

CPD - Práctica 6

Elena Cantero Molina

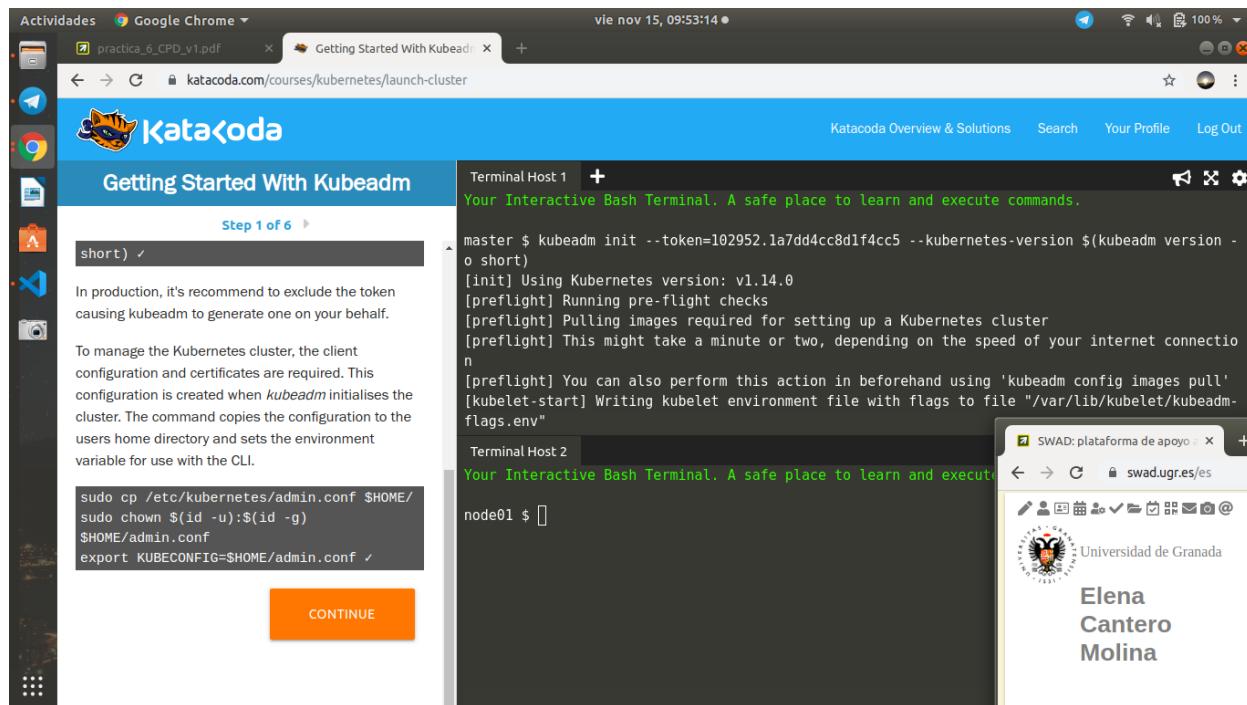
Kubernetes 101

I) Kubernetes

II) Primeros pasos con Kubernetes

II.a) Lanzamos: “Launch a multi-node cluster using Kubeadm”

Iniciamos los nodos de Kubernetes con Kubeadm



Activamos la capa de red que permite que se vean entre sí de forma transparente todos los contenidos situados en diversas máquinas, Container Network Interface (CNI).

The screenshot shows a desktop environment with a browser window titled "Getting Started With Kubeadm" open at katacoda.com/courses/kubernetes/launch-cluster. The browser also has tabs for "practica_6_CPD_v1.pdf" and "Getting Started With Kubeadm". To the right of the browser is a terminal window titled "Terminal Host 1" showing command-line output related to Kubernetes cluster setup. Below the terminal is another terminal window titled "Terminal Host 2" with the message "Your Interactive Bash Terminal. A safe place to learn and execute". On the far right, there is a small sidebar for "Universidad de Granada" with the name "Elena" and "Cantero Molina".

This screenshot is similar to the one above, showing the same browser setup and terminal windows. The terminal output in "Terminal Host 1" now shows a list of pods in the kube-system namespace, indicating the cluster is fully initialized. The "Your Interactive Bash Terminal" window is still present. The "Universidad de Granada" sidebar on the right remains the same.

Iniciamos un Pod

The screenshot shows a Katacoda interface. On the left, there's a sidebar with icons for file, camera, and network. The main area has a blue header with the Katacoda logo and navigation links like 'Katacoda Overview & Solutions', 'Search', 'Your Profile', and 'Log Out'. Below the header, it says 'Getting Started With Kubeadm' and 'Step 5 of 6'. The content area contains text about using Kubectl to deploy pods and a terminal window showing the command to create a deployment and its status.

```

Terminal Host 1
t bootstrap token generated by 'kubeadm init'. system:bootstrappers:kubeadm:default-node-token
en
master $ kubectl get nodes
NAME STATUS ROLES AGE VERSION
master Ready master 13m v1.14.0
node01 Ready <none> 10s v1.14.0
master $ kubectl create deployment http --image=katacoda/docker-http-server:latest
deployment.apps/http created
master $ kubectl get pods
NAME READY STATUS RESTARTS AGE
http-7f8cbdf584-4dcgw 1/1 Running 0 6s
master $ [REDACTED]
  
```

The status of the Pod creation can be viewed using `kubectl get pods`.

Once running, you can see the Docker Container running on the node.

```

docker ps | grep docker-http-server
  
```

CONTINUE

On the right side, there's another terminal window labeled 'Terminal Host 2' showing the node joining the cluster and a log entry from the kubelet.

```

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

Run 'kubectl get nodes' on the control-plane to see this node join.
  
```

```

node01 $ docker ps | grep docker-http-server
40073310ec96 katacoda/docker-http-server "/app"
Up 9 seconds
82ec567-0786-11ea-9a4d-0242ac10010_0
node01 $ [REDACTED]
  
```

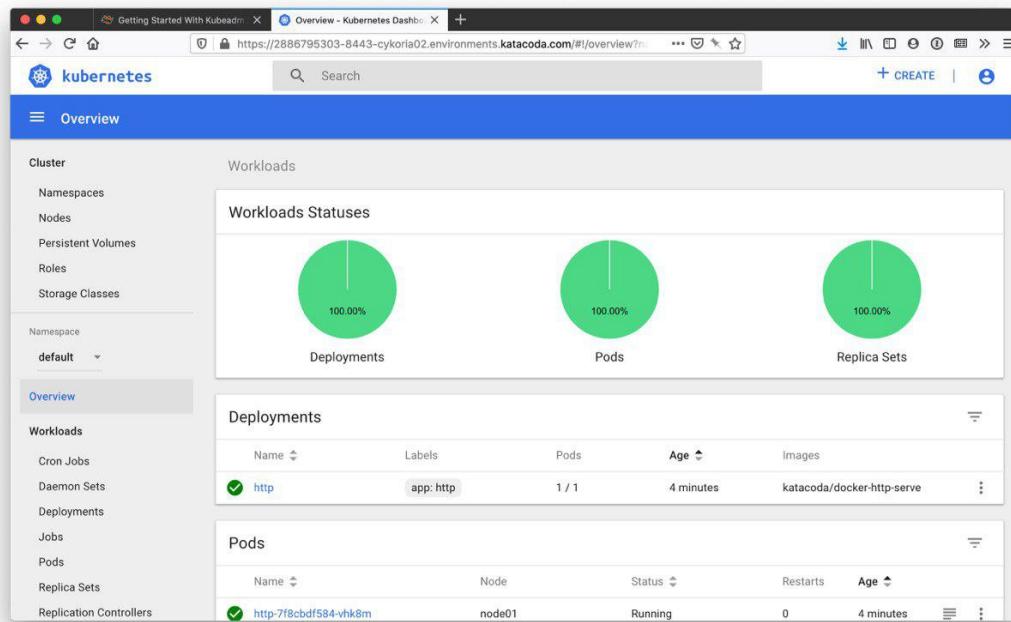
Activamos el Dashboard

A la hora de realizar esta parte me da este error

The screenshot shows a browser window with a 'Captura de pantalla' button. The address bar shows a URL starting with '2886795318-8443-ollie09.environments.katacoda.com'. The page content displays an error message: 'Esta página no funciona' (The page is not working) and 'La página 2886795318-8443-ollie09.environments.katacoda.com no puede procesar esta solicitud ahora.' (The page 2886795318-8443-ollie09.environments.katacoda.com cannot process this request now). There's also an 'HTTP ERROR 502' message. A purple 'Volver a cargar' (Reload) button is visible.

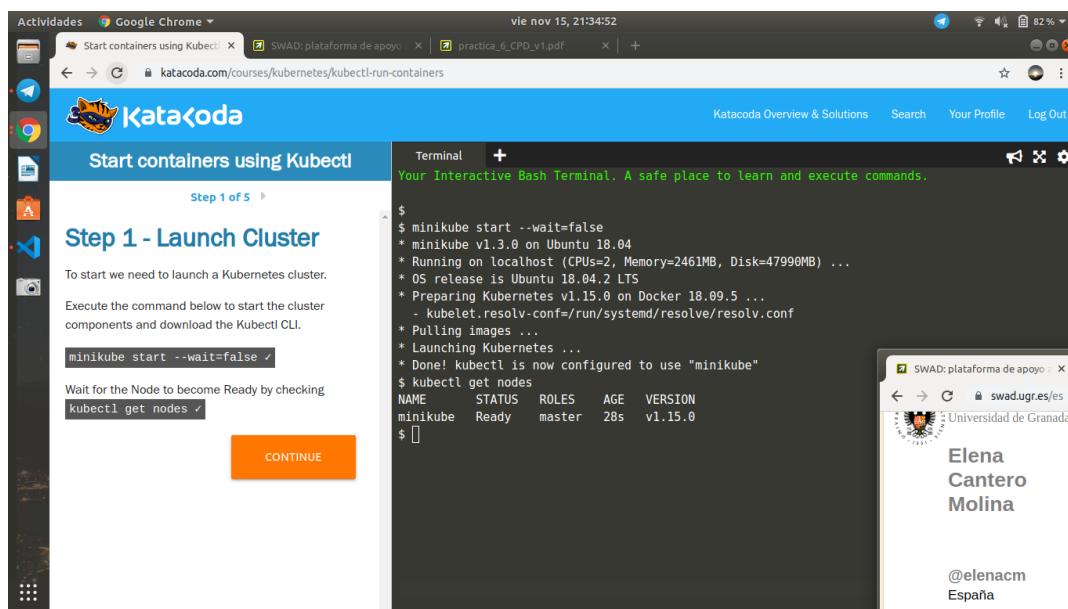
On the right side, there's a separate window titled 'SWAD: plataforma de apoyo' showing a profile for 'Elena Cantero Molina' with the handle '@elenacm' and location 'España'.

Como no he logrado arreglarlo, en la siguiente captura de pantalla le muestro lo que deberia salir con la captura de pantalla de un compañero de clase



II.b) Lanzamos “Deploy Containers Using Kubectl”

Lanzamos minikube



Ejecutamos un despliegue y exportamos el puerto para acceder

Vie nov 15, 21:40:01

Start containers using Kubectl

Step 3 of 5

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="172.17.0.39" --port=8000 --target-port=80 ✓
```

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

```
curl http://172.17.0.39:8000 ✓
```

CONTINUE

Terminal

```
Conditions:
  Type        Status  Reason
  ----        ----  -----
  Available   True    MinimumReplicasAvailable
  Progressing True    NewReplicaSetAvailable
OldReplicaSets: <none>
NewReplicaSet:  http-5fcf9dd9cb (1/1 replicas created)
Events:
  Type     Reason         Age   From           Message
  ----     ----         --   --            --
  Normal  ScalingReplicaSet 68s  deployment-controller  Scaled up replica set http-5fcf9dd9cb to 1
$ kubectl expose deployment http --external-ip="172.17.0.39" --port=8000 --target-port=80
$ curl http://172.17.0.39:8000
<h1>This request was processed by host: http-5fcf9dd9cb-zhpn6</h1>
$ curl http://172.17.0.39:8000
<h1>This request was processed by host: http-5fcf9dd9cb-zhpn6</h1>
$ curl http://172.17.0.39:8000
<h1>This request was processed by host: http-5fcf9dd9cb-zhpn6</h1>
$ curl http://172.17.0.39:8000
<h1>This request was processed by host: http-5fcf9dd9cb-zhpn6</h1>
$ curl http://172.17.0.39:8000
<h1>This request was processed by host: http-5fcf9dd9cb-zhpn6</h1>
$ curl http://172.17.0.39:8000
<h1>This request was processed by host: http-5fcf9dd9cb-zhpn6</h1>
```

SWAD:plataforma de apoyo x swad.ugr.es/ Universidad de Granada Elena Cantero Molina @elenacm España

Vie nov 15, 21:43:53

Start containers using Kubectl

Step 4 of 5

Use the command command to create a second http service exposed on port *8001*.

```
kubectl run httpexposed --image=katacoda/docker-http-server:latest --replicas=1 --port=80 --hostport=8001 ✓
```

You should be able to access it using

```
curl http://172.17.0.39:8001 ✓
```

Under the covers, this exposes the Pod via Docker Port Mapping. As a result, you will not see the service listed using *kubectl get svc*.

To find the details you can use

```
docker ps | grep httpexposed ✓
```

Pause Containers

Running the above command you'll notice the ports are exposed on the Pod, not the http container itself. The Pause container is responsible for defining the

Terminal

```
$ kubectl run httpexposed --image=katacoda/docker-http-server:latest --replicas=1 --port=80 --hostport=8001
kubectl run --generator=deployment/apps.v1 is DEPRECATED and will be removed in a future version. Use kubectl run --generator=run-pod/v1 or kubectl create instead.
deployment.apps/httpexposed created
$ curl http://172.17.0.39:8001
curl: (7) Failed to connect to 172.17.0.39 port 8001: Connection refused
$ curl http://172.17.0.39:8001
<h1>This request was processed by host: httpexposed-569df5d86-g2m8l</h1>
$ curl http://172.17.0.39:8001
<h1>This request was processed by host: httpexposed-569df5d86-g2m8l</h1>
$ curl http://172.17.0.39:8001
<h1>This request was processed by host: httpexposed-569df5d86-g2m8l</h1>
$ kubectl get svc
NAME      TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
http      ClusterIP  10.101.77.66  172.17.0.39      8000/TCP   2m5s
kubernetes  ClusterIP  10.96.0.1   <none>          443/TCP    8m2s
$ docker ps | grep httpexposed
ac5221le0f8f      katacoda/docker-http-server  "/app"
  Up About a minute                                k8s_httpexposed_httpexposed-569df5d86-g2m8l_0
  680cdede5a-155c-4362-9c20-98a6b4647b4d_0
a158f230250c      k8s.gcr.io/pause:3.1          "/pause"
  Up About a minute  0.0.0.0:8001->80/tcp  k8s_POD_httpexposed-569df5d86-g2m8l_0
$
```

SWAD:plataforma de apoyo x swad.ugr.es/ Universidad de Granada Elena Cantero Molina @elenacm España

Escalamos

```

e5a-155c-4362-9c20-98a6b4647b4d_0
$ kubectl scale --replicas=3 deployment http
deployment.extensions/http scaled
$ kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
http-5fcf9dd9cb-c67ls   0/1     ContainerCreating   0      3s
http-5fcf9dd9cb-mx2xp   0/1     ContainerCreating   0      3s
http-5fcf9dd9cb-zhpn6   1/1     Running   0      8m29s
httpexposed-569df5d86-q2m8l 1/1     Running   0      3m20s
$ kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
http-5fcf9dd9cb-c67ls   1/1     Running   0      5s
http-5fcf9dd9cb-mx2xp   1/1     Running   0      5s
http-5fcf9dd9cb-zhpn6   1/1     Running   0      8m31s
httpexposed-569df5d86-q2m8l 1/1     Running   0      3m22s
$ kubectl describe svc http
Name:           http
Namespace:      default
Labels:         run=http
Annotations:    <none>
Selector:       run=http
Type:          ClusterIP
IP:            10.101.77.66
External IPs:  172.17.0.39
Port:          <unset>  8000/TCP

```

Elena
Cantero
Molina
@elenacm
España

II.c) Lanzamos “Deploy Containers Using YAML”

Creamos un Deployment y un Servicio con ficheros YAML.

Step 1 of 3

```

image: katacoda/docker-http-server:latest
ports:
- 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`

CONTINUE

```

deployment.yaml  service.yaml
1 apiVersion: extensions/v1beta1
2 kind: Deployment
3 metadata:
4   name: webapp1
5   spec:
6     replicas: 1
7     template:
8       metadata:
9         labels:
10        app: webapp1
11        spec:
12          containers:
13            - name: webapp1
14              image: katacoda/docker-http-server:latest
15            ports:
16              - containerPort: 80
17

```

```

* Preparing Kubernetes v1.15.0 on Docker 18.09.5 ...
- kubelet.resolv.conf=/run/systemd/resolve/resolv.conf
* Pulling images ...
* Launching Kubernetes ...
* Done! kubectl is now configured to use "minikube"
$ kubectl create -f deployment.yaml
deployment.extensions/webapp1 created
$ kubectl get deployment
NAME      READY   UP-TO-DATE   AVAILABLE   AGE
webapp1  1/1     1           1           36s
$ kubectl describe deployment webapp1
Name:           webapp1

```

Elena
Cantero
Molina
@elenacm
España

Actividades Google Chrome ▾

vie nov 15, 21:50:57

Deploy Containers Using YAML | SWAD:plataforma de apoyo | practica_6_CPD_v1.pdf

Katacoda Overview & Solutions Search Your Profile Log Out

Katacoda

Deploy Containers Using YAML

Step 2 of 3

```
- port: 80
  nodePort: 30080
  selector:
    app: webapp1
```

All Kubernetes objects are deployed in a consistent way using `kubectl`.

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 ✓
```

CONTINUE

Terminal

```
$ kubectl get svc
NAME      TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
kubernetes   ClusterIP   10.96.0.1      <none>        443/TCP      75s
webapp1-svc  NodePort    10.105.104.243  <none>        80:30080/TCP  2s

$ kubectl describe svc webapp1-svc
Name:           webapp1-svc
Namespace:      default
Labels:         app=webapp1
Annotations:    <none>
Selector:       app=webapp1
Type:          NodePort
IP:            10.105.104.243
Port:          <unset>  80/TCP
TargetPort:    80/TCP
NodePort:      <unset>  30080/TCP
Endpoints:     172.18.0.3:80
Session Affinity: None
External Traffic Policy: Cluster
Events:        <none>

$ curl host01:30080
<h1>This request was processed by host: webapp1-5f4c9c89-tb4fw</h1>
```

SWAD:plataforma de apoyo | swad.ugr.es/es

Elena
Cantero
Molina

@elenacm
España

Actividades Google Chrome ▾

vie nov 15, 21:51:55

Deploy Containers Using YAML | SWAD:plataforma de apoyo | practica_6_CPD_v1.pdf

Katacoda Overview & Solutions Search Your Profile Log Out

Katacoda

Deploy Containers Using YAML

Step 3 of 3

the updated YAML file using
`kubectl apply -f deployment.yaml ✓`

Instantly, the desired state of our cluster has been updated, viewable with
`kubectl get deployment ✓`

Additional Pods will be scheduled to match the request.
`kubectl get pods ✓`

As all the Pods have the same label selector, they'll be load balanced behind the Service NodePort deployed.

Issuing requests to the port will result in different containers processing the request
`curl host01:30080 ✓`

Additional Kubernetes Networking details and Object Definitions will be covered in future scenarios.

CONTINUE

Terminal

```
deployment.extensions/webapp1 configured
$ kubectl get deployment
NAME      READY   UP-TO-DATE   AVAILABLE   AGE
webapp1  1/4     4           1           2m16s
$ kubectl get deployment
NAME      READY   UP-TO-DATE   AVAILABLE   AGE
webapp1  4/4     4           4           2m20s
$ kubectl get pods
NAME                           READY   STATUS   RESTARTS   AGE
webapp1-5f4c9c89-ksxh7  1/1     Running   0          10s
webapp1-5f4c9c89-sc54q  1/1     Running   0          10s
webapp1-5f4c9c89-tb4fw  1/1     Running   0          2m18s
webapp1-5f4c9c89-xmclm  1/1     Running   0          10s
$ curl host01:30080
<h1>This request was processed by host: webapp1-5f4c9c89-ksxh7</h1>
$ curl host01:30080
<h1>This request was processed by host: webapp1-5f4c9c89-xmclm</h1>
$
```

SWAD:plataforma de apoyo | swad.ugr.es/es

Elena
Cantero
Molina

@elenacm
España

II.d) Lanzamos “Kubernetes - Networking Introduction”

Cluster IP

Actividades Google Chrome ▾

vie nov 15, 21:55:07

Networking Introduction | Kula | SWAD:plataforma de apoyo... | practica_6_CPD_v1.pdf +

katacoda.com/courses/kubernetes/networking-introduction

Katacoda Overview & Solutions Search Your Profile Log Out

Katacoda

Networking Introduction

Step 1 of 5

Step 1 - Cluster IP

Cluster IP is the default approach when creating a Kubernetes Service. The service is allocated an internal IP that other components can use to access the pods.

By having a single IP address it enables the service to be load balanced across multiple Pods.

Services are deployed via `kubectl apply -f clusterip.yaml ✓`

The definition can be viewed at `cat clusterip.yaml ✓`

This will deploy a web app with two replicas to showcase load balancing along with a service. The Pods can be viewed at `kubectl get pods ✓`

It will also deploy a service. `kubectl get svc ✓`

Terminal

```
image: katacoda/docker-http-server:latest
ports:
- containerPort: 80
...
master $ kubectl get pods
NAME                                     READY   STATUS    RESTARTS   AGE
webapp1-clusterip-deployment-669c7c65c4-qn6fd   0/1     Pending   0          8s
webapp1-clusterip-deployment-669c7c65c4-zgdzd   0/1     Pending   0          8s
master $ kubectl get pods
NAME                                     READY   STATUS    RESTARTS   AGE
webapp1-clusterip-deployment-669c7c65c4-qn6fd   0/1     Pending   0          10s
webapp1-clusterip-deployment-669c7c65c4-zgdzd   0/1     Pending   0          10s
master $ kubectl get pods
NAME                                     READY   STATUS    RESTARTS   AGE
webapp1-clusterip-deployment-669c7c65c4-qn6fd   0/1     Pending   0          10s
webapp1-clusterip-deployment-669c7c65c4-zgdzd   0/1     Pending   0          10s
master $ kubectl get pods
NAME                                     READY   STATUS    RESTARTS   AGE
webapp1-clusterip-deployment-669c7c65c4-qn6fd   0/1     Pending   0          10s
webapp1-clusterip-deployment-669c7c65c4-zgdzd   0/1     Pending   0          10s
master $ kubectl get pods
NAME                                     READY   STATUS    RESTARTS   AGE
webapp1-clusterip-deployment-669c7c65c4-qn6fd   1/1     Running  0          10s
webapp1-clusterip-deployment-669c7c65c4-zgdzd   1/1     Running  0          10s
master $ ]
```

SWAD:plataforma de apoyo... swad.ugr.es/es Universidad de Granada

Elena
Cantero
Molina
@elenacm
España

Actividades Google Chrome ▾

vie nov 15, 21:55:35

Networking Introduction | Kula | SWAD:plataforma de apoyo... | practica_6_CPD_v1.pdf +

katacoda.com/courses/kubernetes/networking-introduction

Katacoda Overview & Solutions Search Your Profile Log Out

Katacoda

Networking Introduction

Step 1 of 5

endpoints (Pods) can be viewed via `kubectl describe svc/webapp1-clusterip-svc ✓`

After deploying, the service can be accessed via the ClusterIP allocated.

```
export CLUSTER_IP=$(kubectl get services/webapp1-clusterip-svc -o go-template='{{(index .spec.clusterIPs)}}')
master $ echo CLUSTER_IP=$CLUSTER_IP
CLUSTER_IP=10.106.212.98
master $ curl $CLUSTER_IP:80
<h1>This request was processed by host: webapp1-clusterip-deployment-669c7c65c4-qn6fd</h1>
master $ export CLUSTER_IP=$(kubectl get services/webapp1-clusterip-svc -o go-template='{{(index .spec.clusterIPs)}}')
master $ echo CLUSTER_IP=$CLUSTER_IP
CLUSTER_IP=10.106.212.98
master $ curl $CLUSTER_IP:80
<h1>This request was processed by host: webapp1-clusterip-deployment-669c7c65c4-zgdzd</h1>
master $ curl $CLUSTER_IP:80
<h1>This request was processed by host: webapp1-clusterip-deployment-669c7c65c4-qn6fd</h1>
master $ curl $CLUSTER_IP:80
<h1>This request was processed by host: webapp1-clusterip-deployment-669c7c65c4-zgdzd</h1>
master $ curl $CLUSTER_IP:80
<h1>This request was processed by host: webapp1-clusterip-deployment-669c7c65c4-qn6fd</h1>
master $ curl $CLUSTER_IP:80
<h1>This request was processed by host: webapp1-clusterip-deployment-669c7c65c4-zgdzd</h1>
```

curl \$CLUSTER_IP:80 ✓

CONTINUE

SWAD:plataforma de apoyo... swad.ugr.es/es Universidad de Granada

Elena
Cantero
Molina
@elenacm
España

Target Ports

Terminal

```

Annotations:      kubectl.kubernetes.io/last-applied-configuration:
                  {"apiVersion":"v1","kind":"Service","metadata":{"annotations":{},"labels":{},"name":"webapp1-clusterip-targetport-svc","namespace":"default"}, "spec":{"clusterIP": "10.98.76.136","selector":{"app": "webapp1-clusterip-targetport"}, "type": "ClusterIP"}, "status":{}}
Selector:        app=webapp1-clusterip-targetport
Type:           ClusterIP
IP:             10.98.76.136
Port:          <unset> 8080/TCP
TargetPort:     80/TCP
Endpoints:     10.32.0.7:80,10.32.0.8:80
Session Affinity: None
Events:         <none>
master $ export CLUSTER_IP=$(kubectl get services/webapp1-clusterip-targetport-svc -o yaml | {{(index .spec.clusterIP)}})
master $ echo $CLUSTER_IP
CLUSTER_IP=10.98.76.136
master $ curl $CLUSTER_IP:8080
<h1>This request was processed by host: webapp1-clusterip-targetport</h1>
master $ curl $CLUSTER_IP:8080
<h1>This request was processed by host: webapp1-clusterip-targetport</h1>
master $ curl $CLUSTER_IP:8080
<h1>This request was processed by host: webapp1-clusterip-targetport</h1>
master $ []

```

The application itself is still configured to listen on port 80. Kubernetes Service manages the translation between the two.

CONTINUE

NodePort

Terminal

```

webapp1-clusterip-targetport-svc  ClusterIP  10.98.76.136  <none>   8080/TCP  47
s
webapp1-nodeport-svc    NodePort   10.101.62.233  <none>   80:30080/TCP  3s
master $ kubectl describe svc/webapp1-nodeport-svc
Name:           webapp1-nodeport-svc
Namespace:      default
Labels:         app=webapp1-nodeport
Annotations:    kubectl.kubernetes.io/last-applied-configuration:
                  {"apiVersion":"v1","kind":"Service","metadata":{"annotations":{},"labels":{},"name":"webapp1-nodeport-svc","namespace":"default"}, "spec":{"clusterIP": null,"nodePort": 30080,"ports": [{"port": 80,"targetPort": 80,"nodePort": 30080,"protocol": "TCP"}],"selector":{"app": "webapp1-nodeport"}, "status":{}}
Labels:         {"app": "webapp1-nodeport", "name": "webapp1-nodeport-svc", "namespace": "default"}
Selector:       app=webapp1-nodeport
Type:          NodePort
IP:            10.101.62.233
Port:          <unset> 80/TCP
TargetPort:    80/TCP
NodePort:      <unset> 30080/TCP
Endpoints:    10.32.0.9:80
Session Affinity: None
External Traffic Policy: Cluster
Events:        <none>
master $ curl 172.17.0.85:30080
<h1>This request was processed by host: webapp1-nodeport-deployment</h1>
master $ curl 172.17.0.85:30080
<h1>This request was processed by host: webapp1-nodeport-deployment</h1>
master $ []

```

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

curl 172.17.0.85:30080

External IPs

The service is now bound to the IP address and Port 80 of the master node.

```

cat externalip.yaml
kubectl apply -f externalip.yaml
kubectl get svc
kubectl describe svc/webapp1-externalip-svc
curl 172.17.0.85

```

CONTINUE

```

webapp1-nodeport-svc      NodePort  10.101.62.233  <none>   80:30080/TCP  3
4s
master $ kubectl describe svc/webapp1-externalip-svc
Name:           webapp1-externalip-svc
Namespace:     default
Labels:         app=webapp1-externalip
Annotations:   kubectl.kubernetes.io/last-applied-configuration:
               {"apiVersion":"v1","kind":"Service","metadata":{"annotations":{},"labels":{"app":"webapp1-externalip"},"name":"webapp1-externalip-svc","namespace":"default"}}
               {"app":"webapp1-externalip","name":"webapp1-externalip-svc","namespace":"default"}
Selector:       app=webapp1-externalip
Type:          ClusterIP
IP:            10.109.250.242
External IPs:  172.17.0.85
Port:          <unset> 80/TCP
TargetPort:    80/TCP
Endpoints:    <none>
Session Affinity: None
Events:        <none>
master $ curl 172.17.0.85
<h1>This request was processed by host: webapp1-externalip-deployment<br/>master $ curl 172.17.0.85
<h1>This request was processed by host: webapp1-externalip-deployment<br/>master $ curl 172.17.0.85
<h1>This request was processed by host: webapp1-externalip-deployment<br/>master $ []

```

Load Balancer

provider.

As Katacoda is not a cloud provider, it's still possible to dynamically allocate IP addresses to LoadBalancer type services. This is done by deploying the Cloud Provider using

```

kubectl apply -f clouddriver.yaml

```

When running in a service provided by a Cloud Provider this is not required.

When a service requests a Load Balancer, the provider will allocate one from the 10.10.0.0/26 range defined in the configuration.

```

kubectl get pods -n kube-system
kubectl apply -f loadbalancer.yaml

```

The service is configured via a Load Balancer as defined in `cat loadbalancer.yaml`.

While the IP address is being defined, the service will show Pending. When allocated, it will appear in the

```

katacoda-cloud-provider-548f9bd9dc-7ktrb  1/1  Running   0   5m15s
keepalived-cloud-provider-78fc4468b-c4stt  0/1  ContainerCreating   0   2s
kube-apiserver-master                    1/1  Running   0   4m16s
kube-controller-manager-master           1/1  Running   0   4m3s
kube-keepalived-vip-tmbdt               1/1  Running   0   4m28s
kube-proxy-rzvfp                        1/1  Running   0   5m14s
kube-scheduler-master                   1/1  Running   0   4m31s
weave-net-m2k5x                          2/2  Running   1   5m15s
master $ kubectl get pods -n kube-system
NAME                           READY   STATUS    RESTARTS   AGE
coredns-fb8b8ddcf-h2skd       1/1    Running   0          5m15s
coredns-fb8b8ddcf-mwkkv       1/1    Running   0          5m15s
etcd-master                     1/1    Running   0          5m15s
katacoda-cloud-provider-548f9bd9dc-7ktrb  1/1    Running   0          5m15s
keepalived-cloud-provider-78fc4468b-c4stt  1/1    Running   0          5m15s
kube-apiserver-master           1/1    Running   0          5m15s
kube-controller-manager-master  1/1    Running   0          5m15s
kube-keepalived-vip-tmbdt      1/1    Running   0          5m15s
kube-proxy-rzvfp                1/1    Running   0          5m15s
kube-scheduler-master          1/1    Running   0          5m15s
weave-net-m2k5x                 2/2    Running   1          5m15s
master $ kubectl apply -f loadbalancer.yaml
service/webapp1-loadbalancer-svc created
deployment.extensions/webapp1-loadbalancer-deployment created
master $ cat loadbalancer.yaml

```

The screenshot shows a Katacoda terminal session titled "Networking Introduction". In the terminal, the user runs `kubctl get svc` and `kubctl describe svc/webapp1-loadbalancer-svc`. The output shows a LoadBalancer service with IP 10.102.57.213 and port 80/TCP. The user then exports the LoadBalancerIP and uses curl to access it. A separate browser window shows the Katacoda profile of Elena Cantero Molina.

```

Type: LoadBalancer
IP: 10.102.57.213
LoadBalancer Ingress: 172.17.0.85
Port: <unset> 80/TCP
TargetPort: 80/TCP
NodePort: <unset> 31472/TCP
Endpoints: 10.32.0.14:80,10.32.0.15:80
Session Affinity: None
External Traffic Policy: Cluster
Events:
  Type Reason Age From Message
  ---- ---- -- -- -----
  Normal CreatingLoadBalancer 19s service-controller Creating load balancer
  Normal CreatedLoadBalancer 19s service-controller Created load balancer
master $ export LoadBalancerIP=$(kubectl get services/webapp1-loadbalancer-svc -o yaml | grep 'loadBalancer.ingress 0.ip' | cut -d: -f2)
master $ echo LoadBalancerIP=$LoadBalancerIP
LoadBalancerIP=172.17.0.85
master $ curl $LoadBalancerIP
<h1>This request was processed by host: webapp1-loadbalancer-deployment</h1>
master $ curl $LoadBalancerIP
<h1>This request was processed by host: webapp1-loadbalancer-deployment</h1>
master $ curl $LoadBalancerIP
<h1>This request was processed by host: webapp1-loadbalancer-deployment</h1>
master $ 

```

II.e) Lanzamos “Create Ingress Routing”

The screenshot shows a Katacoda terminal session titled "Create Ingress Routing". In the terminal, the user applies a deployment YAML file, creating three deployments: webapp1, webapp2, and webapp3. The user then checks the deployment status with `kubectl get deployment`. A separate browser window shows the Katacoda profile of Elena Cantero Molina.

```

labels:
  app: webapp3
spec:
  ports:
  - port: 80
  selector:
    app: webapp3
master $ kubectl apply -f deployment.yaml
deployment.extensions/webapp1 created
deployment.extensions/webapp2 created
deployment.extensions/webapp3 created
service/webapp1-svc created
service/webapp2-svc created
service/webapp3-svc created
master $ kubectl get deployment
NAME READY UP-TO-DATE AVAILABLE AGE
webapp1 0/1 1 0 2s
webapp2 0/1 1 0 2s
webapp3 0/1 1 0 2s
master $ kubectl get deployment
NAME READY UP-TO-DATE AVAILABLE AGE
webapp1 1/1 1 1 37s
webapp2 1/1 1 1 37s
webapp3 1/1 1 1 37s
master $ 

```

Actividades Google Chrome ▾

vie nov 15, 22:34:55

Create Ingress Routing | Kube | SWAD:plataforma de apoyo | practica_6_CPD_v1.pdf | +

l katacoda.com/courses/kubernetes/create-kubernetes-ingress-routes

Katacoda

Create Ingress Routing

Step 2 of 4

connections and is required by the Nginx Default Example.

```
cat ingress.yaml ✓
```

Task

The Ingress controllers are deployed in a familiar fashion to other Kubernetes objects with `kubectl create -f ingress.yaml`.

The status can be identified using `kubectl get deployment -n nginx-ingress`.

CONTINUE

Terminal

```
name: https
selector:
  app: nginx-ingress
externalIPs:
  - 172.17.0.74
master $ kubectl create -f ingress.yaml
namespace/nginx-ingress created
secret/default-server-secret created
serviceaccount/nginx-ingress created
configmap/nginx-config created
deployment.extensions/nginx-ingress created
service/nginx-ingress created
master $ kubectl get deployment -n nginx-ingress
NAME      READY  UP-TO-DATE   AVAILABLE   AGE
nginx-ingress  0/1    1          0           2s
master $ kubectl get deployment -n nginx-ingress
NAME      READY  UP-TO-DATE   AVAILABLE   AGE
nginx-ingress  0/1    1          0           5s
master $ kubectl get deployment -n nginx-ingress
NAME      READY  UP-TO-DATE   AVAILABLE   AGE
nginx-ingress  0/1    1          0           11s
master $ kubectl get deployment -n nginx-ingress
NAME      READY  UP-TO-DATE   AVAILABLE   AGE
nginx-ingress  1/1    1          1           16s
master $ []
```

SWAD:plataforma de apoyo x swad.ugr.es/es Universidad de Granada

Elena
Cantero
Molina
@elenacm
España

Actividades Google Chrome ▾

vie nov 15, 22:35:22

Create Ingress Routing | Kube | SWAD:plataforma de apoyo | practica_6_CPD_v1.pdf | +

l katacoda.com/courses/kubernetes/create-kubernetes-ingress-routes

Katacoda

Create Ingress Routing

Step 3 of 4

- path: /webapp1
 backend:
 - serviceName: webapp1-svc
 servicePort: 80
- path: /webapp2
 backend:
 - serviceName: webapp2-svc
 servicePort: 80
- backend:
 - serviceName: webapp3-svc
 servicePort: 80

Task

As with all Kubernetes objects, they can be deployed via `kubectl create -f ingress-rules.yaml`.

Once deployed, the status of all the Ingress rules can be discovered via `kubectl get ing`.

CONTINUE

Terminal

```
spec:
rules:
- host: my.kubernetes.example
  http:
    paths:
      - path: /webapp1
        backend:
          serviceName: webapp1-svc
          servicePort: 80
      - path: /webapp2
        backend:
          serviceName: webapp2-svc
          servicePort: 80
      - path: /webapp3
        backend:
          serviceName: webapp3-svc
          servicePort: 80
master $ kubectl create -f ingress-rules.yaml
ingress.extensions/webapp-ingress created
master $ kubectl get ing
NAME      HOSTS          ADDRESS     PORTS   AGE
webapp-ingress  my.kubernetes.example  80        1s
master $ kubectl get ing
NAME      HOSTS          ADDRESS     PORTS   AGE
webapp-ingress  my.kubernetes.example  80        8s
master $ []
```

SWAD:plataforma de apoyo x swad.ugr.es/es Universidad de Granada

Elena
Cantero
Molina
@elenacm
España

Actividades Google Chrome ▾

vie nov 15, 22:36:02

Create Ingress Routing | Kube | SWAD:plataforma de apoyo | practica_6_CPD_v1.pdf | +

l katacoda.com/courses/kubernetes/create-kubernetes-ingress-routes

Katacoda

Create Ingress Routing

Step 4 of 4

172.17.0.74/webapp1 ✓

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

```
curl -H "Host: my.kubernetes.example" 172.17.0.74/webapp2 ✓
```

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

```
curl -H "Host: my.kubernetes.example" 172.17.0.74 ✓
```

CONTINUE

Terminal

```
master $ curl -H "Host: my.kubernetes.example" 172.17.0.74/webapp1
<h1>This request was processed by host: webapp1-6d7df9f8d-kv5dk</h1>
master $ curl -H "Host: my.kubernetes.example" 172.17.0.74/webapp2
<h1>This request was processed by host: webapp2-6d48b8ff76-zwxgf</h1>
master $ curl -H "Host: my.kubernetes.example" 172.17.0.74
<h1>This request was processed by host: webapp3-7df59dc67b-wqjv4</h1>
master $ curl -H "Host: my.kubernetes.example" 172.17.0.74/webapp1
<h1>This request was processed by host: webapp1-6d7df9f8d-kv5dk</h1>
master $ curl -H "Host: my.kubernetes.example" 172.17.0.74/webapp1
<h1>This request was processed by host: webapp1-6d7df9f8d-kv5dk</h1>
master $ curl -H "Host: my.kubernetes.example" 172.17.0.74/webapp1
<h1>This request was processed by host: webapp1-6d7df9f8d-kv5dk</h1>
master $ curl -H "Host: my.kubernetes.example" 172.17.0.74/webapp2
<h1>This request was processed by host: webapp2-6d48b8ff76-zwxgf</h1>
master $ curl -H "Host: my.kubernetes.example" 172.17.0.74/webapp2
<h1>This request was processed by host: webapp2-6d48b8ff76-zwxgf</h1>
master $ curl -H "Host: my.kubernetes.example" 172.17.0.74
<h1>This request was processed by host: webapp3-7df59dc67b-wqjv4</h1>
master $ curl -H "Host: my.kubernetes.example" 172.17.0.74
<h1>This request was processed by host: webapp3-7df59dc67b-wqjv4</h1>
master $ curl -H "Host: my.kubernetes.example" 172.17.0.74
<h1>This request was processed by host: webapp3-7df59dc67b-wqjv4</h1>
master $ []
```

SWAD:plataforma de apoyo x swad.ugr.es/es Universidad de Granada

Elena
Cantero
Molina
@elenacm
España

II.f) Lanzamos “Running Stateful Services on Kubernetes”

Activamos un servidor NFS.

The screenshot shows a Katacoda interface with a terminal window. The terminal output is as follows:

```

Terminal + 
Your Interactive Bash Terminal. A safe place to learn and execute commands.

master $ launch.sh
Waiting for Kubernetes to start...
Kubernetes started
master $ docker run -d --net=host \
>   --privileged --name nfs-server \
>   katacoda/contained-nfs-server:centos7 \
>   /exports/data-0001 /exports/data-0002
Unable to find image 'katacoda/contained-nfs-server:centos7' locally
centos7: Pulling from katacoda/contained-nfs-server
8d30e94188e7: Pull complete
2b2b27f1f462: Pull complete
133e63cf95fe: Pull complete
Digest: sha256:5f2ea4737fe27f26be5b5cabaa23e24180079a4dce8d5db235492
Status: Downloaded newer image for katacoda/contained-nfs-server:centos7
e152d516f8c650d68b40fc317316d5df8ea12d4a12e0c0b298c2a4989bc74be3
master $ []

```

The NFS server exposes two directories, `data-0001` and `data-0002`. In the next steps, this is used to store data.

CONTINUE

Desplegamos los volúmenes persistentes (PV)

The screenshot shows a Katacoda interface with a terminal window. The terminal output is as follows:

```

Terminal + 
nfs:
  server: 172.17.0.64
  path: /exports/data-0001
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfs-0002
spec:
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteOnce
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Recycle
  nfs:
    server: 172.17.0.64
    path: /exports/data-0002
master $ kubectl get pv
NAME      CAPACITY   ACCESS MODES  RECLAIM POLICY  STATUS   CLAIM
nfs-0001  2Gi        RWO,RWX     Recycle        Available
          17s
nfs-0002  5Gi        RWO,RWX     Recycle        Available
          14s
master $ []

```

Create two new PersistentVolume definitions to point at the two available NFS shares.

```

kubectl create -f nfs-0001.yaml ✓
kubectl create -f nfs-0002.yaml ✓

```

View the contents of the files using

```

cat nfs-0001.yaml nfs-0002.yaml ✓

```

Once created, view all PersistentVolumes in the cluster using `kubectl get pv`

CONTINUE

Desplegamos los recursos que demandan volúmenes persistentes (PVC)

Running Stateful Services on Kubernetes

Task

Create two claims for two different applications. A MySQL Pod will use one claim, the other used by an HTTP server.

```
kubectl create -f pvc-mysql.yaml ✓
kubectl create -f pvc-http.yaml ✓
```

View the contents of the files using

```
cat pvc-mysql.yaml pvc-http.yaml ✓
```

Once created, view all `PersistentVolumeClaims` in the cluster using `kubectl get pvc`.

The claim will output which Volume the claim is mapped to.

CONTINUE

Terminal

```
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 3Gi
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: claim-mysql
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
master $ kubectl get pvc
NAME        STATUS   VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS
claim-mysql Bound    nfs-0002  5Gi        RWO,RWX
claim-mysql Bound    nfs-0001  2Gi        RWO,RWX
master $ kubectl get pvc
NAME        STATUS   VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS
claim-mysql Bound    nfs-0001  2Gi        RWO,RWX
claim-mysql Bound    nfs-0002  5Gi        RWO,RWX
master $ [REDACTED]
```

Katacoda Overview & Solutions **Search** **Your Profile** **Log Out**

SWAD: plataforma de apoyo **practica_6_CPD_v1.pdf**

Elena
Cantero
Molina
@elenacm
España

Utilizamos los volúmenes

Running Stateful Services on Kubernetes

Task

Use the command below to view the definition of the Pods.

```
cat pod-mysql.yaml pod-www.yaml ✓
```

You can see the status of the Pods starting using `kubectl get pods`.

If a Persistent Volume Claim is not assigned to a Persistent Volume, then the Pod will be in `Pending` mode until it becomes available. In the next step, we'll read/write data to the volume.

CONTINUE

Terminal

```
- 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-mysql
master $ kubectl get pods
NAME     READY   STATUS      RESTARTS   AGE
mysql   0/1     ContainerCreating   0          5s
www     0/1     ContainerCreating   0          3s
master $ kubectl get pods
NAME     READY   STATUS      RESTARTS   AGE
mysql   1/1     Running     0          16s
www     0/1     ContainerCreating   0          14s
master $ kubectl get pods
NAME     READY   STATUS      RESTARTS   AGE
mysql   1/1     Running     0          23s
www     1/1     Running     0          21s
master $ [REDACTED]
```

Katacoda Overview & Solutions **Search** **Your Profile** **Log Out**

SWAD: plataforma de apoyo **practica_6_CPD_v1.pdf**

Elena
Cantero
Molina
@elenacm
España

Borramos y reactivamos otro Pod para verificar el almacenamiento persistente.

The screenshot shows a Katacoda terminal session titled "Running Stateful Services on Kubernetes". The terminal window displays a series of commands being run on a master node (IP 10.32.0.6). The commands include:

```

Terminal + 10.32.0.6
master $ curl $ip
Hello World
master $ docker exec -it nfs-server bash -c "echo 'Hello NFS World' > /exports/data-0001/index.html"
master $ curl $ip
Hello NFS World
master $ kubectl delete pod www
pod "www" deleted
kubectl create -f pod-www2.yaml
ip=$(kubectl get pod www2 -o yaml | grep podIP | awk '{split($0,a,""); print a[2]}'); curl $ip
curl: try 'curl --help' or 'curl --manual' for more information
master $ kubectl create -f pod-www2.yaml
pod/www2 created
master $ ip=$(kubectl get pod www2 -o yaml | grep podIP | awk '{split($0,a,""); print a[2]}'); curl $ip
curl: try 'curl --help' or 'curl --manual' for more information
Error from server (NotFound): pods "www" not found
master $ kubectl create -f pod-www2.yaml
Error from server (AlreadyExists): error when creating "pod-www2.yaml"
master $ ip=$(kubectl get pod www2 -o yaml | grep podIP | awk '{split($0,a,""); print a[2]}'); curl $ip
Hello NFS World
master $ 

```

The terminal also shows a message about curl help and errors related to pod creation.

II.g) Lanzamos “Deploying a service from source onto Kubernetes”

Creamos un contenedor Docker y a lanzarlo dentro de Kubernetes, definiendo límites de uso de CPU y memoria.

The screenshot shows a Katacoda terminal session titled "Deploying a service from source onto Kubernetes". The terminal window displays a series of commands being run on a master node (IP 10.32.0.6). The commands include:

```

Terminal + Your Interactive Bash Terminal.
master $ export PATH=$PATH:~/bin/
>&& git clone https://github.com/datawire/hello-webapp.git \
>&& export FORGE_SETUP_IMAGE=registry.hub.docker.com/datawire/forge-setup-test-katacoda:1 \
>&& cat /proc/sys/kernel/random/uuid > hello-webapp/uuid.txt \
>&& export LANG=C.UTF-8 && export LC_ALL=C.UTF-8
Cloning into 'hello-webapp'...
remote: Enumerating objects: 41, done.
remote: Total 41 (delta 0), reused 0 (delta 0), pack-reused 41
Unpacking objects: 100% (41/41), done.
Checking connectivity... done.
master $ cd hello-webapp
master $ cat Dockerfile
# Run server
FROM alpine:3.5
RUN apk add --no-cache python py2-pip py2-gevent
COPY requirements.txt .
RUN pip install -r requirements.txt
COPY . /app
WORKDIR /app
EXPOSE 8080
ENTRYPOINT ["python"]
CMD ["app.py"]
master $ 

```

The terminal shows the execution of a Dockerfile to build a container named "hello-webapp" and its deployment to Kubernetes.

Deploying a service from source onto Kubernetes

Step 2 of 6

container, from the operating system (Alpine Linux) to the runtime environment (Python) to the actual app (hello world).

Let's run the container:

```
docker run -d -p 80:8080 hello-webapp:v1
```

(This command runs the container, and maps the container's port 8080 to the hosts port 80.) We can verify that the webapp is running successfully:

```
curl host01
```

(We're now sending an HTTP request to port 80 on host01, which then maps to the 8080 in the container.)

CONTINUE

Terminal

```
Step 6/9 : WORKDIR /app
--> Running in 831a8d7ddda9
Removing intermediate container 831a8d7ddda9
--> dd272079f7df
Step 7/9 : EXPOSE 8080
--> Running in 9ec3532c146e
Removing intermediate container 9ec3532c146e
--> 2210f9f73e21
Step 8/9 : ENTRYPOINT ["python"]
--> Running in 7c9b94fb96d1
Removing intermediate container 7c9b94fb96d1
--> 785c2e1a8f18
Step 9/9 : CMD ["app.py"]
--> Running in 0e1711166c9b
Removing intermediate container 0e1711166c9b
--> 93274e855948
Successfully built 93274e855948
Successfully tagged hello-webapp:v1
master $ docker run -d -p 80:8080 hello-webapp:v1
786dc9382e6b8fb065ff852277200736afdc527e206397736ffacc20bf1351a2
master $ curl host01
Hello World! (up 0:00:01)
master $ curl host01
Hello World! (up 0:00:03)
master $
```

Katacoda Overview & Solutions **Search** **Your Profile** **Log Out**

SWAD: plataforma de apoyo **swad.ugr.es/es** Universidad de Granada

Elena
Cantero
Molina

@elenacm
España

Utilizamos un registro de contenedores.

Deploying a service from source onto Kubernetes

Step 4 of 6

environment variable with the command:

```
export REGISTRY=2886795313-5000-  
ollie09.environments.katacoda.com
```

In order to push the Docker Image to the registry, we need to create a tag for the Docker image that contains our Docker repository name.

```
docker tag hello-webapp:v1  
$REGISTRY/hello-webapp:v1
```

Then, we can push the image we just created to the Docker Registry:

```
docker push $REGISTRY/hello-webapp:v1
```

CONTINUE

Terminal

```
spec:
  containers:
    - image: IMAGE_URL
      imagePullPolicy: IfNotPresent
      name: hello-webapp
      resources:
        limits:
          memory: 0.25G
          cpu: 0.25
        terminationMessagePath: /dev/termination-log
      dnsPolicy: ClusterFirst
      restartPolicy: Always
      securityContext: {}
      terminationGracePeriodSeconds: 30
master $ export REGISTRY=2886795313-5000-ollie09.environments.katacoda.com
master $ docker tag hello-webapp:v1 $REGISTRY/hello-webapp:v1
master $ docker push $REGISTRY/hello-webapp:v1
The push refers to repository [2886795313-5000-ollie09.environments.5eb047be2402: Pushed
f35eb8d36fcfd: Pushed
70bb7dc873b3: Pushed
8672b2c8a765: Pushed
f566c57e6f2d: Pushed
v1: digest: sha256:1be8f8a6481f7c4d6d74f85a06e268532f8ebd70a6204f71b
master $
```

Katacoda Overview & Solutions **Search** **Your Profile** **Log Out**

SWAD: plataforma de apoyo **swad.ugr.es/es** Universidad de Granada

Elena
Cantero
Molina

@elenacm
España

```

Actividades Google Chrome -> Deploying a service from source onto Kubernetes | SWAD:plataforma de apoyo... | practica_6_CPD_v1.pdf + 
vie nov 15, 22:50:57
katacoda.com/courses/kubernetes/deploy-service-from-source

Katacoda Overview & Solutions Search Your Profile Log Out

Terminal +
containers:
  - image: 2886795313-5000-ollie09.environments.katacoda.com/hello-webapp:v1
    imagePullPolicy: IfNotPresent
    name: hello-webapp
    resources:
      limits:
        memory: 0.25G
        cpu: 0.25
      terminationMessagePath: /dev/termination-log
    dnsPolicy: ClusterFirst
    restartPolicy: Always
    securityContext: {}
    terminationGracePeriodSeconds: 30
master $ scp -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no root@host01:/root/.kube/config ~/ kube/ 
Warning: Permanently added 'host01' (ECDSA) to the list of known hosts.
config
master $ kubectl apply -f deployment.yaml
service/hello-webapp created
deployment.extensions/hello-webapp created
master $ kubectl get services
NAME      TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)
hello-webapp   NodePort   10.107.241.199   <none>        80:30902/TCP
kubernetes   ClusterIP  10.96.0.1     <none>        443/TCP
master $ 

```

Now, we can run actually get the service running:

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

We're telling Kubernetes to actually process the information in the manifest.

We can see the services running:

```
kubectl get services ✓
```

Accessing the cluster

Now, we want to send an HTTP request to the service. Most Kubernetes services are not exposed to the Internet, for obvious reasons. So we need to be able to access services that are running in the cluster directly.

```

Actividades Google Chrome -> Deploying a service from source onto Kubernetes | SWAD:plataforma de apoyo... | practica_6_CPD_v1.pdf + 
vie nov 15, 22:52:21
katacoda.com/courses/kubernetes/deploy-service-from-source

Katacoda Overview & Solutions Search Your Profile Log Out

Terminal +
securityContext: {}
terminationGracePeriodSeconds: 30
master $ scp -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no root@host01:/root/.kube/config ~/ kube/
Warning: Permanently added 'host01' (ECDSA) to the list of known hosts.
config
master $ kubectl apply -f deployment.yaml
service/hello-webapp created
deployment.extensions/hello-webapp created
master $ kubectl get services
NAME      TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)
hello-webapp   NodePort   10.107.241.199   <none>        80:30902/TCP
kubernetes   ClusterIP  10.96.0.1     <none>        443/TCP
master $ 

```

We've just performed four main things 1) created a container 2) created the Kubernetes manifest 3) pushed the container to a registry and 4) finally told Kubernetes all about these pieces with an updated manifest.

So, this was a bunch of steps, and if you have to do this over-and-over again (as you would for development), this would quickly grow tedious. Let's take a quick look at how we might script these steps so we could create a more useable workflow.

CONTINUE

Utilizar forge para automatizar el lanzamiento.

Actividades Google Chrome ▾

vie nov 15, 22:53:01

Deploying a service from source onto Kubernetes

Step 6 of 6

services onto 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:
2886795313-5000-
ollie09.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:

forge deploy ✓

Katacoda Overview & Solutions Search Your Profile Log Out

Terminal +

```
hello-webapp-649ff8888-vsbcv 1/1 Running 0 53s
master $ kubectl get svc
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
hello-webapp NodePort 10.107.241.199 <none> 80:30902/TCP 55s
kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 5m51s
master $ export PORT=$(kubectl get svc hello-webapp -o yaml | go-template='{{range .spec.ports}}{{if .nodePort}}{{.nodePort}}{{"\n"}}{{end}}{{end}}')
master $ curl host01:$PORT
Hello World! (up 0:00:57)
master $ sed -i -e 's>Hello World!/Hello Hacker News!!!/' app.py
master $ forge setup
== Checking Kubernetes Setup ==

kubectl version --short
Client Version: v1.14.0
Server Version: v1.14.0
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 7m3s
1 tasks run, 0 errors

== Setting up Docker ==
Registry type (one of ecr, gcr, generic)[generic]: ↴
```

SWAD: plataforma de apoyo x swad.ugr.es/es Universidad de Granada Elena Cantero Molina @elenacm España

Actividades Google Chrome ▾

vie nov 15, 22:54:48

Deploying a service from source onto Kubernetes

Step 6 of 6

services onto 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:
2886795313-5000-
ollie09.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:

forge deploy ✓

Katacoda Overview & Solutions Search Your Profile Log Out

Terminal +

```
== Done ==
master $ forge deploy
docker build /root/hello-webapp/ -f /root/hello-webapp/Dockerfile -t 2886795313-5000-ollie09.environments.katacoda.com/root/hello-webapp:0116b24806730d21c776856ce58af75a9907b124.sha
Sending build context to Docker daemon 137.2KB
Step 1/9 : FROM alpine:3.5
--> f80194ae2e0c
Step 2/9 : RUN apk add --no-cache python py2-pip py2-gevent
--> Using cache
--> 86a7705b6443
Step 3/9 : COPY requirements.txt .
--> Using cache
--> c8f5b4a84a9e
Step 4/9 : RUN pip install -r requirements.txt
--> Using cache
--> f83e8accf661
Step 5/9 : COPY . /app
--> 27d4e78a3035
Step 6/9 : WORKDIR /app
--> Running in 2e413a2b4246
Removing intermediate container 2e413a2b4246
--> c18dc14f4137
Step 7/9 : EXPOSE 8080
--> Running in f4fc24648ca6
Removing intermediate container f4fc24648ca6
```

SWAD: plataforma de apoyo x swad.ugr.es/es Universidad de Granada Elena Cantero Molina @elenacm España

Actividades Google Chrome ▾

vie nov 15, 22:55:26

Deploying a service from source onto Kubernetes

Step 6 of 6

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

Now, 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 into Kubernetes.

CONTINUE

Katacoda Overview & Solutions Search Your Profile Log Out

Terminal +

```
48 tasks run, 0 errors
built: Dockerfile
pushed: hello-webapp:0116b24806730d21c776856ce58af75a9907b124.sha
rendered: service/hello-webapp, deployment.extensions/hello-webapp
deployed: hello-webapp
master $ kubectl get pods
NAME READY STATUS RESTARTS AGE
hello-webapp-649ff8888-vsbcv 1/1 Terminating 0 4m26s
hello-webapp-84b994cc6-tmv54 1/1 Running 0 29s
master $ export PORT=$(kubectl get svc hello-webapp -o yaml | go-template='{{range .spec.ports}}{{if .nodePort}}{{.nodePort}}{{"\n"}}{{end}}{{end}}')
master $ curl host01:$PORT
Hello Hacker News!!! (up 0:00:30)
master $ curl host01:$PORT
Hello Hacker News!!! (up 0:00:33)
master $ curl host01:$PORT
Hello Hacker News!!! (up 0:00:39)
master $ curl host01:$PORT
Hello Hacker News!!! (up 0:00:41)
master $ curl host01:$PORT
Hello Hacker News!!! (up 0:00:42)
master $
```

SWAD: plataforma de apoyo x swad.ugr.es/es Universidad de Granada Elena Cantero Molina @elenacm España

