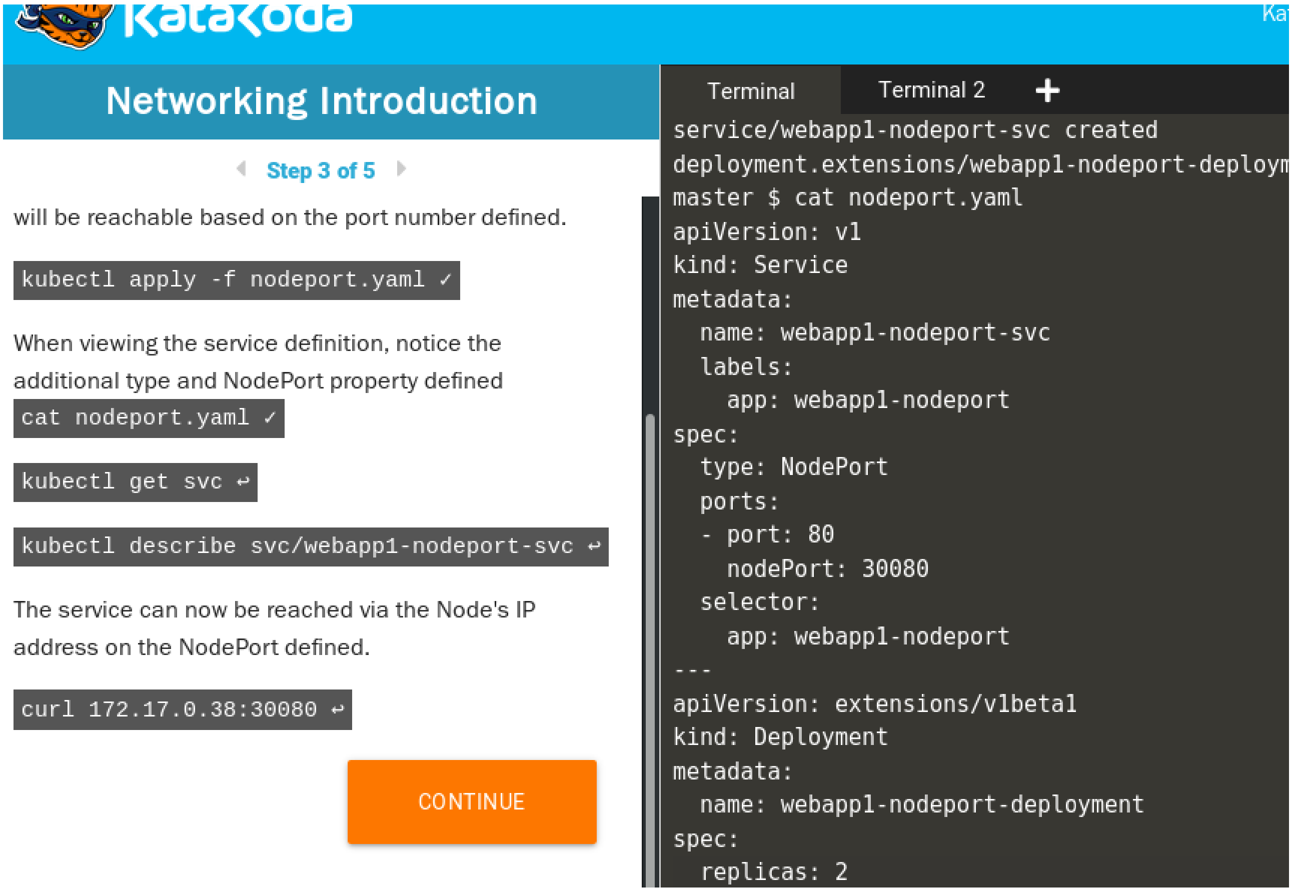
```
Events:
  Type    Reason             Age   From                   Message
  ----    ------             ----  ----                   -------
  Normal  ScalingReplicaSet  57s   deployment-controller  Scaled up replica set webapp1-6c66d9cb4f to 1
$ kubectl create -f service.yaml
service/webapp1-svc created
$ kubectl get svc
NAME          TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes    ClusterIP   10.96.0.1       <none>        443/TCP        10m
webapp1-svc   NodePort    10.101.99.114   <none>        80:30080/TCP   7s
$ kubectl describe svc webapp1-svc
```

Modificando el número de replicas a 4:



```
$ kubectl get deployment
NAME       DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
webapp1    4         4         4            4           3m
$ 
```

Kubernetes - Networking Introduction
II.d) Captura donde aparece personalizado el NodePort.



Networking Introduction

Step 3 of 5

will be reachable based on the port number defined.

```
kubectl apply -f nodeport.yaml ✓
```

When viewing the service definition, notice the additional type and NodePort property defined

```
cat nodeport.yaml ✓
```

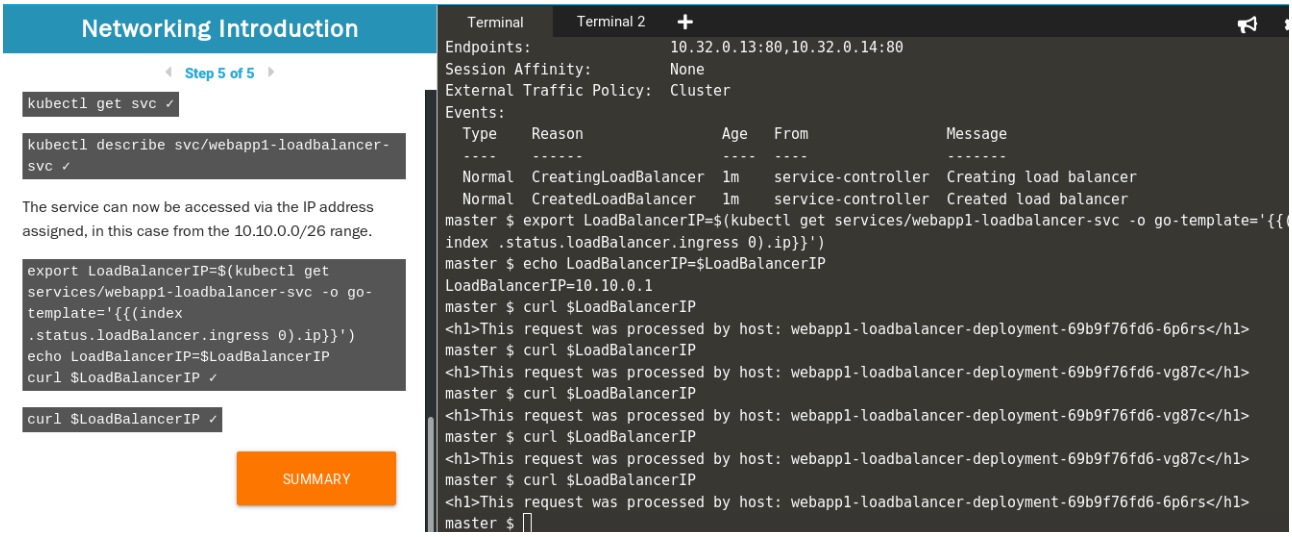
```
kubectl get svc ↩
```

```
kubectl describe svc/webapp1-nodeport-svc ↩
```

The service can now be reached via the Node's IP address on the NodePort defined.

```
curl 172.17.0.38:30080 ↩
```

```
service/webapp1-nodeport-svc created
deployment.extensions/webapp1-nodeport-deploym
master $ cat nodeport.yaml
apiVersion: v1
kind: Service
metadata:
  name: webapp1-nodeport-svc
  labels:
    app: webapp1-nodeport
spec:
  type: NodePort
  ports:
  - port: 80
    nodePort: 30080
  selector:
    app: webapp1-nodeport
---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: webapp1-nodeport-deployment
spec:
  replicas: 2
```



Networking Introduction

Step 5 of 5

```
kubectl get svc ✓
```

```
kubectl describe svc/webapp1-loadbalancer-svc ✓
```

The service can now be accessed via the IP address assigned, in this case from the 10.10.0.0/26 range.

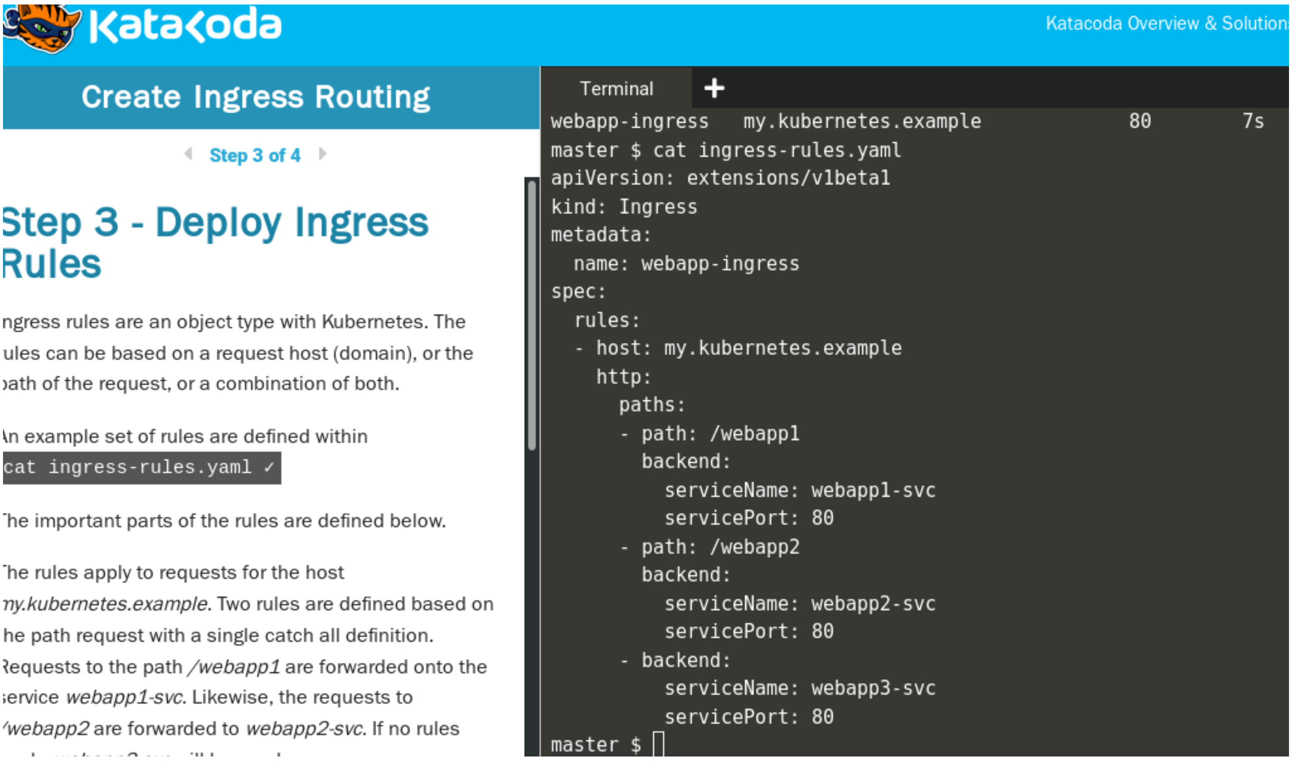
```
export LoadBalancerIP=$(kubectl get services/webapp1-loadbalancer-svc -o go-template='{{(index .status.loadBalancer.ingress 0).ip}}')
echo LoadBalancerIP=$LoadBalancerIP
curl $LoadBalancerIP ✓
```

```
curl $LoadBalancerIP ✓
```

```
Endpoints:                10.32.0.13:80,10.32.0.14:80
Session Affinity:         None
External Traffic Policy:  Cluster
Events:
  Type    Reason                Age   From                Message
  ----    ------                ----  ----                -------
  Normal  CreatingLoadBalancer  1m    service-controller  Creating load balancer
  Normal  CreatedLoadBalancer   1m    service-controller  Created load balancer
master $ export LoadBalancerIP=$(kubectl get services/webapp1-loadbalancer-svc -o go-template='{{(
index .status.loadBalancer.ingress 0).ip}}')
master $ echo LoadBalancerIP=$LoadBalancerIP
LoadBalancerIP=10.10.0.1
master $ curl $LoadBalancerIP
<h1>This request was processed by host: webapp1-loadbalancer-deployment-69b9f76fd6-6p6rs</h1>
master $ curl $LoadBalancerIP
<h1>This request was processed by host: webapp1-loadbalancer-deployment-69b9f76fd6-vg87c</h1>
master $ curl $LoadBalancerIP
<h1>This request was processed by host: webapp1-loadbalancer-deployment-69b9f76fd6-vg87c</h1>
master $ curl $LoadBalancerIP
<h1>This request was processed by host: webapp1-loadbalancer-deployment-69b9f76fd6-vg87c</h1>
master $ curl $LoadBalancerIP
<h1>This request was processed by host: webapp1-loadbalancer-deployment-69b9f76fd6-6p6rs</h1>
master $ 
```

Create Ingress Routing

II.e) Captura donde personalizamos el enrutamiento de nivel 4.



Create Ingress Routing

Step 3 of 4

Step 3 - Deploy Ingress Rules

ngress rules are an object type with Kubernetes. The ules can be based on a request host (domain), or the ›ath of the request, or a combination of both.

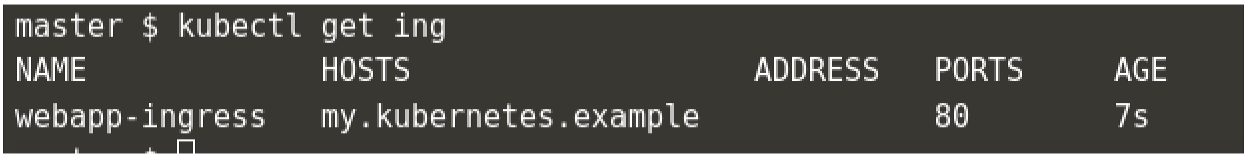
\n example set of rules are defined within

`cat ingress-rules.yaml ✓`

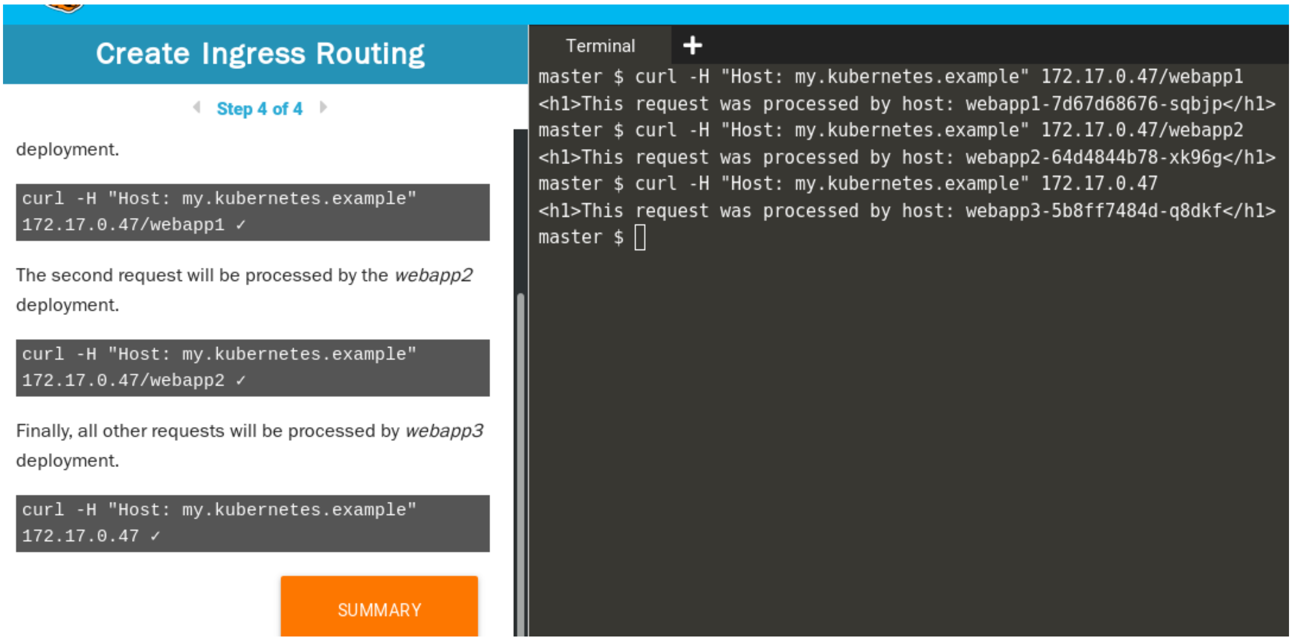
'he important parts of the rules are defined below.

'he rules apply to requests for the host *ny.kubernetes.example*. Two rules are defined based on he path request with a single catch all definition. Requests to the path */webapp1* are forwarded onto the ›ervice *webapp1-svc*. Likewise, the requests to *'webapp2* are forwarded to *webapp2-svc*. If no rules

```
webapp-ingress   my.kubernetes.example              80        7s
master $ cat ingress-rules.yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: webapp-ingress
spec:
  rules:
  - host: my.kubernetes.example
    http:
      paths:
      - path: /webapp1
        backend:
          serviceName: webapp1-svc
          servicePort: 80
      - path: /webapp2
        backend:
          serviceName: webapp2-svc
          servicePort: 80
      - backend:
          serviceName: webapp3-svc
          servicePort: 80
master $
```



```
master $ kubectl get ing
NAME             HOSTS                   ADDRESS   PORTS     AGE
webapp-ingress   my.kubernetes.example             80        7s
```



Create Ingress Routing

Step 4 of 4

deployment.

```
curl -H "Host: my.kubernetes.example"
172.17.0.47/webapp1 ✓
```

The second request will be processed by the *webapp2* deployment.

```
curl -H "Host: my.kubernetes.example"
172.17.0.47/webapp2 ✓
```

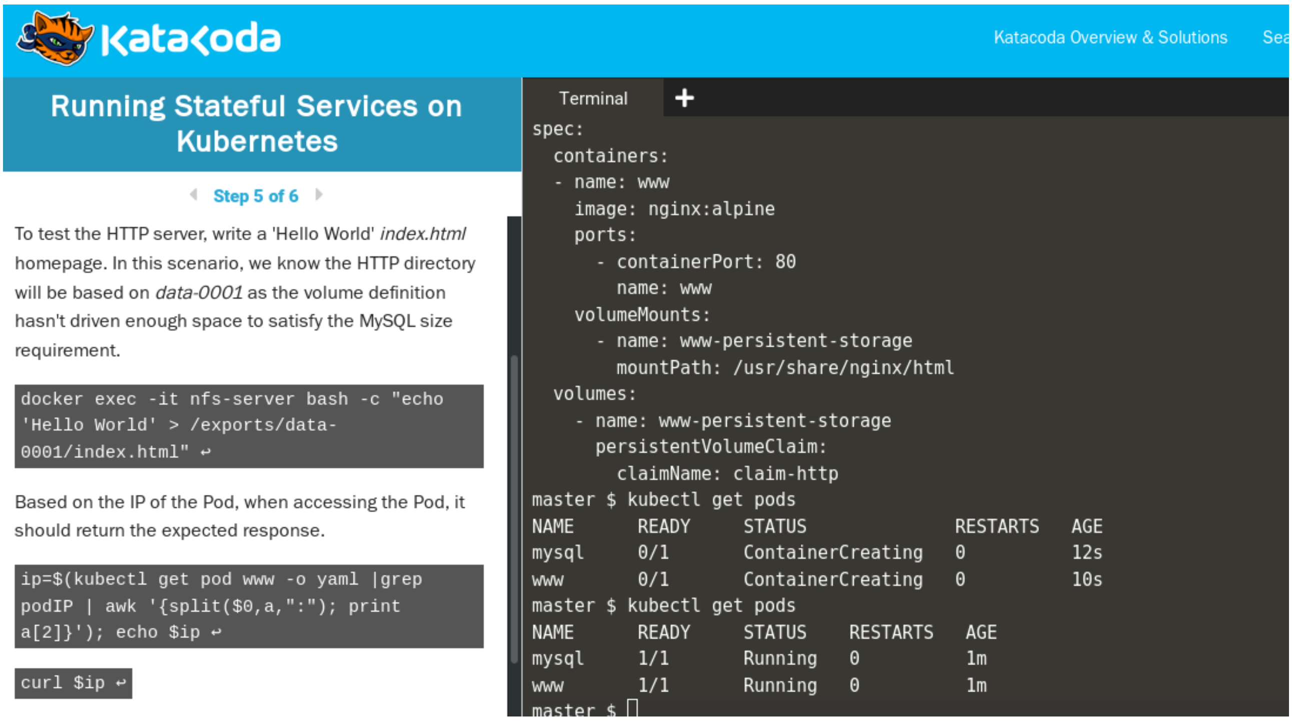
Finally, all other requests will be processed by *webapp3* deployment.

```
curl -H "Host: my.kubernetes.example"
172.17.0.47 ✓
```

```
master $ curl -H "Host: my.kubernetes.example" 172.17.0.47/webapp1
<h1>This request was processed by host: webapp1-7d67d68676-sqbjp</h1>
master $ curl -H "Host: my.kubernetes.example" 172.17.0.47/webapp2
<h1>This request was processed by host: webapp2-64d4844b78-xk96g</h1>
master $ curl -H "Host: my.kubernetes.example" 172.17.0.47
<h1>This request was processed by host: webapp3-5b8ff7484d-q8dkf</h1>
master $
```

Running Stateful Services on Kubernetes

II.f) Captura donde personalizamos el mensaje de salida en el almacenamiento persistente.



Running Stateful Services on Kubernetes

Step 5 of 6

To test the HTTP server, write a 'Hello World' *index.html* homepage. In this scenario, we know the HTTP directory will be based on *data-0001* as the volume definition hasn't driven enough space to satisfy the MySQL size requirement.

```
docker exec -it nfs-server bash -c "echo
'Hello World' > /exports/data-
0001/index.html" ↩
```

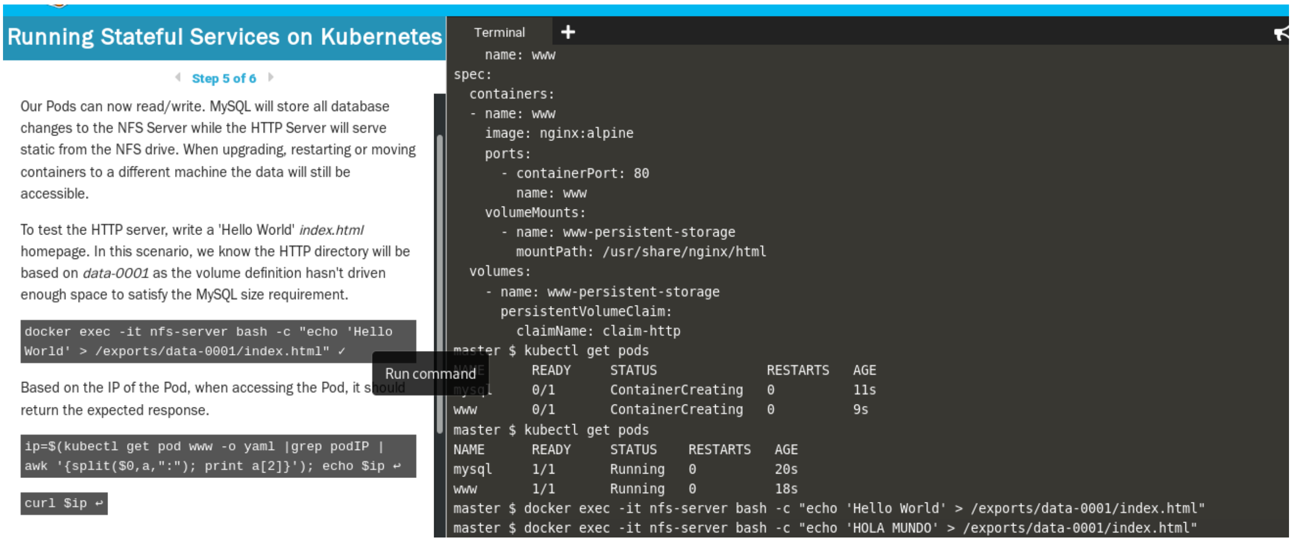
Based on the IP of the Pod, when accessing the Pod, it should return the expected response.

```
ip=$(kubectl get pod www -o yaml |grep
podIP | awk '{split($0,a,":"); print
a[2]}'); echo $ip ↩
```

```
curl $ip ↩
```

```
spec:
  containers:
  - name: www
    image: nginx:alpine
    ports:
      - containerPort: 80
        name: www
    volumeMounts:
      - name: www-persistent-storage
        mountPath: /usr/share/nginx/html
  volumes:
    - name: www-persistent-storage
      persistentVolumeClaim:
        claimName: claim-http
master $ kubectl get pods
NAME      READY     STATUS              RESTARTS   AGE
mysql     0/1       ContainerCreating   0          12s
www       0/1       ContainerCreating   0          10s
master $ kubectl get pods
NAME      READY     STATUS    RESTARTS   AGE
mysql     1/1       Running   0          1m
www       1/1       Running   0          1m
master $
```

Modificando el “HOLA MUNDO”.



Running Stateful Services on Kubernetes

Step 5 of 6

Our Pods can now read/write. MySQL will store all database changes to the NFS Server while the HTTP Server will serve static from the NFS drive. When upgrading, restarting or moving containers to a different machine the data will still be accessible.

To test the HTTP server, write a 'Hello World' *index.html* homepage. In this scenario, we know the HTTP directory will be based on *data-0001* as the volume definition hasn't driven enough space to satisfy the MySQL size requirement.

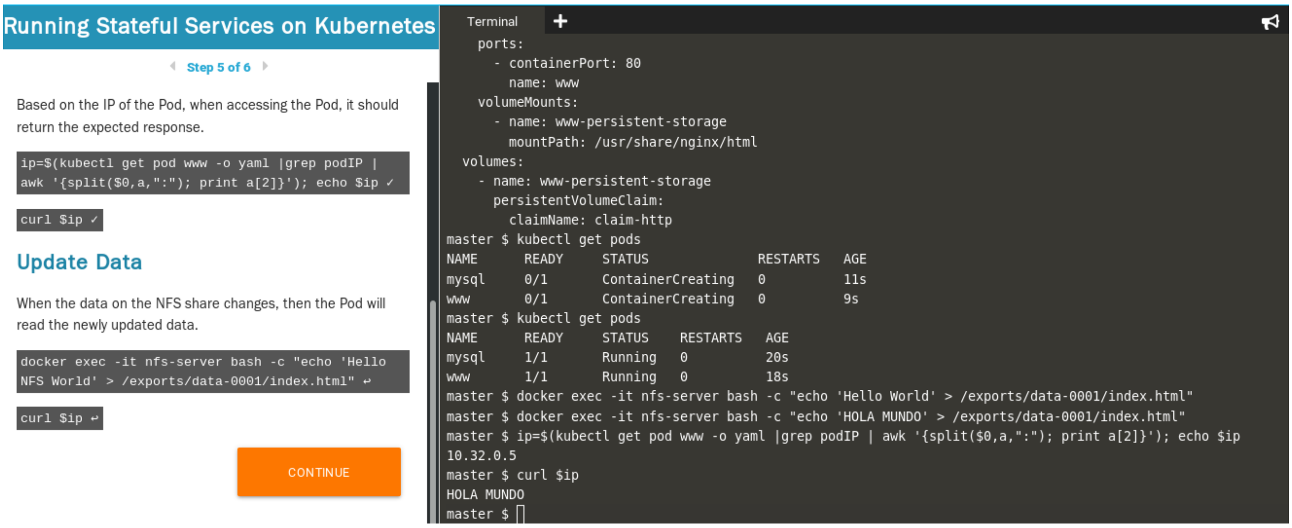
```
docker exec -it nfs-server bash -c "echo 'Hello
World' > /exports/data-0001/index.html" ✓
```

Based on the IP of the Pod, when accessing the Pod, it should return the expected response.

```
ip=$(kubectl get pod www -o yaml |grep podIP |
awk '{split($0,a,":"); print a[2]}'); echo $ip ↩
```

```
curl $ip ↩
```

```
    name: www
spec:
  containers:
  - name: www
    image: nginx:alpine
    ports:
      - containerPort: 80
        name: www
    volumeMounts:
      - name: www-persistent-storage
        mountPath: /usr/share/nginx/html
  volumes:
    - name: www-persistent-storage
      persistentVolumeClaim:
        claimName: claim-http
master $ kubectl get pods
NAME      READY     STATUS              RESTARTS   AGE
mysql     0/1       ContainerCreating   0          11s
www       0/1       ContainerCreating   0          9s
master $ kubectl get pods
NAME      READY     STATUS    RESTARTS   AGE
mysql     1/1       Running   0          20s
www       1/1       Running   0          18s
master $ docker exec -it nfs-server bash -c "echo 'Hello World' > /exports/data-0001/index.html"
master $ docker exec -it nfs-server bash -c "echo 'HOLA MUNDO' > /exports/data-0001/index.html"
```



Running Stateful Services on Kubernetes

Step 5 of 6

Based on the IP of the Pod, when accessing the Pod, it should return the expected response.

```
ip=$(kubectl get pod www -o yaml |grep podIP | awk '{split($0,a,":"); print a[2]}'); echo $ip ✓
```

```
curl $ip ✓
```

Update Data

When the data on the NFS share changes, then the Pod will read the newly updated data.

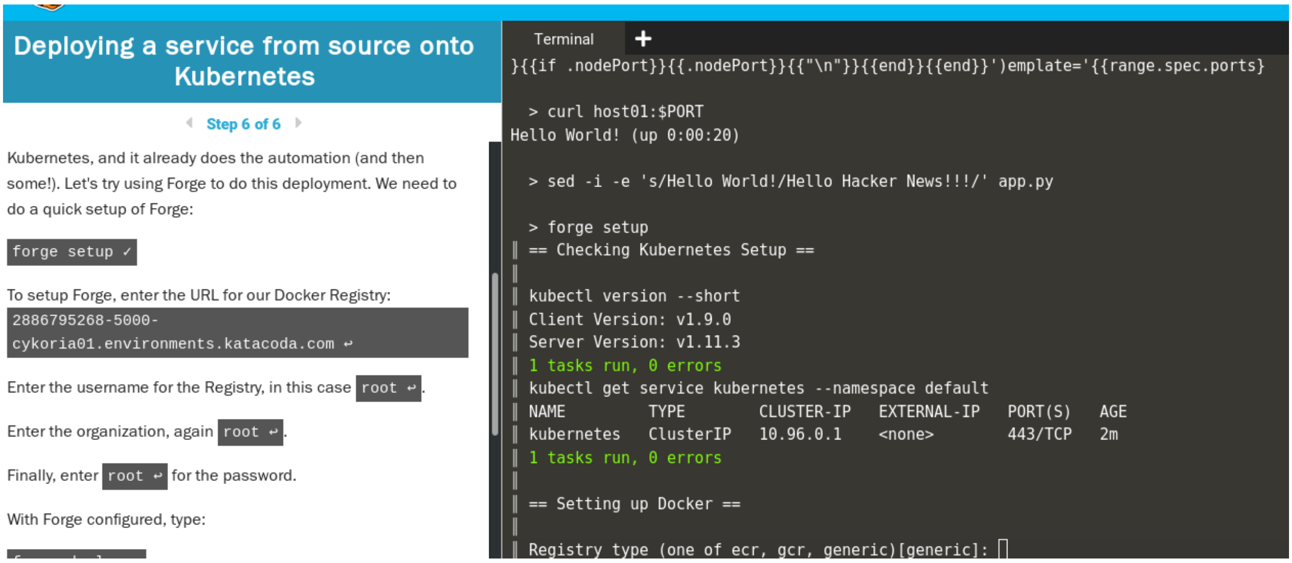
```
docker exec -it nfs-server bash -c "echo 'Hello NFS World' > /exports/data-0001/index.html" ↩
```

```
curl $ip ↩
```

```
    ports:
      - containerPort: 80
        name: www
    volumeMounts:
      - name: www-persistent-storage
        mountPath: /usr/share/nginx/html
  volumes:
    - name: www-persistent-storage
      persistentVolumeClaim:
        claimName: claim-http
master $ kubectl get pods
NAME      READY     STATUS              RESTARTS   AGE
mysql     0/1       ContainerCreating   0          11s
www       0/1       ContainerCreating   0          9s
master $ kubectl get pods
NAME      READY     STATUS    RESTARTS   AGE
mysql     1/1       Running   0          20s
www       1/1       Running   0          18s
master $ docker exec -it nfs-server bash -c "echo 'Hello World' > /exports/data-0001/index.html"
master $ docker exec -it nfs-server bash -c "echo 'HOLA MUNDO' > /exports/data-0001/index.html"
master $ ip=$(kubectl get pod www -o yaml |grep podIP | awk '{split($0,a,":"); print a[2]}'); echo $ip
10.32.0.5
master $ curl $ip
HOLA MUNDO
master $ 
```

Deploying a service from source onto Kubernetes

II.g) Captura de la ejecución de la automatización con forge



Deploying a service from source onto Kubernetes

Step 6 of 6

Kubernetes, and it already does the automation (and then some!). Let's try using Forge to do this deployment. We need to do a quick setup of Forge:

```
forge setup ✓
```

To setup Forge, enter the URL for our Docker Registry:

```
2886795268-5000-cykoria01.environments.katacoda.com ↩
```

Enter the username for the Registry, in this case `root ↩`.

Enter the organization, again `root ↩`.

Finally, enter `root ↩` for the password.

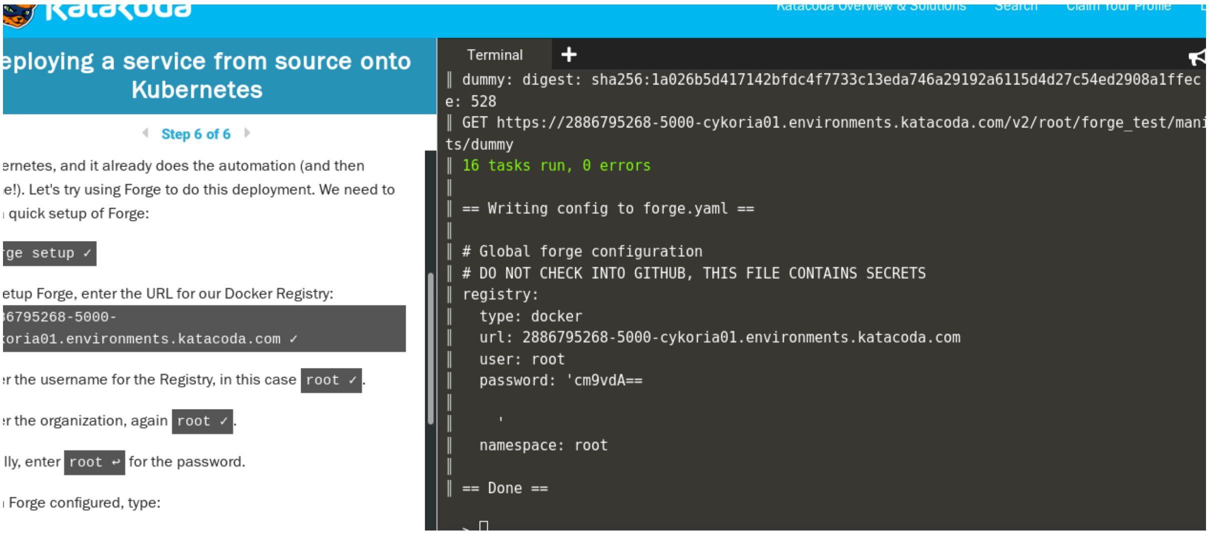
With Forge configured, type:

```
}{{if .nodePort}}{{.nodePort}}{{"\n"}}{{end}}{{end}}')emplate='{{range.spec.ports}

  > curl host01:$PORT
Hello World! (up 0:00:20)

  > sed -i -e 's/Hello World!/Hello Hacker News!!!/' app.py

  > forge setup
║ == Checking Kubernetes Setup ==
║
║ kubectl version --short
║ Client Version: v1.9.0
║ Server Version: v1.11.3
║ 1 tasks run, 0 errors
║ kubectl get service kubernetes --namespace default
║ NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
║ kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   2m
║ 1 tasks run, 0 errors
║
║ == Setting up Docker ==
║
║ Registry type (one of ecr, gcr, generic)[generic]: 
```



eploying a service from source onto Kubernetes

Step 6 of 6

ernetes, and it already does the automation (and then e!). Let's try using Forge to do this deployment. We need to quick setup of Forge:

```
ge setup ✓
```

etup Forge, enter the URL for our Docker Registry:

```
36795268-5000-
coria01.environments.katacoda.com ✓
```

r the username for the Registry, in this case `root ✓`.

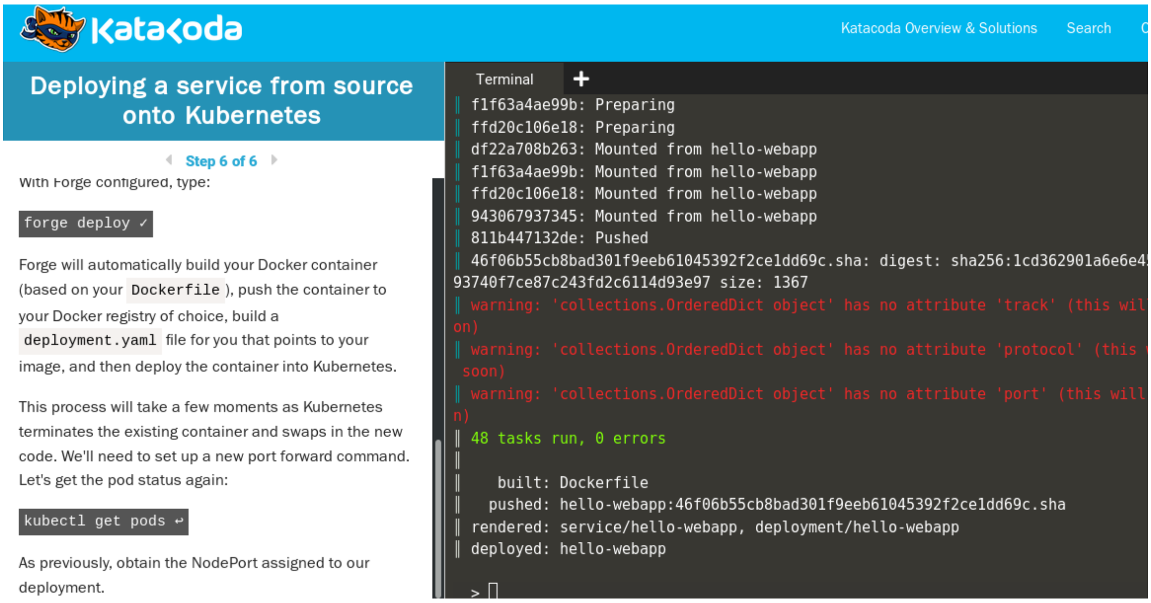
r the organization, again `root ✓`.

lly, enter `root ↵` for the password.

Forge configured, type:

```
║ dummy: digest: sha256:1a026b5d417142bfdc4f7733c13eda746a29192a6115d4d27c54ed2908a1ffec
e: 528
║ GET https://2886795268-5000-cykoria01.environments.katacoda.com/v2/root/forge_test/mani
ts/dummy
║ 16 tasks run, 0 errors
║
║ == Writing config to forge.yaml ==
║
║ # Global forge configuration
║ # DO NOT CHECK INTO GITHUB, THIS FILE CONTAINS SECRETS
║ registry:
║   type: docker
║   url: 2886795268-5000-cykoria01.environments.katacoda.com
║   user: root
║   password: 'cm9vdA==
║
║     '
║   namespace: root
║
║ == Done ==

  > 
```



Deploying a service from source onto Kubernetes

Step 6 of 6

With Forge configured, type:

```
forge deploy ✓
```

Forge will automatically build your Docker container (based on your `Dockerfile`), push the container to your Docker registry of choice, build a `deployment.yaml` file for you that points to your image, and then deploy the container into Kubernetes.

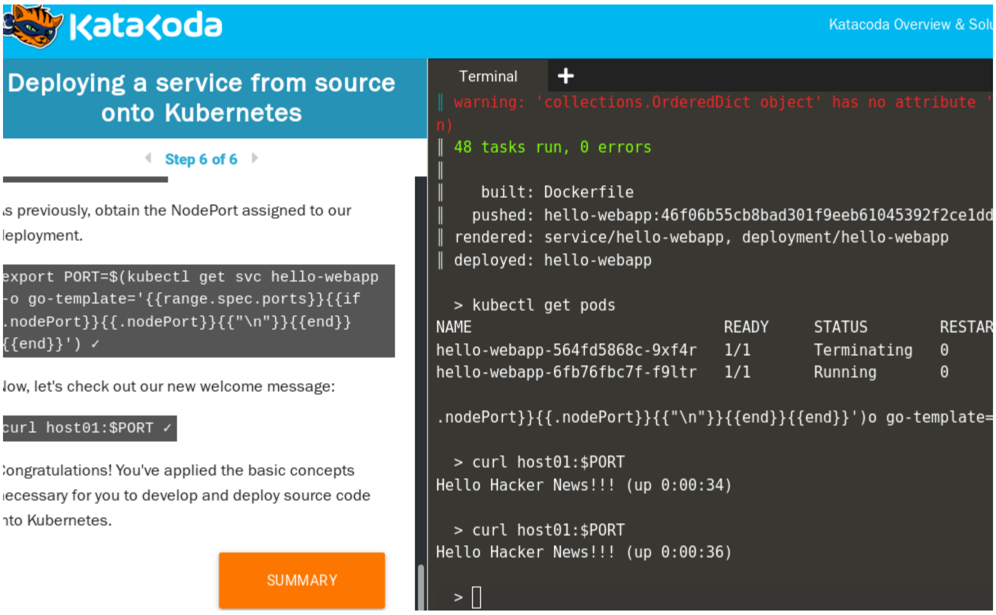
This process will take a few moments as Kubernetes terminates the existing container and swaps in the new code. We'll need to set up a new port forward command. Let's get the pod status again:

```
kubectl get pods ↵
```

As previously, obtain the NodePort assigned to our deployment.

```
║ f1f63a4ae99b: Preparing
║ ffd20c106e18: Preparing
║ df22a708b263: Mounted from hello-webapp
║ f1f63a4ae99b: Mounted from hello-webapp
║ ffd20c106e18: Mounted from hello-webapp
║ 943067937345: Mounted from hello-webapp
║ 811b447132de: Pushed
║ 46f06b55cb8bad301f9eeb61045392f2ce1dd69c.sha: digest: sha256:1cd362901a6e6e4
93740f7ce87c243fd2c6114d93e97 size: 1367
║ warning: 'collections.OrderedDict object' has no attribute 'track' (this wil
on)
║ warning: 'collections.OrderedDict object' has no attribute 'protocol' (this 
 soon)
║ warning: 'collections.OrderedDict object' has no attribute 'port' (this will
n)
║ 48 tasks run, 0 errors
║
║    built: Dockerfile
║   pushed: hello-webapp:46f06b55cb8bad301f9eeb61045392f2ce1dd69c.sha
║ rendered: service/hello-webapp, deployment/hello-webapp
║ deployed: hello-webapp

  > 
```



Deploying a service from source onto Kubernetes

Step 6 of 6

s previously, obtain the NodePort assigned to our leployment.

```
export PORT=$(kubectl get svc hello-webapp
-o go-template='{{range.spec.ports}}{{if
.nodePort}}{{.nodePort}}{{"\n"}}{{end}}
{{end}}') ✓
```

low, let's check out our new welcome message:

```
curl host01:$PORT ✓
```

Congratulations! You've applied the basic concepts necessary for you to develop and deploy source code nto Kubernetes.

```
║ warning: 'collections.OrderedDict object' has no attribute '
n)
║ 48 tasks run, 0 errors
║
║    built: Dockerfile
║   pushed: hello-webapp:46f06b55cb8bad301f9eeb61045392f2ce1dd
║ rendered: service/hello-webapp, deployment/hello-webapp
║ deployed: hello-webapp

  > kubectl get pods
NAME                               READY     STATUS        RESTAR
hello-webapp-564fd5868c-9xf4r      1/1       Terminating   0
hello-webapp-6fb76fbc7f-f9ltr      1/1       Running       0

.nodePort}}{{.nodePort}}{{"\n"}}{{end}}{{end}}')o go-template=

  > curl host01:$PORT
Hello Hacker News!!! (up 0:00:34)

  > curl host01:$PORT
Hello Hacker News!!! (up 0:00:36)

  > 
```