

Formación en DevOps

# Working Time Kubernetes (K8s)



## Working Time #3

### ≡ Kubernetes

Es hora de que pongas en práctica todo lo aprendido. 😊

✅ **Entregable:** Adjunta en la plataforma un screenshot del resultado final de cada laboratorio 🙌

### Índice

<b>Laboratorio Práctico 1</b>	2
Paso 1: Configuración del Entorno	3
Paso 2: Creación de un Pod Simple	4
Paso 3: Uso de ConfigMaps	4
Paso 4: Uso de Secrets	5
Paso 5: Limpieza	6
<b>Laboratorio Práctico 2</b>	7
Paso 1: Configuración del Entorno	7
Paso 2: Despliegue de Aplicaciones	8
Paso 3: Verificación de Comunicación	8
Paso 5: Verificación de Restricciones	10
Paso 6: Limpieza:	10
<b>Laboratorio Práctico 3</b>	10
Paso 1: Configuración del Entorno	11
Paso 2: Despliegue de Aplicación con Volumen Persistente	11
Paso 3: Verificación de Persistencia	12
Paso 4: Limpieza	13

# Laboratorio Práctico 1

## Introducción

Manejo de Pods, ConfigMaps y Secrets en Kubernetes

### ¿Qué practicaremos?

1. Crear y gestionar Pods en Kubernetes.
2. Utilizar ConfigMaps para gestionar la configuración de las aplicaciones.
3. Utilizar Secrets para gestionar información sensible de manera segura.

### Recursos:

<https://minikube.sigs.k8s.io/docs/start/>

<https://kubernetes.io/docs/tasks/tools/install-kubectl/>

¡Manos a la Obra! 💪

## ● Paso 1: Configuración del Entorno

Asegúrate de tener **kubectl** y **minikube** instalados en tu máquina local.

# Instalar minikube (<https://minikube.sigs.k8s.io/docs/start/>)

# Instalar kubectl (<https://kubernetes.io/docs/tasks/tools/install-kubectl/>)

Inicia el clúster de minikube:

```
minikube start
```

## ● Paso 2: Creación de un Pod Simple

Crea un archivo YAML llamado **simple-pod.yaml** para definir un Pod básico.

```
apiVersion: v1
kind: Pod
metadata:
  name: simple-pod
spec:
  containers:
    - name: nginx-container
      image: nginx
```

Aplica el Pod al clúster:

```
kubectl apply -f simple-pod.yaml
```

Verifica que el Pod está en ejecución:

```
kubectl get pods
```

## ● Paso 3: Uso de ConfigMaps

Crea un ConfigMap para almacenar la configuración de la aplicación.

**config-map.yaml**.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config
data:
  DATABASE_URL: "mongodb://mongo-server:27017/mydb"
  API_KEY: "my-api-key"
```

Aplica el ConfigMap al clúster:

```
kubectl apply -f config-map.yaml
```

Actualiza el archivo **simple-pod.yaml** para usar valores del ConfigMap.

```
apiVersion: v1
kind: Pod
metadata:
  name: simple-pod
spec:
  containers:
    - name: nginx-container
      image: nginx
      env:
        - name: DATABASE_URL
          valueFrom:
            configMapKeyRef:
              name: app-config
              key: DATABASE_URL
        - name: API_KEY
          valueFrom:
            configMapKeyRef:
              name: app-config
              key: API_KEY
```

Aplica la actualización al clúster:

```
kubectl apply -f simple-pod.yaml
```

## Paso 4: Uso de Secrets

Crea un Secret para almacenar información sensible.

my-secret.yaml.

```
apiVersion: v1
kind: Secret
metadata:
  name: my-secret
type: Opaque
data:
  username: YWRtaW4=
  password: cGFzc3dvcmQ=
```

Aplica el Secret al clúster:

```
kubectl apply -f my-secret.yaml
```

Actualiza el archivo `simple-pod.yaml` para usar valores del Secret.

```
apiVersion: v1
kind: Pod
metadata:
  name: simple-pod
spec:
  containers:
    - name: nginx-container
      image: nginx
      env:
        - name: DATABASE_URL
          valueFrom:
            configMapKeyRef:
              name: app-config
              key: DATABASE_URL
        - name: API_KEY
          valueFrom:
            configMapKeyRef:
              name: app-config
              key: API_KEY
        - name: DB_USERNAME
          valueFrom:
            secretKeyRef:
              name: my-secret
              key: username
        - name: DB_PASSWORD
          valueFrom:
            secretKeyRef:
              name: my-secret
              key: password
```

Aplica la actualización al clúster:

```
kubectl apply -f simple-pod.yaml
```

## Paso 5: Limpieza

Cuando hayas terminado, asegúrate de limpiar los recursos creados:

```
kubectl delete pod simple-pod
kubectl delete configmap app-config
kubectl delete secret my-secret
```

# Laboratorio Práctico 2

## Introducción

Políticas de Networking en Kubernetes

### ¿Qué practicaremos?

1. Crear un clúster de Kubernetes.
2. Desplegar dos aplicaciones sencillas.
3. Configurar una política de red para restringir la comunicación entre las aplicaciones.

### Recursos:

<https://minikube.sigs.k8s.io/docs/start/>

<https://kubernetes.io/docs/tasks/tools/install-kubectl/>

¡Manos a la Obra! 🛠️

## 🟡 Paso 1: Configuración del Entorno

Asegúrate de tener **kubectl** y **minikube** instalados en tu máquina local.

# Instalar minikube (<https://minikube.sigs.k8s.io/docs/start/>)

# Instalar kubectl (<https://kubernetes.io/docs/tasks/tools/install-kubectl/>)

Inicia el clúster de minikube:

```
minikube start
```

## 🟡 Paso 2: Despliegue de Aplicaciones

Crea dos archivos YAML para desplegar dos aplicaciones sencillas: **app-a.yaml** y **app-b.yaml**.

app-a.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: app-a
spec:
  containers:
  - name: app-a-container
    image: nginx
```

app-b.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: app-b
spec:
  containers:
  - name: app-b-container
    image: nginx
```

Aplica los archivos YAML al clúster:

```
kubectl apply -f app-a.yaml kubectl apply -f app-b.yaml
```

## Paso 3: Verificación de Comunicación

Confirma que las aplicaciones se han desplegado correctamente y pueden comunicarse entre sí.

# Obtener direcciones IP de los Pods

```
kubectl get pods -o wide
```

# Acceder a una shell dentro de los Pods

```
kubectl exec -it app-a -- /bin/sh kubectl exec -it app-b -- /bin/sh
```

Dentro de **app-a** intentar acceder a la aplicación B

```
curl <IP-DE-APP-B>
```

Dentro de **app-b** intentar acceder a la aplicación A

```
curl <IP-DE-APP-A>
```



## ● Paso 4: Configuración de Política de Networking

Crear una política de red básica para restringir la comunicación entre **app-a** y **app-b**.  
Crea un archivo **network-policy.yaml**.

**network-policy.yaml**

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: restrict-communication
spec:
  podSelector:
    matchLabels:
      app: app-a
  policyTypes:
    - Ingress
  ingress:
    - from:
      - podSelector:
          matchLabels:
            app: app-b
```

Aplica la política al clúster:

```
kubectl apply -f network-policy.yaml
```

## ● Paso 5: Verificación de Restricciones

Confirma que la política de red ha restringido la comunicación entre **app-a** y **app-b**.

```
# Dentro de app-a, intentar acceder a la aplicación B (debería fallar)
curl <IP-DE-APP-B>
```

```
# Dentro de app-b, intentar acceder a la aplicación A (debería fallar)
curl <IP-DE-APP-A>
```

## ● Paso 6: Limpieza

Cuando hayas terminado, asegúrate de limpiar los recursos creados:

```
kubectl delete pod app-a app-b
```

```
kubectl delete networkpolicy restrict-communication
```

## Laboratorio Práctico 3

### Introducción

Volúmenes Persistentes en Kubernetes

Para ejecutar los comandos de este practico se necesita PowerShell.

### ¿Qué practicaremos?

1. Crear un clúster de Kubernetes.
2. Desplegar una aplicación que use un volumen persistente.
3. Verificar la persistencia de los datos en el volumen.

### Recursos:

<https://minikube.sigs.k8s.io/docs/start/>

<https://kubernetes.io/docs/tasks/tools/install-kubectl/>

¡Manos a la Obra! 🛠️

### 🔵 Paso 1: Configuración del Entorno

Asegúrate de tener **kubectl** y **minikube** instalados en tu máquina local.

# Instalar minikube (<https://minikube.sigs.k8s.io/docs/start/>)

# Instalar kubectl (<https://kubernetes.io/docs/tasks/tools/install-kubectl/>)

Inicia el clúster de minikube:

```
minikube start
```

## ● Paso 2: Despliegue de Aplicación con Volumen Persistente

Crea un archivo YAML para desplegar una aplicación con un volumen persistente:  
**app-with-pv.yaml**.

```
apiVersion: v1
kind: Pod
metadata:
  name: app-with-pv
spec:
  containers:
    - name: app-container
      image: mcr.microsoft.com/windows/servercore/iis:windowsservercore-ltsc2019
      volumeMounts:
        - name: data-volume
          mountPath: C:\app-data
  volumes:
    - name: data-volume
      persistentVolumeClaim:
        claimName: data-pvc
```

Crea un archivo YAML para definir un PersistentVolume y un PersistentVolumeClaim:  
**pv-pvc.yaml**.

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: data-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "C:\\data-pv"

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: data-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

Aplica los archivos YAML al clúster:

```
kubectl apply -f pv-pvc.yaml
kubectl apply -f app-with-pv.yaml
```

## Paso 3: Verificación de Persistencia

Confirma que los datos persisten en el volumen incluso después de reiniciar el Pod.

```
# Obtener la dirección IP del Pod
kubectl get pods -o wide
```

```
# Acceder a una shell dentro del Pod
kubectl exec -it app-with-pv -- /bin/sh
```

Dentro del Pod:

```
# Crear un archivo en el volumen
(Si no funciona puede usar Notepad, recuerde la extensión del archivo debe ser esa)
```

```
Add-Content -Path C:\app-data\index.html -Value "Hola, Mundo!"
Exit
```

```
# Reiniciar el Pod
```

```
kubectl delete pod app-with-pv
```

```
# Obtener la dirección IP del nuevo Pod
```

```
kubectl get pods -o wide
```

```
# Acceder a una shell dentro del nuevo Pod
```

```
kubectl exec -it app-with-pv -- /bin/sh
```

Dentro del nuevo Pod:

```
# Verificar que el archivo creado anteriormente sigue presente
```

```
Get-Content C:\app-data\index.html
```

## Paso 4: Limpieza

Cuando hayas terminado, asegúrate de limpiar los recursos creados:

```
kubect1 delete pod app-with-pv kubect1 delete pv,pvc data-pv
```



# ¡Muchas gracias!

Nos vemos en la próxima clase