

Universidad de Granada

Escuela Técnica Superior de Ingeniería Informática y
Telecomunicaciones

Grado en Ingeniería Informática

Centros de Procesamiento de Datos

Práctica 7 Kubernetes 101

Javier Martín Gómez

2018/2019

II.a) Captura del Dashboard.

The screenshot shows the Kubernetes Dashboard in a web browser. The left sidebar contains a menu with options: Cluster, Namespaces, Nodes, Persistent Volumes, Roles, and Storage Classes. The main area displays the 'Cluster' overview, which includes a table of Namespaces and a table of Nodes.

Name	Labels	Status	Age
default	-	Active	13 minutes
kube-public	-	Active	13 minutes
kube-system	-	Active	13 minutes

Name	Labels	Ready	CPU requests (cores)	CPU limits (cores)	Memory requests (bytes)	Memory limits (bytes)	Age
node01	beta.kubernetes.io...	True	0.02 (0.50%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	11 minutes
master	beta.kubernetes.io...	True	0.77 (19.25%)	0 (0.00%)	140 Mi (7.00%)	340 Mi (17.00%)	13 minutes

II.b) Captura donde aparece el puerto modificado.

The screenshot shows a Katacoda course titled 'Start containers using Kubectl'. The main content area displays the 'Step 5 - Scale Containers' section, which includes a description of scaling a deployment and a task to scale the deployment to 3 replicas. A terminal window on the right shows the command `kubectl get pods` and its output, which lists four pods in a 'Running' state.

Step 5 - Scale Containers

With our deployment running we can now use *kubectl* to scale the number of replicas.

Scaling the deployment will request Kubernetes to launch additional Pods. These Pods will then automatically be load balanced using the exposed Service.

Task

The command *kubectl scale* allows us to adjust the number of Pods running for a particular deployment or replication controller.

```
kubectl scale --replicas=3 deployment http
```

Listing all the pods, you should see three running for the

```
$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
http-7b77c4cd66-2qtf5               1/1     Running   0           50s
http-7b77c4cd66-ct6jh               1/1     Running   0           50s
http-7b77c4cd66-xw9q5               1/1     Running   0           4m
httpexposed-5c4cf8b7d8-ggqkb        1/1     Running   0           2m
```

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

The screenshot shows the Katacoda interface for the 'Deploy Containers Using YAML' exercise. The left panel displays instructions for Step 3 of 3, including commands like `replicas: 4`, `kubectl apply -f deployment.yaml`, `kubectl get deployment`, `kubectl get pods`, and `curl host01:30080`. The right panel shows the YAML definitions for a Deployment and a Service.

```
deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: webapp1-deploy
spec:
  replicas: 4
  selector:
    matchLabels:
      app: webapp1
  template:
    metadata:
      labels:
        app: webapp1
    spec:
      containers:
      - name: webapp1
        image: nginx:1.15.10
        ports:
        - containerPort: 80
```

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

The terminal output shows the deployment configuration and the resulting pods:

```
deployment.extensions/webapp1 configured
$ kubectl get deployment
NAME          DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
webapp1        4          4          4             2           2m
$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
webapp1-6c66d9cb4f-dvrxg            1/1     Running   0           8s
webapp1-6c66d9cb4f-fv7qk            1/1     Running   0           2m
webapp1-6c66d9cb4f-g728w            1/1     Running   0           8s
webapp1-6c66d9cb4f-wblqr            1/1     Running   0           8s
```

II.d) Captura donde aparece personalizado el NodePort.

The screenshot shows the Katacoda interface for the 'Networking Introduction' exercise. The left panel displays instructions for Step 3 of 5, including commands like `kubectl apply -f nodeport.yaml`, `cat nodeport.yaml`, `kubectl get svc`, `kubectl describe svc/webapp1-nodeport-svc`, and `curl 172.17.0.120:30080`. The right panel shows the terminal output for these commands.

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

The terminal output shows the service definition and the resulting curl command:

```
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":{"name":"webapp1-nodeport-svc","namespace":"default...
Selector:      app=webapp1-nodeport
Type:          NodePort
IP:            10.109.135.114
Port:          <unset> 80/TCP
TargetPort:    80/TCP
NodePort:      <unset> 30080/TCP
Endpoints:     10.32.0.8:80,10.32.0.9:80
Session Affinity: None
External Traffic Policy: Cluster
Events:        <none>
master $ curl 172.17.0.120:30080
<h1>This request was processed by host: webapp1-nodeport-deployment-785989576b-4g4h9</h1>
master $ curl 172.17.0.120:30080
<h1>This request was processed by host: webapp1-nodeport-deployment-785989576b-jk977</h1>
master $
```

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

Aplicaciones Lugares Google Chrome mié, 21 de nov, 12:32

Create Ingress Routing | Katacoda

Step 4 of 4

```
curl -H "Host: my.kubernetes.example" 172.17.0.83/webapp1 ✓
```

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

```
curl -H "Host: my.kubernetes.example" 172.17.0.83/webapp2 ✓
```

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

```
curl -H "Host: my.kubernetes.example" 172.17.0.83 ✓
```

SUMMARY

Terminal

```
- path: /webapp1
  backend:
    serviceName: webapp1-svc
    servicePort: 80
- path: /webapp2
  backend:
    serviceName: webapp2-svc
    servicePort: 80
- 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           5s
master $ curl -H "Host: my.kubernetes.example" 172.17.0.83/webapp1
<h1>This request was processed by host: webapp1-7d67d68676-6shk5</h1>
master $ curl -H "Host: my.kubernetes.example" 172.17.0.83/webapp2
<h1>This request was processed by host: webapp2-64d4844b78-nsk44</h1>
master $ curl -H "Host: my.kubernetes.example" 172.17.0.83
<h1>This request was processed by host: webapp3-5b8ff7484d-qn2fs</h1>
master $
```

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

Aplicaciones Lugares Google Chrome mié, 21 de nov, 12:42

Running Stateful Services on Kubernetes | Katacoda

Step 5 of 6

```
pod1: /bin/sh -c (sleep 30s && echo 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 ✓
```

CONTINUE

Terminal

```
master $ curl $ip
Hello NFS World
master $
```

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

Deploying a service from source onto Kubernetes

Step 6 of 6

As previously, obtain the NodePort assigned to our deployment.

```
export PORT=$(kubectl get svc hello-webapp -o 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.

SUMMARY

Terminal

```
on!
| warning: 'collections.OrderedDict object' has no attribute 'protocol' (this will become an error soon)
| warning: 'collections.OrderedDict object' has no attribute 'port' (this will become an error soon)
| 48 tasks run, 0 errors
|
| built: Dockerfile
| pushed: hello-webapp:ba3e7b7b6db0cafd441aa27ce0cd8b8442538b1.sha
| rendered: service/hello-webapp, deployment/hello-webapp
| deployed: hello-webapp
|
> kubectl get pods
NAME                                READY    STATUS    RESTARTS   AGE
hello-webapp-5c85d6664c-vyb6m        1/1      Terminating    0           6m
hello-webapp-7cbcf48c44-nn6kf        1/1      Running         0           7s
|
| .nodePort}}{{.nodePort}}{{"\n"}}{{end}}{{end}}'o go-template='{{range.spec.ports}}{{if
|
> curl host01:$PORT
Hello Hacker News!!! (up 0:00:09)
|
>
```