

# DevOps Sesión 7 (2022-03-02) k8s

## Documentación relacionada

- lpi-devops-study-master → documentación estudio
- 2-Despliegue de Aplicaciones Kubernetes/1-Laboratorio kubernetes Curso-DevOps.txt
- 2-Despliegue de Aplicaciones Kubernetes/1-Laboratorios Kubernetes 2020.pdf
- 2-Despliegue de Aplicaciones Kubernetes/2-Laboratorios basicos kubernetes .pdf

## Clase

- `~/.kube/config`: credenciales para conectar con el cluster
- Lens, the kubernetes IDE → <https://k8slens.dev/>
- k8s certificación
  - DCA: administrador (tipo test)
  - DCK
  - CKD
  - CKS

## deployment

- 2-Despliegue de Aplicaciones Kubernetes/1-Laboratorios Kubernetes 2020.pdf pàg 37
- 2-Despliegue de Aplicaciones Kubernetes/kubernetes-curso/deployment

- <https://kubernetes.io/es/docs/concepts/workloads/controllers/deployment/>
- desplegamos:

```
kubectl apply -f /vagrant/kubernetes-curso/deployment/helloworld.yml --record
kubectl get deployment,rs,pod
```

- Debes indicar el parámetro **--record** para registrar el comando ejecutado en la anotación de recurso [kubernetes.io/change-cause](https://kubernetes.io/es/docs/concepts/workloads/controllers/deployment/). Esto es útil para futuras introspecciones, por ejemplo para comprobar qué comando se ha ejecutado en cada revisión del Deployment. To be deprecated.

- ```
apiVersion: apps/v1 # Usa apps/v1beta2 para versiones anteriores a 1.9.0
kind: Deployment
metadata:
  name: helloworld-deployment
spec:
  selector: #permite seleccionar un conjunto de objetos que cumplan las
    condicione
  matchLabels:
```

```
  app: helloworld
  replicas: 3
  template:
    metadata:
      labels:
        app: helloworld
    spec:
      containers:
        - name: k8s-demo
          image: wardviaene/k8s-demo
          ports:
            - name: nodejs-port
              containerPort: 3000
```

- expongo el servicio:

```
kubectl expose deployment helloworld-deployment --type=NodePort --
name=helloworld-deployment-service
kubectl describe service helloworld-deployment-service
```

- <http://10.0.0.11:<NodePort>>

- actualizo la versión:

```
kubectl set image deployment/helloworld-deployment k8s-demo=wardviaene/k8s-
demo:2 --record
```

- **k8s-demo** es el nombre del contenedor que quiero actualizar (así especificado en el yaml)

- mostrar history:

```
kubectl rollout history deployment helloworld-deployment
```

- hacer rollout (en este caso un **undo**, por defecto k8s almacena 10 versiones):

```
kubectl rollout undo deployment helloworld-deployment
```

- `kubectl rollout status deployment helloworld-deployment`

- hacer rollout a otra versión:

```
kubectl rollout undo deployment helloworld-deployment --to-revision=3
```

- el número es el que aparece en el **history**
- también: `kubectl annotate deployment.v1.apps/nginx-deployment kubernetes.io/change-cause=<image updated to 1.9.1>`

- escalamos:

```
kubectl scale deployment helloworld-deployment --replicas=5
kubectl edit deployments.apps helloworld-deployment
kubectl scale --replicas=4 -f /vagrant/kubernetes-
curso/deployment/helloworld.yml
```

- eliminamos:

```
kubectl delete deployment helloworld-deployment
kubectl delete -f /vagrant/kubernetes-curso/deployment/helloworld.yml
kubectl delete service helloworld-deployment-service
```

```
kubectl get service,deployment
```

## estrategias de update:

2-Despliegue de Aplicaciones Kubernetes/Seminario kubernetes Troubleshooting Deployments Applications.pdf pag 41 2-Despliegue de Aplicaciones Kubernetes/Laboratorio-deployment-strategies/deployment-blue-green

- <https://kubernetes.io/es/docs/concepts/workloads/controllers/deployment/#estrategia>
- rollingupdate
  - actualización continua. Se crean los de la nueva versión, y cuando estén listos, va eliminando los viejos.
  - maxSurge:
  - maxUnavailable
- recreate
  - los PODs se eliminan antes de que los nuevos se creen.
  - se cae el servicio y se levanta.
  - para aplicaciones sin estado
- Canary:
  - despliegue progresivo, sustituyendo 1 a 1 y siguiendo si todo va bien
- Blue-Green
  - se despliegan las 2 versiones y el servicio cambia de una a otra
  - se elimina la versión anterior
- A/B
  - estrategia de pruebas de diferentes prototipos
- HPA = Horizontal Pod autoscaler
- defines max y mínimos y hasta donde puede ampliar los PODs

## namespaces

2-Despliegue de Aplicaciones Kubernetes/1-Laboratorios Kubernetes 2020.pdf, pág 176

- Los namespaces (espacios de nombres) en Kubernetes permiten establecer un nivel adicional de separación entre los contenedores que comparten los recursos de un clúster.
  - colisión de nombres
  - limitación de recursos: memoria, cpu, objetos
  - los servicios se ven a través de los namespaces (autodescubrimiento DNS)
    - con **network policies** podemos restringir esa comunicación
  - no cambio nombre namespace en caliente
  - cluster virtual sobre cluster real

```
kubectl get namespaces
kubectl get ns # idem anterior
kubectl get pod --namespace kube-system
kubectl get pod -n kube-system -o wide # idem anterior
kubectl get pods --all-namespaces

kubectl create ns formacion
```

```
kubectl describe ns formacion # muestra cuotas
kubectl run nginx --image=nginx -n formacion
kubectl get pods
kubectl get pods --all-namespaces

kubectl config get-contexts # saber namespace por defecto
kubectl config set-context --current --namespace=formacion

kubectl delete ns formacion
```

## descubrimiento (Kubernetes service discovery)

2-Despliegue de Aplicaciones Kubernetes/1-Laboratorios Kubernetes 2020.pdf, pág 132

- Cada vez que se crea un nuevo servicio se crea un registro de tipo A con el nombre **<servicio>.<namespace>.svc.cluster.local**
- Para cada puerto nombrado se crea un registro SRV del tipo **\_nombre-puerto.\_nombre-protocolo.my-svc.my-namespace.svc.cluster.local** que resuelve el número del puerto y al CNAME: **servicio.namespace.svc.cluster.local**
- Para cada pod creado con dirección IP 1.2.3.4, se crea un registro A de la forma **1-2-3-4.default.pod.cluster.local**.
- pods en **kube-system coredns**

```
kubectl get rs.apps -n kube-system
kubectl get pod -o wide -n kube-system
kubectl describe pod coredns -n kube-system
kubectl get service -o wide -n kube-system # DNS del cluster
```

## lab

```
kubectl run -i --tty centos1 --image centos:centos7 --restart=Never -- bash
# en el pod:
yum install dns-utils
nslookup...
```

## Healthcheck

2-Despliegue de Aplicaciones Kubernetes/1-Laboratorios Kubernetes 2020.pdf, pág 84

- Los Healthchecks son un mecanismo fundamental para cargas productivas. Es el principal mecanismo por el cual Kubernetes va a saber si nuestros Pods están funcionando correctamente o no.
- por CLI (o comando) o por HTTPGets
- 2 tipos «básicos»:
  - ReadinessProbe: indica si está listo para recibir tráfico
  - LivenessProbe: indica que ese pod está funcionando

- Kubernetes:
  - Si Readiness falla → detiene el tráfico al POD
  - Si Liveness falla → reinicia el pod
  - Si Readiness funciona → restablece el tráfico hacía el pod

```

• ...
  containers:
  - name: jboss-lab
    image: docker.example.com/jboss:1
    imagePullPolicy: IfNotPresent
    livenessProbe:
      initialDelaySeconds: 30
      periodSeconds: 15
      exec:
        command:
        - /bin/sh
        - -c
        - /opt/jboss/wildfly/bin/jboss-cli.sh --connect --commands=ls | grep
'server-state=running'
    readinessProbe:
      initialDelaySeconds: 30
      periodSeconds: 5
      exec:
        command:
        - /bin/sh
        - -c
        - /opt/jboss/wildfly/bin/jboss-cli.sh --connect --commands=ls | grep
'server-state=running'
    livenessProbe:
      httpGet:
        path: /status
        port: 8080
      initialDelaySeconds: 30
      timeoutSeconds: 10

```

- InitialDelaySeconds: espera X a que arranque
- timeoutSeconds: cada X comprueba de nuevo
- failure Threshold: número de veces para darlo por malo

## lab

2-Despliegue de Aplicaciones Kubernetes/k8-for-devs-master/healthchecks

```

• apiVersion: apps/v1
  kind: Deployment
  metadata:
    name: deployment-lab
  labels:
    app: nginx
  spec:
    replicas: 3
    selector:

```

```
matchLabels:
  app: nginx
template:
  metadata:
    labels:
      app: nginx
  spec:
    containers:
      - name: nginx
        image: nginx
        ports:
          - containerPort: 80
        livenessProbe:
          httpGet:
            path: /
            port: 80
          initialDelaySeconds: 10
          timeoutSeconds: 10
```

- `kubectl apply -f /vagrant/k8-for-devs-master/healthchecks/healcheck.yaml`  
`kubectl get pod`

```
kubectl get deployment.apps/deployment-lab -o yaml
```

```
kubectl describe deployment.apps/deployment-lab |grep -i Liveness
# Liveness: http-get http://:80/ delay=30s timeout=10s period=10s
#success=1 #failure=3
```

## lab

2-Despliegue de Aplicaciones Kubernetes/2-Laboratorios basicos kubernetes .pdf pág 7

- `kubectl apply -f /vagrant/kubernetes-labs2/healthz/pod.yaml`  
`kubectl describe pod hc`  
`kubectl apply -f /vagrant/kubernetes-labs2/healthz/badpod.yaml`  
`kubectl describe pod badpod`  
`kubectl get pods # mirar reinicios`

## lab

```
kubectl create ns miercoles2
kubectl config set-context --current --namespace=miercoles2
kubectl run --image nginx --port=80 aplical
kubectl expose pod aplical --type=NodePort --name aplical-service
kubectl describe service aplical-service
```

## Extra

- `kubectl get events`
- `kubectl logs --previous=true`: mostrar logs de pods/contenedores anteriores
  - centralizar

From:

<https://miguelangel.torresegea.es/wiki/> - **miguel angel torres egea**

Permanent link:

<https://miguelangel.torresegea.es/wiki/info:cursos:pue:devops2022:s7?rev=1646840077>

Last update: **09/03/2022 07:34**

