CONTENIDOS

1 DESPLIEGUE DE APLICACIONES EN CONTENEDORES (2 HORAS)

- Introducción a los contenedores
- Arquitectura de microservicios
- Tecnologías subyacentes y diferencias entre ellas: docker, cri-o, LXC, ...
- Ciclo de vida en el despliegue de aplicaciones con docker

2 INTRO A KUBERNETES (2 HORAS)

- Características, historia, estado actual del proyecto kubernetes (k8s)
- Arquitectura básica de k8s
- Alternativas para instalación simple de k8s: minikube, kubeadm, k3s
- Instalación con minikube
- Instalación y uso de kubectl
- Despliegue de aplicaciones con k8s

3 DESPLIEGUE DE APLICACIONES CON K8S (1:30 HORAS)

- Pods
- ReplicaSet: Tolerancia y escalabilidad
- Deployment: Actualizaciones y despliegues automáticos

4 COMUNICACIÓN ENTRE SERVICIOS Y ACCESO DESDE EL EXTERIOR (1:30 HORAS)

- Services
- DNS
- Ingress
- Ejemplos de uso y despliegues

5 CONFIGURACIÓN DE APLICACIONES (1 HORA)

- Variables de entorno
- ConfigMaps
- Secrets
- Ejemplo de despliegue parametrizado

6 ALMACENAMIENTO (1:30 HORAS)

- Consideraciones sobre el almacenamiento
- PersistentVolume
- PersistentVolumeClaim
- Ejemplo de despliegue con volúmenes

CONTENIDOS

7 OTROS TIPOS DE DESPLIEGUES (1:30 HORAS)

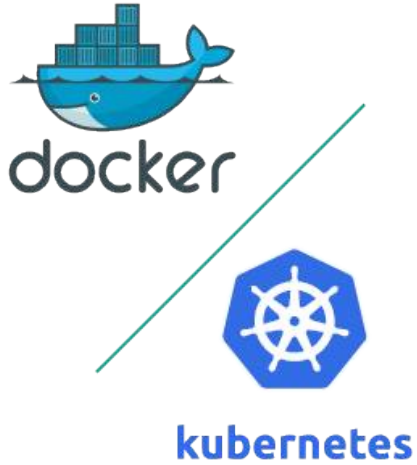
- StatefulSet
- DaemonSet
- AutoScale
- Helm

8 ADMINISTRACIÓN BÁSICA (1 HORA)

- Namespaces
- Usuarios
- RBAC
- Cuotas y límites

9 INSTALACIÓN PASO A PASO (4 HORAS)

- Consideraciones previas: Requisitos hardware, arquitectura física y lógica, entornos y herramientas para el despliegue
- Instalación completa componente a componente en múltiples nodos



MÓDULO 5. CONFIGURACIÓN DE APLICACIONES

Variables de Entorno

```yaml
apiVersion: apps/v1beta1
kind: Deployment
metadata:
 name: mariadb-deployment
 labels:
   app: mariadb
   type: database
spec:
 replicas: 1
 template:
   metadata:
     labels:
       app: mariadb
       type: database
   spec:
     containers:
       - name: mariadb
         image: mariadb
         ports:
           - containerPort: 3306
             name: db-port
         env:
           - name: MYSQL_ROOT_PASSWORD
             value: my-password
```

EJEMPLO 1

Podemos definir un Deployment que defina un contenedor configurado por medio de variables de entorno.

Creamos el despliegue:

```
kubectl create -f mariadb-deployment.yaml
```

O directamente ejecutando:

```
kubectl run mariadb --image=mariadb --env MYSQL_ROOT_PASSWORD=my-password
```

Veamos el pod creado:

```
kubectl get pods -l app=mariadb
```

Y probamos si podemos acceder, introduciendo la contraseña configurada:

```
kubectl exec -it mariadb-deployment-fc75f956-f5zlt -- mysql -u root -p
```

ConfigMap

```
...
     containers:
       - name: mariadb
         image: mariadb
         ports:
           - containerPort: 3306
             name: db-port
         env:
           - name: MYSQL_ROOT_PASSWORD
             valueFrom:
               configMapKeyRef:
                 name: mariadb
                 key: root_password
           - name: MYSQL_USER
             valueFrom:
               configMapKeyRef:
                 name: mariadb
                 key: mysql_usuario
           - name: MYSQL_PASSWORD
             valueFrom:
               configMapKeyRef:
                 name: mariadb
                 key: mysql_password
           - name: MYSQL_DATABASE
             valueFrom:
               configMapKeyRef:
                 name: mariadb
                 key: basededatos
```

EJEMPLO 2

ConfigMap te permite definir un diccionario (clave,valor) para guardar información que puedes utilizar para configurar una aplicación.
Al crear un ConfigMap los valores se pueden indicar desde un directorio, un fichero o un literal.

```
kubectl create cm mariadb --from-literal=root_password=my-password \
                          --from-literal=mysql_usuario=usuario     \
                          --from-literal=mysql_password=password-user \
                          --from-literal=basededatos=test

kubectl get cm
kubectl describe cm mariadb
```

Creamos un deployment indicando los valores guardados en el ConfigMap:

```
kubectl create -f mariadb-deployment-configmap.yaml
kubectl exec -it mariadb-deploy-cm-57f7b9c7d7-ll6pv -- mysql -u usuario -p
```

Secrets

```yaml
apiVersion: apps/v1beta1
kind: Deployment
metadata:
 name: mariadb-deploy-secret
 labels:
   app: mariadb
   type: database
spec:
 replicas: 1
 template:
   metadata:
     labels:
       app: mariadb
       type: database
   spec:
     containers:
       - name: mariadb
         image: mariadb
         ports:
           - containerPort: 3306
             name: db-port
         env:
           - name: MYSQL_ROOT_PASSWORD
             valueFrom:
               secretKeyRef:
                 name: mariadb
                 key: password
```

EJEMPLO 3

Los Secrets nos permiten guardar información sensible que será codificada. Por ejemplo,nos permite guarda contraseñas, claves ssh, ...
Al crear un Secret los valores se pueden indicar desde un directorio, un fichero o un literal.

```
kubectl create secret generic mariadb --from-literal=password=root
kubectl get secret
kubectl describe secret mariadb
```

Creamos el despliegue y probamos el acceso:

```
kubectl create -f mariadb-deployment-secret.yaml
kubectl exec -it mariadb-deploy-secret-f946dddfd-kkmlb -- mysql -u root -p
```

EJEMPLO 4: Desplegando WordPress con MariaDB

mariadb

```
kubectl create secret generic mariadb-secret \
                            --from-literal=dbuser=user_wordpress \
                            --from-literal=dbname=wordpress \
                            --from-literal=dbpassword=password1234 \
                            --from-literal=dbrootpassword=root1234 \
                            -o yaml --dry-run > mariadb-secret.yaml

kubectl create -f mariadb-secret.yaml
```

Creamos el servicio, que será de tipo *ClusterIP*:

```
kubectl create -f mariadb-srv.yaml
```

Y desplegamos la aplicación:

```
kubectl create -f mariadb-deployment.yaml
```

wordpress

Lo primero creamos el servicio:

```
kubectl create -f wordpress-srv.yaml
```

Y realizamos el despliegue:

```
kubectl create -f wordpress-deployment.yaml
```

Por último creamos el recurso ingress que nos va a permitir el acceso a la aplicación utilizando un nombre:

```
kubectl create -f wordpress-ingress.yaml
```

Los pods son efímeros

Cuando se elimina un pod su información se pierde. Por lo tanto nos podemos encontrar con algunas circunstancias:

1. **¿Qué pasa si eliminamos el despliegue de mariadb?, o, ¿se elimina el pod de mariadb y se crea uno nuevo?.**
2. **¿Qué pasa si escalamos el despliegue de la base de datos y tenemos dos pods ofreciendo la base de datos?.**
3. **Si escribimos un post en el wordpress y subimos una imagen, ¿qué pasa con esta información en el pod?**
4. **En el caso que tengamos un pods con contenido estático (por ejemplo imágenes), ¿qué pasa si escalamos el despliegue de wordpress a dos pods?**